UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Master's Degree in Computer Science

Academic year 2024/2025

CYBER-PHYSICAL SYSTEMS AND IOT SECURITY

Prof. Alessandro Brighente

Written by Michael Amista'

Table of contents

1. Introduction

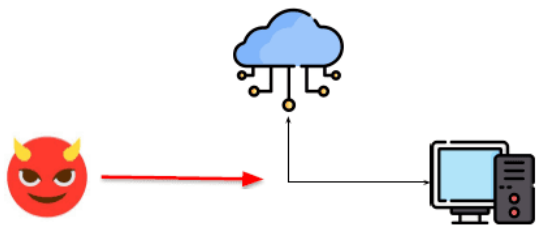
Cyber-Physical Systems (**CPSs**) are characterized by the deep complex intertwining among:

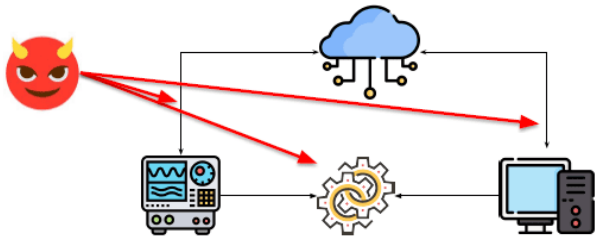
- The world of Operational Technology, or OT (physical / mechanic operations).
- The world of Information Technology, or IT (communication operations).

In CPSs there are devices that create and send information (IT devices – e.g., CAN bus) and some devices that perform physical operations based on the received information (OT devices – e.g. a device that steers the wheel).

Some examples of CPSs are smart grid, smart cars, industrial control systems, e-Health devices.

Traditional IT Systems VS **Cyber-Physical Systems**





In this scenario we have devices connected to the network. In this type of system, the possible attacks are limited with the communication parts and/or the connected devices.

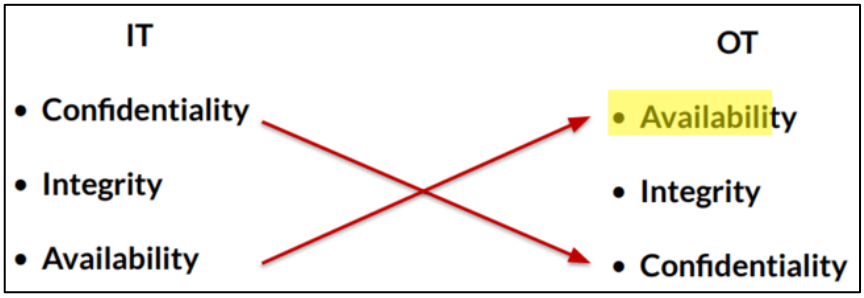
Basic idea: If I want to get the information generated by IT Systems I must access to the device or to the active communication.

In this scenario we still have the IT system structure, but we also have the **OT devices** and some monitoring infrastructure that control their flow (*e.g.,* passive keyless entry system: the monitoring infrastructure needs to check if the car is closed to the user before unlocking the car).

Anyway, **adding new devices to a standard IT system means that the number of breakpoints increases → more possible attacks**. If we don't have security manager, we cannot know if all devices are working correctly.

Security: capability of a system to stand the possible security attacks. More layers that manage the security of the systems in a way to avoid the single point of failure.

According to the literature on cybersecurity, **CIA paradigm** is reversed:



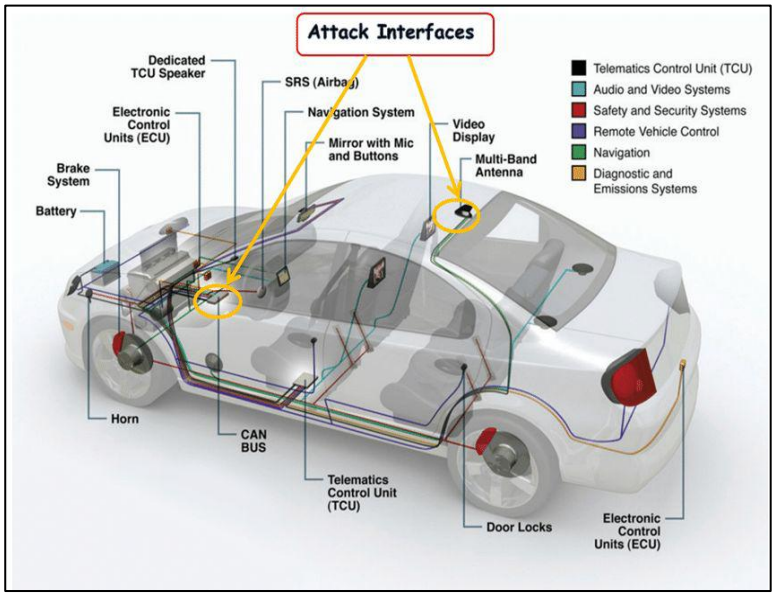
Top → down priority

In our case the most important problem is the **availability of the system**. If nothing is responding, we are not able to control if all devices are working correctly.

The CIA paradigm is defined as follows:

- **Confidentiality**: only the confidential parts of the system should have access to the information. If we are storing customers data, we don't want others to access to this information.
- **Integrity**: the detected information should not be manipulated by others (example: man in the middle).
- **Availability**: we want the usability of the system when requested.

Examples of cyber-physical systems:



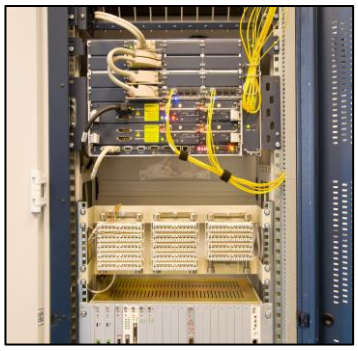
In the car we have some different devices (battery, breaks, small computer, etc.) and these devices are all connected to the CAN bus (Controller Area Network) that permit to connect different electronic devices. **But each CAN bus represent a possible point to attack**.

Another important device/possible breakpoint is the antenna that connect the car to the outside → the car delivers and receive information.

Vehicles platoon

Benefits of organizing vehicles in platoons: Fuel Efficiency; Road Capacity; Road Safety; "Greener".

A vehicle can also communicate with another one. For example: communicate to the platoon that the initial vehicle is breaking so the other ones can break.



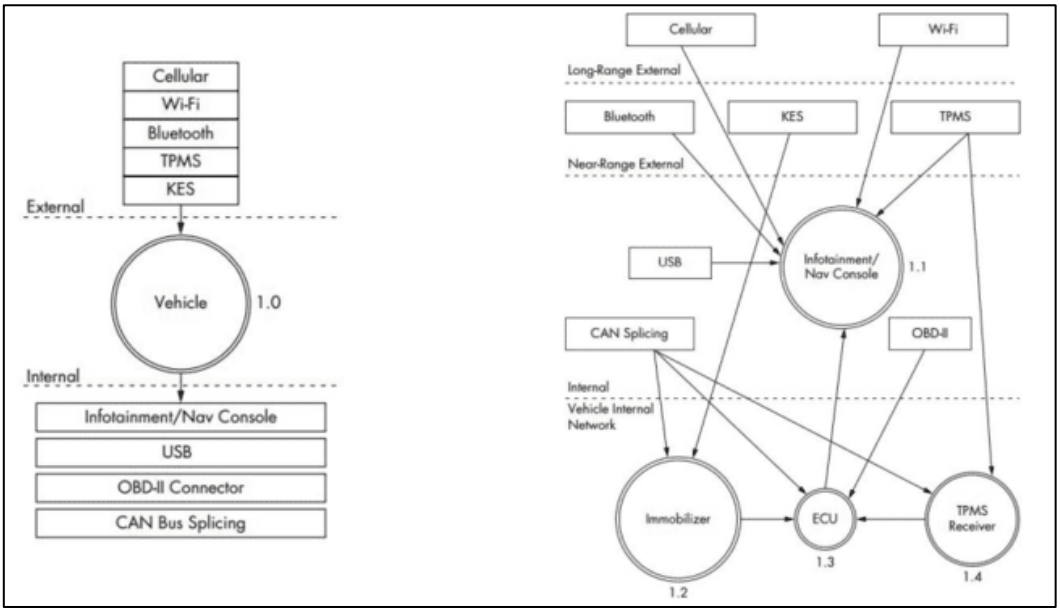
Industrial Systems

- have dedicated networks and components;
- operational technology + information technology;
- dedicated computing systems.

IoT devices are resource constrained and battery limited devices. Data provided by IoT devices can be altered by an attacker and since IoT devices manage important systems it is crucial to consider even in this scenario the possible attacks.

2. CAN bus

What is there in a car? A lot of different interconnected devices:



Modern vehicles contain a large number of **Electronic Control Units** (**ECUs**). ECUs are embedded systems that control one or more (sub)system in a car or other vehicles and connected to the CAN bus. Examples: engine control module, powertrain control module, transmission control module, suspension control module.
ECUs are nodes in a rather complex in-vehicle network, where they should communicate one another to report many different types of information.

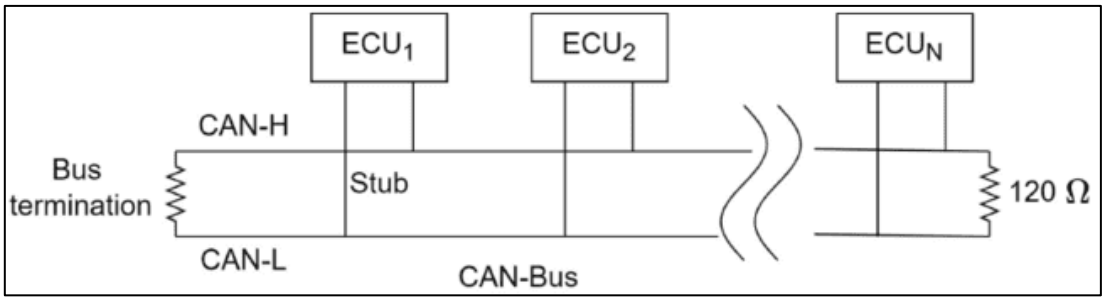
Controller Area Network (**CAN**) is an in-vehicle network bus-based standard developed in 1986 by Bosch. It allows in-vehicle components (ECUs) to communicate one another. It has broad application in automotive systems, including power train, suspension, and braking. The problem in CAN bus communications is the security, we do not have any implemented control. The Society of Automotive Engineers (SAE) standardized CAN bus communications to have asynchronous data rate up to 1 Mbps at a 40 m distance. Maximal cable lengths should be between 500 m (125 kbit/s) and 40 m (1 Mbit/s).

Why CAN bus:

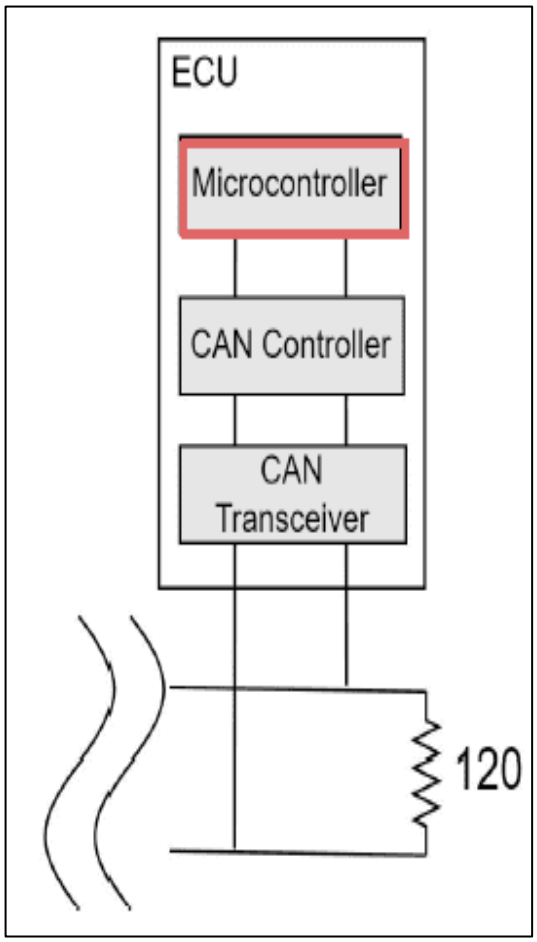
- **Robustness**: ideal for safety applications thanks to its durability and reliability (bit stuffing, bit monitoring, frame check, ack and CRC checks).
- **Low cost**: objective is to reduce errors, weight, wiring and costs.
- **Flexibility**: it is a message-based protocol, so nodes can be added or removed without updating the system. First security problem: the system is easily reconfigurable.
- **Efficiency**: messages with high priority are clearly marked and have prioritized access to the bus.

To achieve reliability, it is important to prevent message collision such that no data gets lost. Furthermore, thanks to the shared bus and the message handling system, CAN allows for a reduced number of wires to achieve reliability. Redundancy is the best way to achieve reliability, however it comes with a higher communication implementation cost (huge number of wires). Point-to-point communications were used before CAN bus but since modern vehicles have a large number of ECUs a lot of wired connections are costly and difficult to place efficiently in a car.

CAN BUS ARCHITECTURE



ECU ARCHITECTURE



MICROCONTROLLER: the host processor (also called *microcontroller*) decides what the received messages mean and which messages it wants to transmit. Sensors, actuators and control devices can be connected to the host processor.

CAN CONTROLLER:

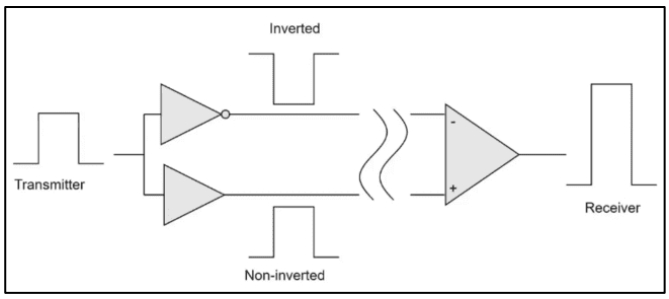
- **Receiving**: the CAN controller stores the received serial bits from the bus until an entire message is available, which can then be fetched by the host processor (usually by the CAN controller triggering an interrupt).
- **Sending**: the host processor sends the message(s) to transmit to a CAN controller, which transmits the bits serially onto the bus when the bus is free.

CAN TRANSCEIVER:

- **Receiving**: it converts the data stream from CAN bus levels to levels that the CAN controller uses. It usually has protective circuitry to protect the CAN controller.
- **Transmitting**: it converts the data stream from the CAN controller to CAN bus levels.

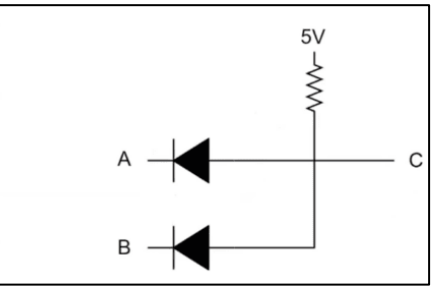
2.1 Data transmission

CAN bus uses **differential wired-AND signal**. ECUs emit two signals (two voltage values) to send a message: CAN High (CAN-H) and CAN Low (CAN-L). They are either driven to a *dominant state* with CAN-H > CAN-L or driven to a *recessive state*, with CAN-H ≤ CAN-L.



Differential: once the signal is emitted an inverted signal is computed and the received signal corresponds to the original signal minus the inverted signal. This is done to avoid interference, ensuring robust transmissions.

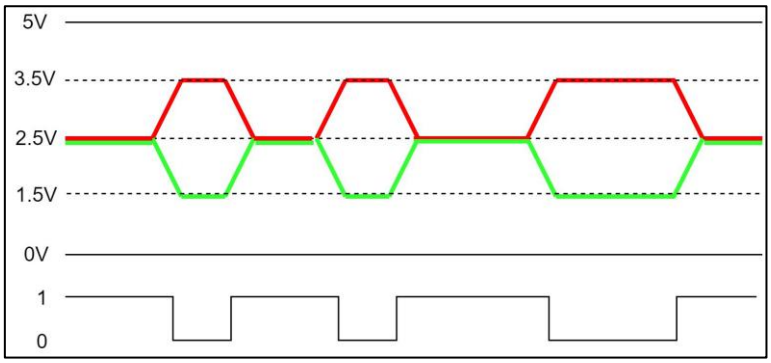
Input		Output
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



Wired-AND signal: basically, we are implementing the AND operation considering the two different voltages values applied to CAN-H and CAN-L lines and computing so the transmitted bit.

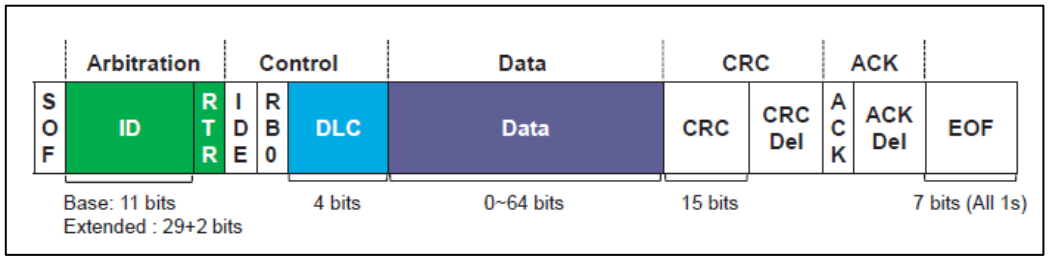
The CAN specifications use the terms **dominant bits** and **recessive bits**, where **dominant is a logical 0** (actively driven to a voltage by the transmitter) and **recessive is a logical 1** (passively returned to a voltage by a resistor). The idle state is represented by the recessive level.
By using this process, any node that transmits a logical 1, when another node transmits a logical 0, loses the arbitration and drops out. A node that loses arbitration re-queues its message for later transmission and the CAN frame bit-stream continues without error until only one node is left transmitting. Since the 11 (or 29 for CAN 2.0B) bit identifier is transmitted by all nodes at the start of the CAN frame, the node with the lowest identifier transmits more zeros at the start of the frame, and that is the node that wins the arbitration or has the highest priority.

Here below an example of a CAN bus transmission using two different voltage values corresponding to CAN-H and CAN-L (here highest voltage value corresponds to bit 1 and the smallest to bit 0, as you can see when the bit 1 collides with bit 0, the 0 wins).



CAN FRAMES

The standard CAN frame is composed by 8 message fields. CAN 2.0A and 2.0B differ in the length of the ID field (11 vs 29 id bits).



- **Start of Frame (SoF)**: dominant 0 to tell the other a node wants to talk.
- **ID**: frame identifier. The lower the ID, the higher the priority. **It is not an identifier of the sender** (know the sender is not so important in broadcast communications like the one implemented in CAN bus).
- **Remote Transmission Request (RTR)**: indicates whether a node sends data or requests data from another node.
- **Control**: contains the Identifier Extension Bit (IEB) which is dominant 0 for 11 bits. It also contains the 4 bit Data Length Code (DLC) that specifies the length of the data bytes to be transmitted (0 to 8 bytes) → it tells us how much data we can transmit with this frame.
- **Data**: payload in common network terminology. CAN signals that can be extracted and decoded for information.
- **CRC**: check for data integrity. Not used for security purposes but only to check whether the received packet is the correct one (e.g. "if I send a command to brake, I want the car brakes").
- **ACK**: indicates whether a node has acknowledged and received data correctly.
- **End of Frame (EoF)**: end of the CAN frame.

VALIDITY OF THE CAN FRAME

CAN frames need to satisfy certain conditions to be valid. If a node detects its transmission of an erroneous message, it takes actions accordingly. This is the **CAN bus error handling mechanism**, and each node has its own error counter and state. Depending on its counter value, each node can assume the following states: {active, passive, bus off}.

An ECU transmits with certain voltage values to define the message to send, and the ECU can communicate these values to be sure that on the other part everything received has no errors. In this way it is possible to understand if the error originates from the sender or not.

Counters are **Transmit Error Counter (TEC)** and **Receive Error Counter (REC)** for each node:

- TEC = TEC + 8 if an error occurs during transmission (ECU fault)
- REC = REC + 1 if an error in the reception (not ECU fault)

TEC increases faster than REC because when the error originates from the transmitter node there might be some problem on that ECU, and it is better to signal it to avoid further errors. Success decreases the counter by one in both cases.

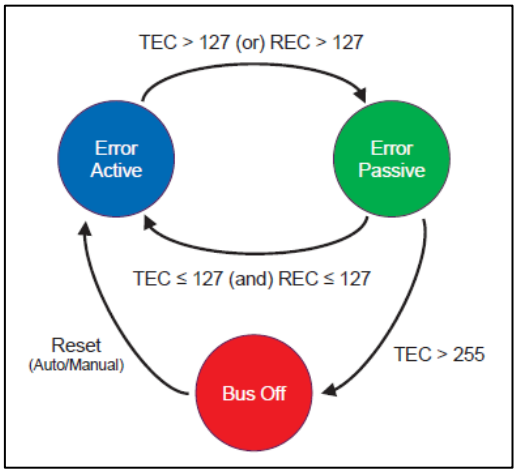
In CAN bus transmissions we can have five types of errors checked by CAN controller:

1. **Bit error**: occurs when an ECU, after comparing its transmitted bits with those on the CAN bus, detects a mismatch (errors in converting bits to voltage values and vice versa). This does not happen during CAN arbitration.
2. **Stuff error**: after transmitting five consecutive same polarity-bits, the ECU sends an opposite bit to maintain soft synchronization. An error occurs if stuffing does not happen.
3. **CRC error**: computed one different from the received one (hashing code comparison).
4. **Form error**: if a fixed-form bit field (e.g. EoF) has a bit error the ECU raises a form error.
5. **ACK error**: when a node receives a frame it responds with a dominant bit in the ACK frame. If none respond, then ACK error is erased.

When an error is detected, there are two types of error flags that can be transmitted on the bus:

1. **Active Error Flag**: six dominant bits – transmitted by a node, that is in error active state, detecting an error frame on the network.
2. **Passive Error Flag**: six recessive bits – transmitted by a node, that is in error passive state, detecting an error frame on the network.

These error flags are transmitted to report the other nodes that the message transmitted on the CAN bus contains an error. The transmitted frame causes other nodes to detect a bit-stuffing violation, prompting them to generate their own error frames in response to the stuffing error, and to stop any ongoing transmissions or receptions.



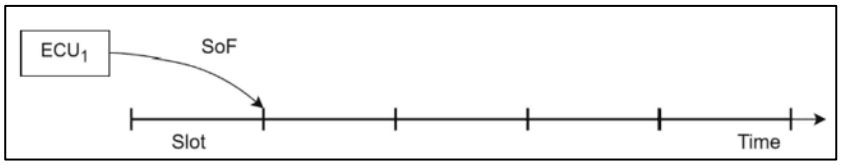
When an ECU transitions to the bus off mode it means that something serious is happening. *Limp home state*: to prevent accidents but still allow the driver to reach a safe place, the ECU in bus off usually runs with predefined parameters and reduced functionality. To return to error active mode, a reset or 128 occurrences of 11 consecutive recessive bits is needed.

BUS ARBITRATION

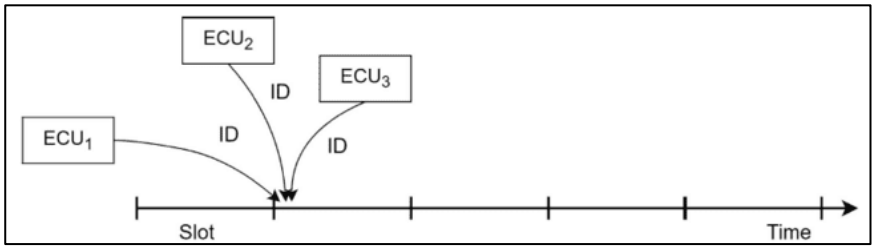
It refers to the process by which nodes get a share of the bus resources to transmit. CAN arbitration protocol is both **priority-based** and **non-preemptive**. The media access protocol alternates contention and transmission phases.

Non-preemptive: a message cannot be preempted by a higher priority one if this was queued after transmission began.

Time is divided into slots, each with a length sufficient for a message to go back and forth in the whole bus. A node wishing to transmit checks whether the channel is idle.



If bus is idle, ECU waits the next slot and starts a contention phase by transmitting a SoF bit.



All other nodes wishing to transmit send their ID. The CAN controller compares its output with the actual bus level at the end of each bit cycle.

S1	1	0	0	0	1	0	0	0
S2	0	0	1	1	1	0	0	1
S3	0	0	1	1	1	0	1	0
Bus	0	0	1	1	1	0	0	1

S1 immediately loses, it begins with a recessive bit. Proceed with S2 and S3 bit comparison and S2 wins the arbitration at the 7th bit.

After the node that wins the competition terminates its transmission with an EoF, there should be new transmissions. The CAN bus uses a 3 bit inter-frame symbols to separate packets. After this, the bus becomes idle again and arbitration shall commence again.

2.2 CAN bus access and attacks

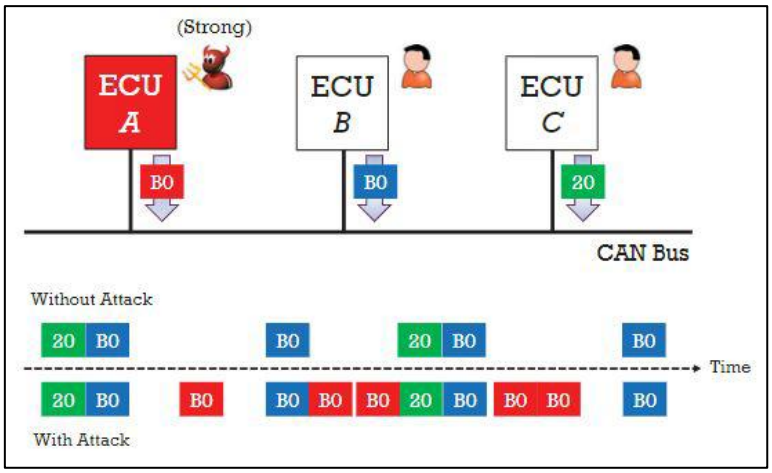
To access CAN bus, you need to connect to the On Board Diagnostic (OBD) port. Basically, all cars nowadays use the OBD-II standard port. It is usually located near the driver's or passenger's seat. To talk with the CAN bus you need a converter, such as in the figure. This converts CAN bus data to something readable via a USB port.

Attacker Model

- Attacker can physically and remotely compromise more than one in-vehicle ECU in different means.
- The attacker objective is to manipulate or impair in-vehicle functions.
- Two means:
  - inject (attack) messages with a spoofed ID.
  - stop/suspend message transmission of the compromised ECU.
- Furthermore, ECUs can be **weakly** or **fully** compromised. Weakly compromised means an attacker cannot inject any fabricated message.

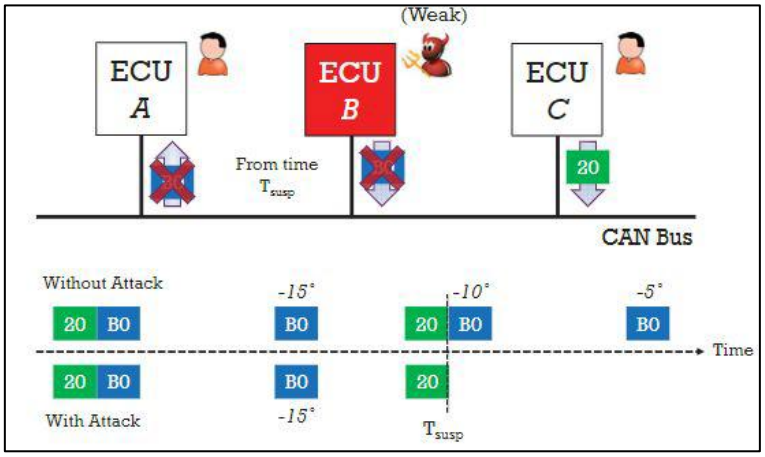
Fabrication Attack

- Requires a compromised ECU as a strong attacker to fabricate and inject messages with forged ID, DLC, and data.
- Objective: override any periodic messages sent by a legitimate safety-critical ECU so that the receiving ECU gets distracted or become inoperable.



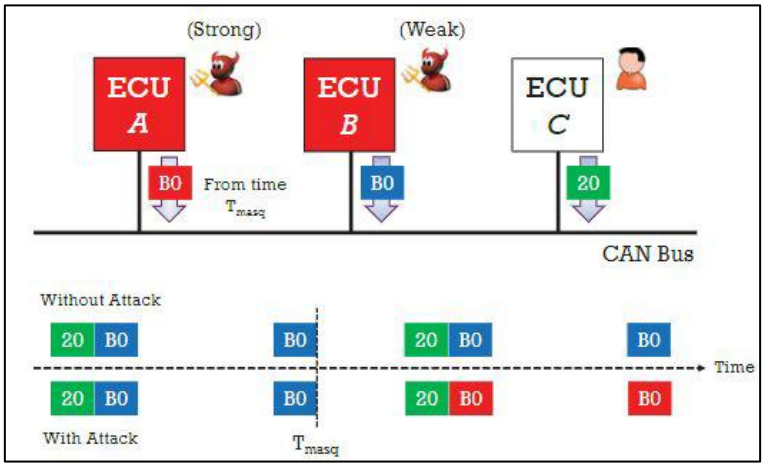
Suspension Attack

- Requires a single weekly compromised ECU.
- Objective: stop/suspend the weakly compromised ECU's message transmission, thus preventing the propagation of info to other ECUs.
- Detrimental also for ECUs that rely on the reception of such messages.



Masquerade Attack

- Need to compromise two ECUs, one as strong and the other as weak.
- Objective: manipulate an ECU while shielding the fact that an ECU is compromised.
- The attacker first monitors the traffic of the weak ECU.
- Once obtained the transmission period, stop the weak and use the strong instead.



The bus-off attack is a **Denial-Of-Service (DoS) attack** that exploits how ECUs access the bus. The error handling mechanism of CAN bus forces ECUs that measure a certain number of errors to go bus-off ➔ **the attacker exploits this feature to iteratively isolate ECUs**. Injecting attack messages, the adversary forces the TEC of an uncompromised/healthy victim ECU to continuously increase. Eventually, the attacker forces the error confinement to force the victim or even the entire network to shut down.

Bus-Off Attack: Adversary Model

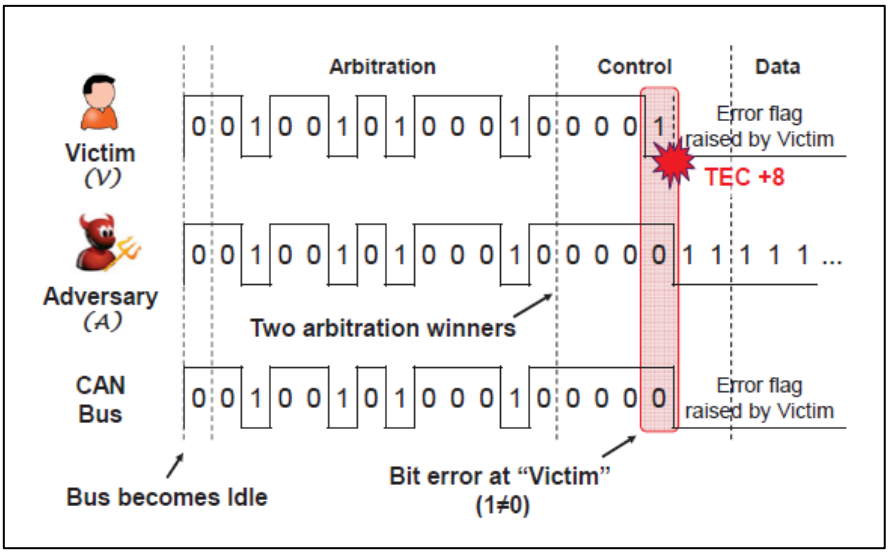
- The attacker can compromise an in-vehicle ECU either physically or remotely to gain its control.
- We do not require the adversary to reverse engineer messages or checksums to be able to deliver the attack. In fact, messages have different structures depending on the manufacturer, but this is not central in bus-off attacks.
- Once the ECU is compromised, the attacker can inject messages with any ID, DLC and data on the CAN bus.

2.2.1 The Bus-Off attack

Suppose that a victim V periodically sends a message M. The attacker can hence deliver a successful bus-off attack if all the following conditions are satisfied:

1. **C1: using the same ID**. In this way there is no winner and both ECUs have the same priority, causing an error on the bus (what an attacker wants to exploit).
2. **C2: transmitting at the same time as M** (the tricky part of the attack). If the attacker transmission is not aligned with the victim the attack fails.
3. **C3: having at least a bit that is dominant in the attack message while being recessive in M** (all preceding bits must be equal).

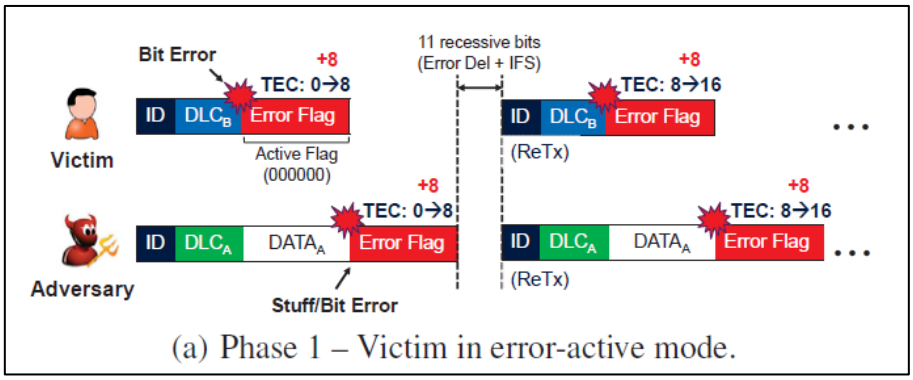
The first step for an attacker is to observe and listen the channel to collect IDs and observe the messages periodicity, then it is possible to plan the attack using the gathered information.



Phase 1: Victim in Error-Active

- Both adversary and victim start in error-active mode, and the attacker targets one of the victim's periodical messages.
- The attacker sends the malicious message and the victim's TEC increases by 8.
- The attacker's TEC also increases by 8, due to the bit errors triggered at the adversary node (active error flag transmitted by the victim that violates the bit stuffing rule).

- Both nodes **automatically retransmit the failed message**.
- Thanks to the automatic retransmission, the attacker brings the victim to error-passive by sending a single message.

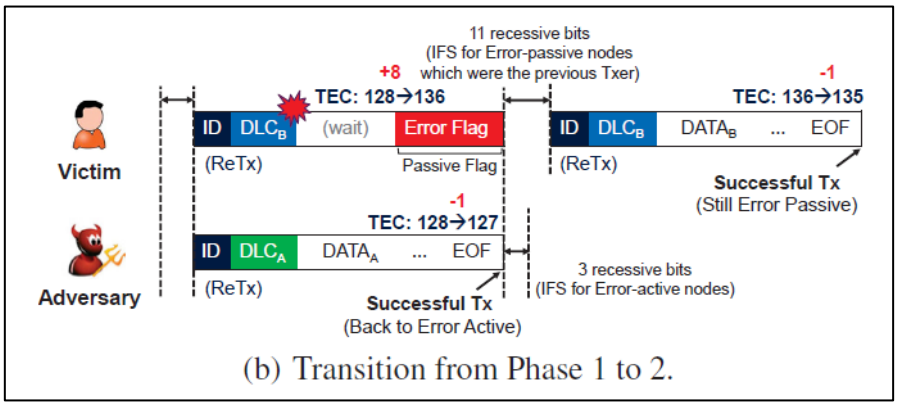


(a) Phase 1 – Victim in error-active mode.

Note: the victim's error counter increases due to failed transmission of packet while attacker has an error due to the victim's active error flag (6 dominant bits) that blocks the adversary transmission. So, the **victim and the attacker TECs increase at different times and for different reasons**. This is fundamental for the success of the attack.

Phase 1 to 2

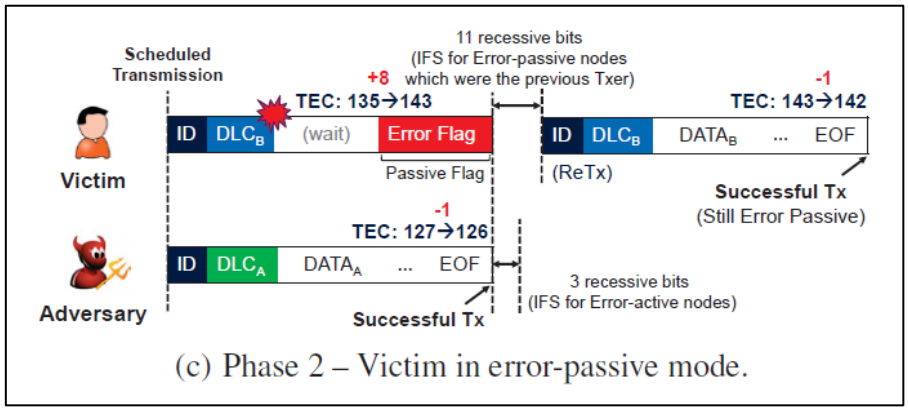
- After 16 retransmissions, both victim's and attacker's TEC = 128 → error-passive.
- Therefore, the victim attempts the delivery of a passive error flag with 6 recessive bits.
- **The passive error transmitted by the victim cannot override what is in the CAN bus because recessive bits cannot override dominant ones**. So, the attacker successfully transmits its message and return to error active (attacker's TEC = TEC - 1).
- Due to the successful transmission, the adversary does not go to error-passive.
- The victim can transmit later the error passive frame decreasing its TEC by 1 but remaining in error passive mode.
- Note: when the adversary node receives the victim's passive error frame, its TEC does not increment since TEC increments whenever a node transmission is disrupted. During the phase 1 adversary's TEC incremented because the error frame sent by the victim contained 6 dominant bits which override every frame on the bus, even the one the adversary had to retransmit.



(b) Transition from Phase 1 to 2.

Phase 2: Victim in Error-Passive

- Once the scheduled interval of the target message is elapsed, V retransmits M again.
- At the same time, the adversary reinjects malicious M.
- Since the victim is in error-passive, the attacker's TEC decreases by 1, whereas the victim's increases by 7 (+8 -1) thus maintaining the error-passive.
- The attacker iterates until the victim's TEC > 255, forcing the victim ECU to bus-off state.



(c) Phase 2 – Victim in error-passive mode.

Note: remember that CAN ID do not contain actual information on the transmitter. The ID contains information that define the time interval and priority of messages. The attacker should hit low-value IDs (dominant) to disconnect possibly safety-critical ECUs.

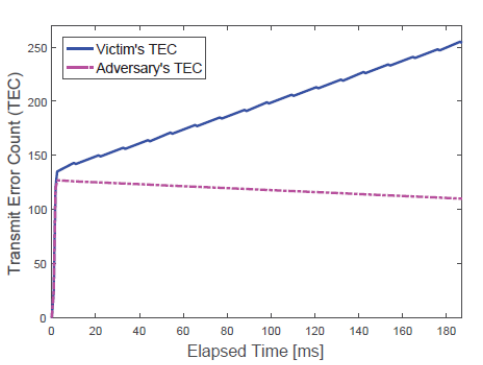


Figure 12: TECs during a bus-off attack.

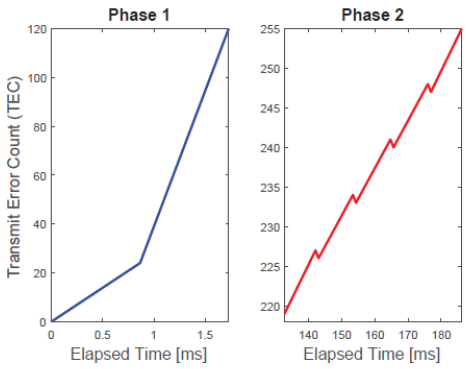


Figure 13: Victim's TEC during Phase 1 and 2.

Where to modify frames?

To trigger a bit error at the victim and increase its TEC, the adversary's attack message must satisfy C3: it must have at least one bit position where its signal is dominant (0) while the victim's is recessive (1), with all preceding bits being identical. Since the bus-off attack requires the attack and target messages to have the same ID (C1), this mismatch must occur in either the **control** or **data field**, as fields like CRC and ACK are managed by the CAN controller and not controllable by the attacker. A practical and effective way to achieve this is for the adversary to set its message's DLC or data values to all 0s. Given that the DLC for a specific CAN ID is typically constant over time, the attacker can observe and match this value to craft a message that satisfies C3, ensuring a dominant-recessive bit mismatch in the victim's transmission.

DIFFICULTIES OF THE BUS-OFF ATTACK

ECUs cannot acquire the IDs of all received messages. Only of those that pass through its message filter. We can distinguish hence between *received message* (the ECU can only see and read the content) and *accepted message* (messages that contains some information related of what the ECU has to do, for instance some physical operations to perform).
The attacker can only read the ID from accepted messages, and he can create only messages with the same ID depending on the filter of the victim ECU → if there is no filter then done, but filters can be remotely manipulated.

Non-Empty Message Filter

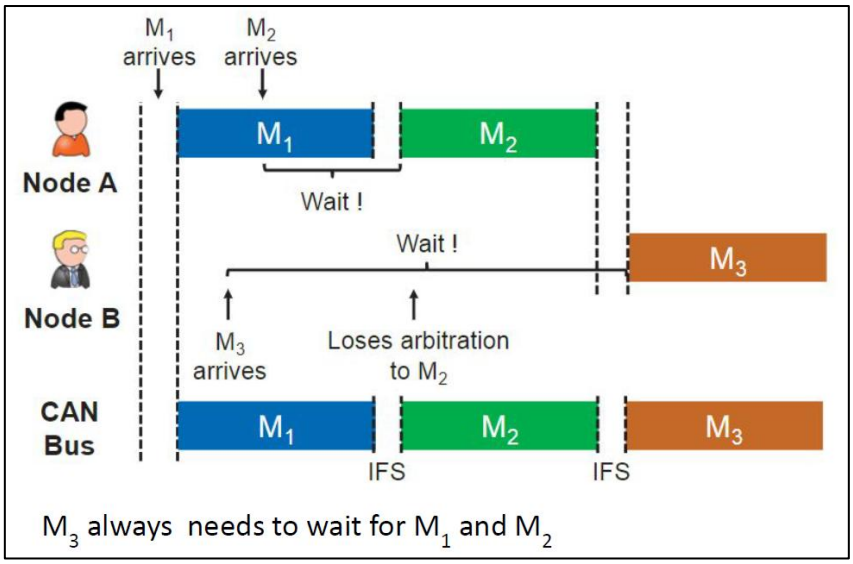
Instead of what Empty Message Filters do, here the compromised ECU only accepts messages with a certain (range of) ID. This however does not represent a restriction for the attacker in compromising it. The attacker can directly modify the message filter of the compromised ECU via software commands. For some controllers, the configuration mode can be triggered via user instruction through the Serial Peripheral Interface.

Another problem of the bus-off attack is that we need **precise synchronization**, so if an attacker does not transmit the malicious packet as soon as the victim is transmitting its packet then the attack is not going to work. It is possible to achieve the synchronization without taking care about the exact time of the victim's transmission: **all messages and priorities are periodic, and we can exploit this to infer when a certain message is going to be transmitted** (preceded message). Usually when a packet's transmission in the CAN bus ends, we wait for 3 bit-time before starting the transmission (IFS).

PRECEDED ID

Although CAN messages are periodic, **jitter** (time variation on sending messages) **may affect the effectiveness of the attack**. To overcome this issue the attacker can exploit the fact that nodes, which have either lost arbitration or had new messages buffered while the bus was busy, attempt to transmit their messages as soon as the bus becomes idle. Given a node with ID M, we define its **preceded ID as the ID of the message that was transmitted immediately before the one the attacker is targeting**.

Since the order of message transmission is determined by the priority of the IDs, and IDs are fixed, the preceded ID provides a reliable indicator for the attacker. By knowing the ID of the message that comes right before the victim's message, the attacker can better predict when the target message will start → Transmission time = 3 bits after the completion of the message with preceded ID. If the target message has no preceded ID, the attacker can create one and send both the preceded ID and the attack message.



Consider an example where node A transmits messages with ID=M1, M2, and node B transmits a message with ID=M3 which has the lowest priority among them. As shown in the above figure, if these messages are arriving and being queued at the depicted times, M1 and M2 would be the preceded IDs of M2 and M3, respectively, with only a 3-bit IFS separating the corresponding message pairs. In other words, the transmissions of M2 and M3 are forced to be buffered until their preceded ID messages have been transmitted on the bus. Since message priorities and periodicities do not change, such a feature implies that **one specific CAN message may always be followed by another specific message, i.e., there is a unique preceded ID for that specified one**.

As an example, if the periodicities of their transmissions are either same or integer multiples (e.g., 5ms for M1, M2, and 10ms for M3), then M2 would always be the preceded ID of M3, i.e., be the unique preceded ID. Hence, regardless of jitter, the exact timing of message transmissions becomes rather predictable and even determinative: 3 bit-time after the preceded ID's completion.

Fabricated preceded ID

Even though the target message does not have such a preceded ID, an adversary can fabricate it to synchronize the timing and thus succeed in mounting the attack.

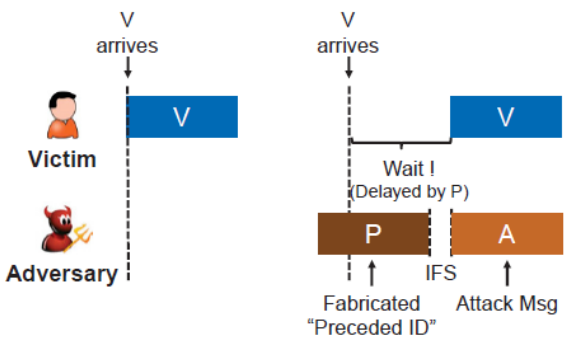


Figure 7: Fabricated preceded ID.

Consider an example where a victim node periodically transmits message V, which has no preceded IDs. In such a case, just before the transmission of V, the adversary can inject some message P and an attack message A, sequentially. Hence, V's transmission gets delayed until the completion of P, i.e., the adversary fabricates P as the preceded ID of V, and thus the attack message is synchronized with its target.

Now we will focus on how an attacker can predict the exact transmission time by considering how jitter may affect the synchronization.

The injection timing, quantity, and content of fabricated preceded ID messages is fundamental for success. Regarding timing, it is fundamental to inject the message right before the target one and we still have the problem with jitter over periodicity. The only source of randomness is jitter, which follows a Gaussian distribution with zero mean and sigma standard deviation. The successive times at which the attacker receives the victim message can be expressed as:

victim transmit time — expected transmit time — delay

$$t_n^{adv} = t_n^{vic} + \mathbb{D} = t_{orig} + J_n + \mathbb{D}$$

$$t_{n+1}^{adv} = t_{n+1}^{vic} + \mathbb{D} = t_{orig} + T + J_{n+1} + \mathbb{D},$$

expected transmit time second message

where J_n is the jitter random variable. We can then compute the next victim's transmit time as

only this is a random variable

$$t_{n+1}^{vic} = t_n^{adv} + T - \mathbb{D} + J_{n+1} - J_n = t_n^{adv} + T - \mathbb{D} + J^*,$$

For fabrication of the preceded ID, the attacker has to start the transmission of its preceded ID messages(s) before the target and make the bus busy until it becomes sure that the attack and target messages would synchronize.

- We can hence write the following constraints for the attacker

start of sending fab. msg.

$$1)\ t_{fab} < \min(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \min(J^*)$$

$$2)\ t_{fab} + \mathbb{H} > \max(t_{n+1}^{vic}) = t_n^{adv} + T - \mathbb{D} + \max(J^*)$$

$$3)\ \mathbb{H} = \kappa\mathbb{F} > \max(J^*) - \min(J^*),$$

duration of bus holding — duration of preceded ID msg.

- J* is a bounded random variable, with boundaries approximated as $|max(J^*)| = |min(J^*)| \simeq \mathbb{I}\sqrt{2}\sigma_v$

attack parameter — measurable

- Setting I=3 provides 99.73% confidence interval and I=4 a 99.99% confidence interval
- To fully exploit the fabricated preceded IDs, the adversary needs to start injecting them prior to $t_n^{adv} + T - \mathbb{D} - \mathbb{I}\sqrt{2}\sigma_v$

- To satisfy the third constraint, the adversary needs to occupy the bus for a duration of $\max(J^*) - \min(J^*) = 2\sqrt{2}\mathbb{I}\sigma_v$
- This can be done by sending 1 or more preceded ID messages
- An adversary aiming at minimizing the number of injections should maximize F
- To this aim, the adversary can exploit the bit stuffing rule and send messages with the most stuffed bits
- With L=DLC = 8, $\mathbb{F}^* = (8L + \boxed{44} + \lfloor 8L/4 \rfloor)/S_{bus} = 124/S_{bus}$

  exterior to data

- Based on this, the number of preceded ID need can be expressed as $\kappa = \left\lceil \frac{\max(J^*) - \min(J^*)}{\mathbb{F}^*} \right\rceil = \left\lceil \frac{2\sqrt{2}\mathbb{I}\sigma_v S_{bus}}{124} \right\rceil$

- For sigma = 0.025ms, S_{bus}=500Kbps an adversary only needs 1 message with I=3 and hence 99.73% confidence
- To raise confidence to 99.99% (i.e., I=4) the adversary needs two messages

Content of Preceding ID Messages

- The adversary needs to decide which ID to use for fabricating the preceded ID messages.
- If only one preceded ID is needed, the attacker can use the next seemingly free ID.
- To be as elusive as possible, such ID can be changed at every attack attempt choosing among the least frequently sent on the bus.
- If two preceded IDs are needed, the adversary can choose one from the pool as before, but carefully select the second one to have higher priority than the target one.

Preventing Bus-Off Attacks

- The defence mechanism can leverage two features of the bus-off.
- **F1 (in phase 1)**: at least two consecutive errors occur during the transmission of frames. Thus, we watch for consecutive error frames with an active error flag.
- **F2 (in phase 2)**: at the time when the error-passive victim's TEC increases, a message with the same ID will be successfully transmitted by some ECU on the bus.

2.2.2 WeepingCAN attack

The Bus-Off attack is easily detectable by a good intrusion detection system (the malicious message is always the same). **WeepingCAN** is a variation of the original bus-off attack, stealthier and more difficult to identify.

Differences from standard Bus-Off attack:

- The attacker disables the automatic retransmission of the attack message with the same ID as the victim.
- The attacker causes recessive bit errors.
- The attacker does not fabricate preceded ID messages.
- The attacker randomizes bit errors. Bits are changed randomly.

The attack message has the victim's ID, bit-time, and prefix bits until a random position where the victim is dominant, but the attacker is recessive. As for the original bus-off, both attacker and victim have TEC = 0 at the beginning. The goal of weepingCAN is to **avoid exhibiting temporal differences due to the error state transition**.

Attack Cycle

- The attacker synchronized with the victim using the same approach of the original bus-off.
- The attacker however disables its retransmission of the packet when errors occur.
- The attacker sends the attack message with the same ID and timing as the victim's message but injects a **random bit error** at some point, causing both the attacker and victim to detect a transmission error.
- The attacker's CAN controller sends an error-active flag due to the bit error in the attacker's packet, which increases both attacker's and victim's TEC by 8.
- Since the attacker has disabled retransmission, the victim successfully retransmits its original message without further interference, decreasing its TEC by 1.
- Victim and attacker decrease TEC for other transmissions.

Disabling retransmission allows to remove an easily detectable feature, i.e., the consecutive retransmissions of the same packet. The attack is now composed by an active error + a successful retransmission. Two ways to avoid retransmissions:

- Disable automatic retransmissions for all messages (via control register): the attacker writes to a control register that the ECU should never retransmit, neither with arbitration.
- Abort transmission on transmit error (via interrupt): most CAN controllers can raise an interrupt in transmitting errors and allows to abort retransmission in input handler.

Recessive Injection

- The attacker injects a recessive bit when the victim "contains" a dominant bit.
- The attacker increases its TEC by 8 and sends an error active flag.
- The victim increases its TEC, then successfully retransmits (TEC = TEC - 1).

- The attacker must identify additional messages it can transmit to keep its TEC lower than that of the victim (otherwise could go bus off!).
- For every attack message the attacker sends, it must transmit at least one additional successful message to keep its TEC in check. Meanwhile, the victim should transmit fewer than seven other messages in the same time frame; otherwise, its TEC may recover and surpass the attacker's TEC.

Basic attack strategy

- Suppose that the attacker and victim both have a single message they can send.
- Suppose A sends its message five times more frequently than the victim sends its message.
- Thus, for every attack message, A transmits 5 messages while V transmits 1 → TEC_A=3, TEC_V=7.
- V can be forced to error passive with 19 attack messages.
- A needs to be careful not to reach error passive before V goes bus-off!
- A can decide to skip some injections if its TEC is increasing faster than the V's TEC. Meanwhile it can transmit some successful messages to decrease its TEC.

Randomized bit position

The second feature making bus-off identifiable is the presence of successive messages over the passive error flags of the victim. Furthermore, the attacker may always use the same packet for the attack. Therefore, it might be good for the attacker to have the possibility to randomize packets.
In particular, to randomize the position of the error bit. The DLC of most messages is a fixed constant that can be discovered by offline analysis of a CAN trace. The number of dominant bits in DLCs vary between 1 and 3, so the entropy is low → bad! A clever attacker may inject recessive error in the data field.
The data field often encodes a number that does not change quickly because of physical constraints or an event identifier that comes from a limited set of events. If A can identify a deterministic pattern, it can inject error in the data.

What About Bit Flipping?

- The idea of injecting errors via controlling CAN frames requires synchronization with the victim and carefully crafting attack frames.
- In the end, the whole purpose of the attacker is to flip some bits on the can BUS to lead the victim ECU to bus off.
- With minimum 32 injected fake frames (or bits?) we can force an ECU to bus off.
- If the attacker can gain control over the bus, they can just set arbitrary bits and cause errors.

Silently Disabling ECUs

- Suppose we include an IDS in our CAN bus that can detect arbitrary IDs appearing on the bus.

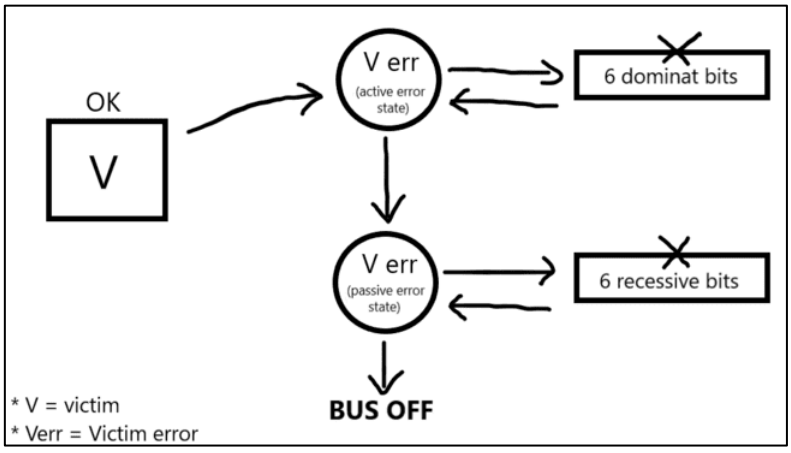
- It contains a whitelist of all arbitration IDs the ECUs on the bus are programmed to transmit.
- Such a whitelist allows us to detect but flip attackers attempting to cheat the arbitration process or transmit never seen before message (needed for preceded ID).

Attacker Model

- A stealthy attacker is able to read and write on the bus, with the goal of disabling and spoofing ECUs without being detected.
- Flipping bits is a non-precise technology, so the attacker has a chance from 0-100% of flipping bits 0⟶1 and 0-100% of flipping 1⟶0.
- Remember that 0 is dominant, so if anyone is transmitting a 0 the 0 wins.
- We make no assumption about the attack vector, as long as it can flip bits.
- We can also consider a blind attacker, which has no read access to the bus, only write.

Stealthy Bus-Off

- The original bus-off attack produces errors on the bus by performing a single 1 to 0 flip.
- Each attack increments the error counter by 8 → doing it 32 times over 32 messages force the victim to bus-off.
- We want to design an attacker node with the same objective but that does not produce errors on the bus.
- We also assume that attacker can arbitrarily flip bits rather than needing to win the arbitration process. Here we want to do something different from the standard bus off attack: we want to transmit erroneous bits not only in CAN frames but also on other transmissions as error flags.
- Main idea: when a CAN error frame is interrupted, it produces another error frame incrementing its TEC by 8 every time it is interrupted. This means that an attacker able to flip bits can force to victim to bus-off with a single message by flipping 32 bits.
  Once the victim is in bus-off, it will stop sending error frames although any of them have actually been delivered to the bus.
  We need to ensure that no receive errors are produced and ensure the rest of the bus processes eventually sent malicious messages (spoofing the victim).



The above schema explains the main goal of this alternative approach where the attacker causes errors during error flag transmissions (changing bits in the error flag and so violating bit stuffing rule). This leads the victim initially in error active mode, then in error passive and finally in bus-off state.

Probabilistic Attacker

- Flipping individual bits on a serial bus is inherently a hard problem, particularly changing the dominant state to a recessive state.
- The attacker hence must be able to:
    - successfully complete a CAN packet (11 flips in a row – EOF, IFS,delim)
    - successfully interrupt an error frame (at least 1 bit flip in 6)
    - transmit a known arbitration ID (for blind attackers)

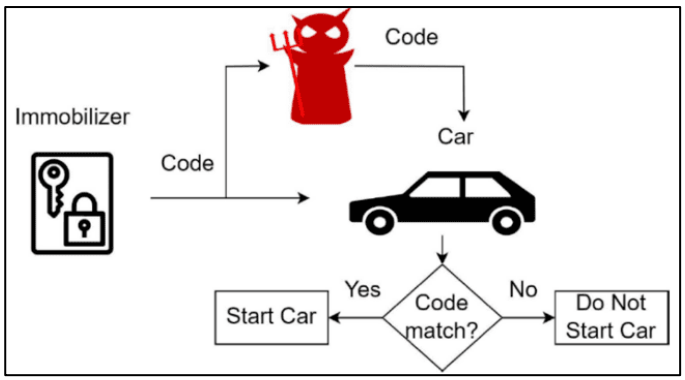
Interrupting an Error Frame

- The ability to interrupt an error frame represents the ability of the attacker to continue transmitting a message without letting an error frame complete and have the victim message be dropped.
- This translates into flipping at least one every 6 bits.
- First caveat: ensure that receiver errors (RCR, ACK, stuffing, frma echeck) do not occur.
- Second caveat: interrupting another ECU attempting to transmit a message.
- Both of them are avoidable by having the ECU in bus-off by the arbitration.

3. Keyless Car Security

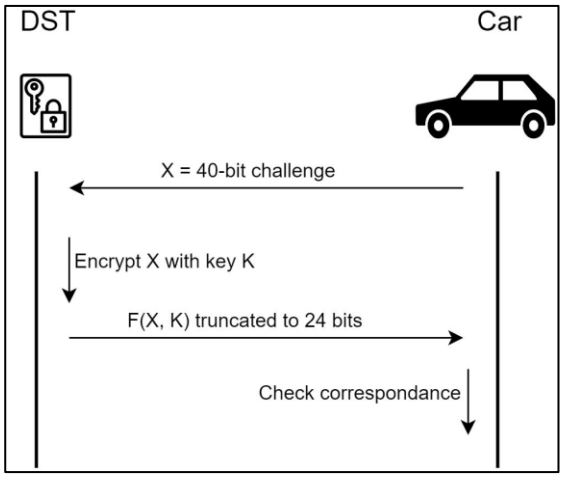
At the beginning, there were ignition systems. Ignition switch is the first step to get a car to start: turning the key sends a signal. This signal starts the ignition system and ignites the fuel vapor. This system has a critical flaw: **you can generate this signal in many ways** if you have physical access to the car and without using the original keys.

The first solution involving a cyber-component to enforce security is the **immobilizer**. The first generation of immobilizers used a small chip embedded into the head of the car key. When the driver inserts the key into the cylinder, the chip emits a code/serial number that can be received by the antenna inside the cylinder. If the code matches the one the car expects, then ignition starts.



PROBLEM: **the immobilizer always transmits the same code**. An attacker with an eavesdropping equipment can easily record the code and later replay it when stealing the car.

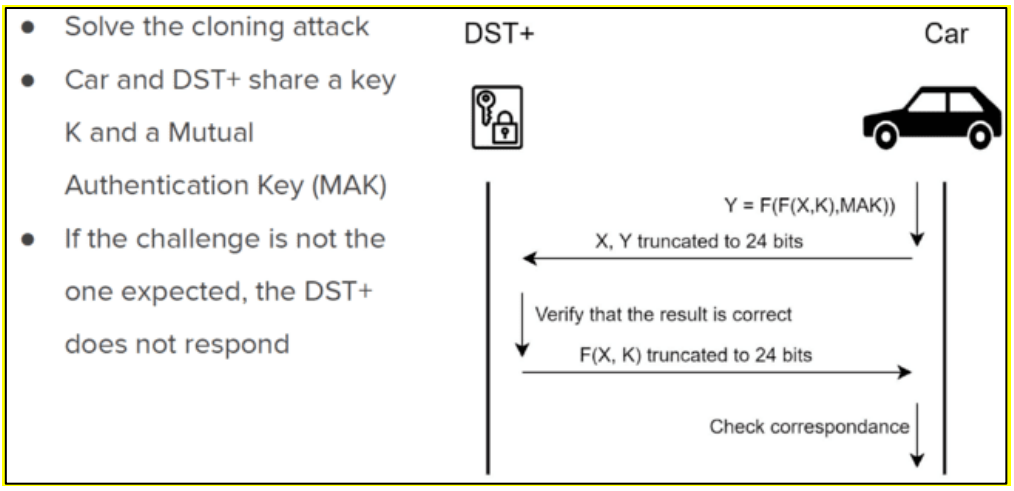
Immobilizers of cars such as Ford, Toyota, and Nissan were based on **Digital Signature Transponders** (**DSTs**), stronger mechanisms. The DST is a tiny RFID chip that, among the others, is enabled with a cypher F() and a 40-bit secret key. The DST and the car both share a copy of the private key K, used to decrypt the communications.



Initially the car sends a 40-bit challenge to the key. The DST encrypts the challenge sent by the car thanks to K and sent it back to the car. At that point the car uses K to decrypt the information received by DST (the challenge) and if this information matches, the engine starts.

PROBLEM: the challenge generated by the car is not temporary, so we cannot know if the challenge has been generated now or in the past. An attacker can capture this value and replay it since the challenge is always the same.

Cloning attack: there is another huge problem with the key length of 40 bits. An attacker might send a challenge and record a response from a car and try all the 1.1 trillion possible key combinations to infer the private keys. Huge number but requires few hours on a FPGA. This led to **DST+** mechanism where both DST+ and the car expects the right challenge as response to start the engine.

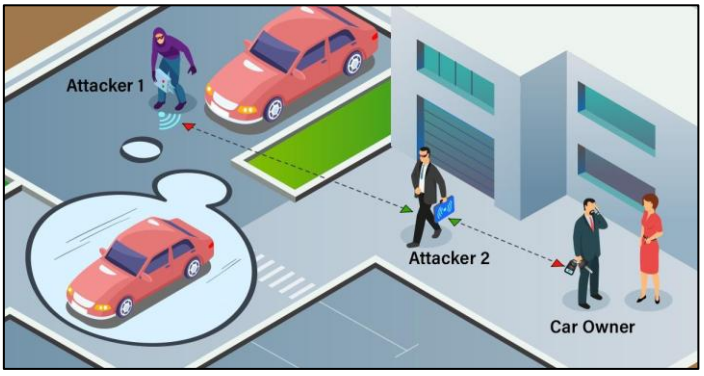


3.1 Relay attack

An evolution of keys is **Passive Keyless Entry System** (**PKES**). It does not require interaction with the user, there is not any button to be pressed ("hand free experience"). The car not only checks for the presence of a legitimate code but **checks also where the physical key is**. It uses a low-frequency RFID channel to check if the key is in remote distance (up to 100 m), outside the car (1-2 m from the door handle), or inside the car. Only in the last case the engine starts.

A different threat model envisions having no direct access to the car's key. Still, keys use wireless communications to talk with the car and execute a challenge response algorithm. So, how do you steal a car in a minute exploiting this technology and why would this work?

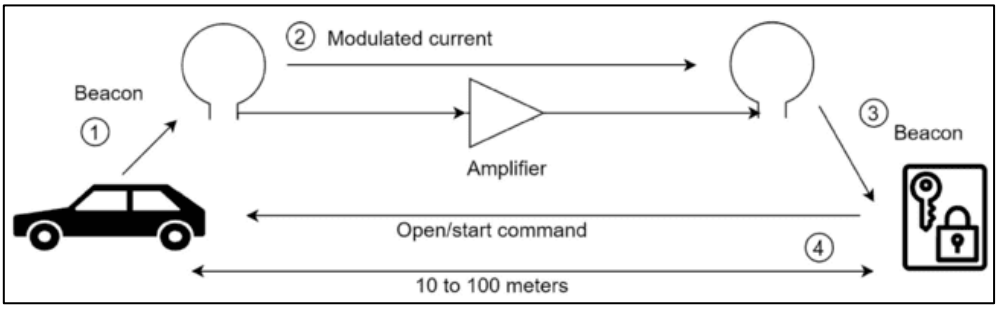
A **relay attack** works by extending the range of communication between two legitimate devices (e.g., a car and its key fob) that would normally be too far apart to communicate.



When relaying signals, we need to care only about the physical layer. We do not need to interpret the signal, modify it, infer keys, … We just need to capture, amplify and retransmit the signal. The signal sent by the car and the response from the key fob is already encrypted and valid, the attacker simply relays the message.

Note: **relay attack adds some delay to communication** (signal is captured, amplified and retransmitted), so we must be sure that the introduced delay is within an accepted range by the application under attack. In the case of the car, the key fob sends an encrypted signal to unlock the car. Normally, this process happens within milliseconds when the key is nearby. In a relay attack, the attackers must ensure that their devices relay the signal fast enough so that the car still "believes" the key fob is nearby. If the delay is too long (beyond what the car's system allows), the car might not respond, and the attack would fail.

Implementation over cable



Two loop antennas connected via a cable that replays the low frequency signal between them. One antenna communicates with the car and the other one with the key. **The car and the key think they are closed so the generated command is accepted**. Basic idea: the two antennas capture the signals generated by the car and the key.

Implementation over the air

Cables may bring suspicions so, let's just remove them. The relay system is now composed by two parts, an emitter and a receiver:

- The emitter captures the low frequency signal and upconverts it to 2.5 GHz, amplifies it, and transmits it over the air.
- The receiver downconverts the signal back to low frequency, it amplifies it again and sends it to the loop antenna.

Note: in this implementation we need to take into account wireless communication constraints and the potential loss of information.

Replay vs relay attack

In replay attacks an attacker captures the signal and transmit it later on to infer the system (not real-time) while in relay attack the attacker captures the signal and transmit it immediately to infer the system (real-time).

How to estimate the real distance between the keys and the car

Distance bounding is a technique used to estimate the distance between two devices (such as a car and its key fob) by measuring how quickly signals are exchanged between them. This ensures that only devices within a certain proximity can communicate successfully, so it is possible to **mitigate relay attacks by measuring how long it takes for the signals to travel between the car and the key**.

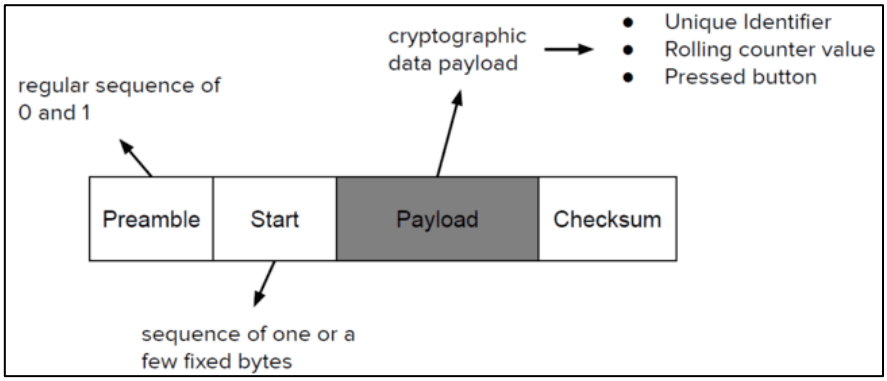
The car uses the round-trip time to compute an upper bound on the distance between itself and the key fob. For example, if the round-trip time is very short, the car knows the key must be nearby (within a certain range, such as a few meters). If the round-trip time is too long, the car concludes the key is too far away and won't unlock or start.

3.2 Rolling Codes

Remote Keyless Entry (RKE) relies on unidirectional data transmission from the remote control (in the car key) to the vehicle. The key is hence active, and upon pressing a button transmits signals in one of the bands 315 MHz, 433 MHz, or 868 MHz (depending on the country). RKE systems enable the user to comfortably lock and unlock the vehicle from a distance and can be used to switch on and off the anti-theft alarm when present.

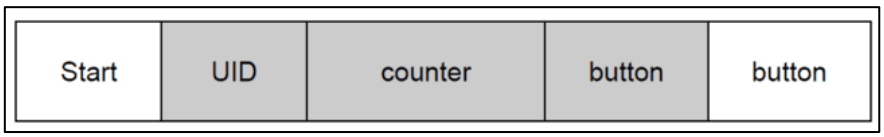
The first generation of RKE used no cryptography, solely relied on a fix-code signal. However, this makes replay attacks super easy. To prevent replay attacks newer RKE use **rolling codes**: every time the button is pressed, the system generates a new code using cryptography combined with a counter value that increments with each press. The car verifies the signal by checking if the counter and other information match expected values. This ensures that even if someone records the signal, they can't use it later, because the code will have changed.

RKE frames



- **Preamble**: is fixed and it's used by the car to detect the presence of messages.
- **Start**: used to start a conversation/information transfer.
- **Payload**, formed by unique identifier (id of the key), rolling counter value, pressed button. There's also the time information about when the key generated the packet.
- **Checksum**.

Payload structure



The grey part represents the encrypted data. The payload is encrypted using a proprietary block cipher. We assume that such a cypher is the AUT64 (as found in many cars from the Volkswagen group).

Authentication in rolling codes may be:

- **Implicit**: the entire payload is symmetrically encrypted. The car receiver checks the UID and if the counter is in its validity window, i.e. if there is a correspondence between the car and key's counter.
- **Explicit**: the sender computes some sort of message authentication code over the data payload and appends it to the packet. The car than checks this message to ensure integrity and authenticity of the message itself.

Note: the key element in rolling codes is **synchronization between the car and the keys**.

Both the car and remote are synchronized to an initial number. It usually relies on some configuration, such as pressing a button or a master key. Once the remote and the car are synchronized, they use an algorithm to choose an initial random number x (to be less affected by possible attacks) and transform it to the next number in the sequence (x+1), not necessarily the following number. If we start the counter x from 0 and we increase the next counter value +1 is easy for the attacker (replay attack) to predict the next number
The following key press will take the result of the previous keypress as input. If a single remote press is intercepted there is no way to determine the next code expected by the receiver (due to encryption). Note: the key's counter always increases but the car's counter increases only when it receives a valid message.

Devices not only store the next code that needs to be transmitted, but a significant number of them (e.g., 255). This is fundamental to have the process working even if in case of erroneous keypresses that increase the counter only on the key side → we can recover the right value.
As soon as a button keypress that is valid is received, the list is updated to be x iterations from that keypress = [1..255] → [2..256]. However, if the number of keypresses is too large, there is no way to catch up on sync and the car might no longer recognize the key fob's codes.

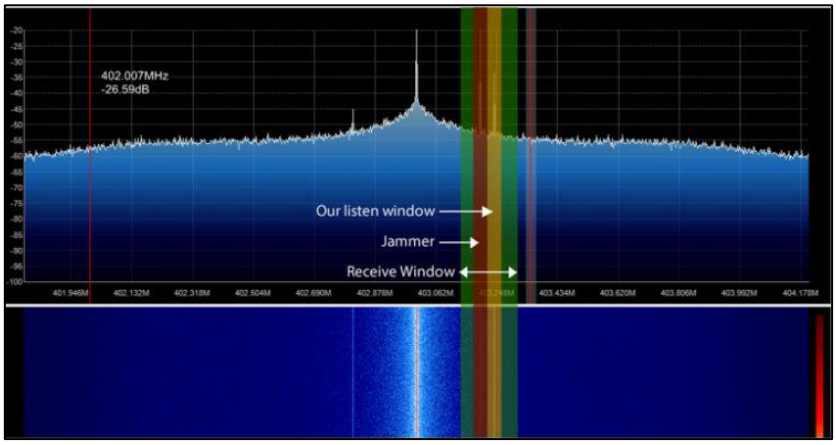
There are few well-known manufacturers providing rolling code based RKE. Microchip Technology provides Classic, Advances, and Ultimate KeeLoq with publicly available documentation and data sheets. Companies like NXP, Omron, and TI provide proprietary solutions.

3.3 RollJam/RollBack attack

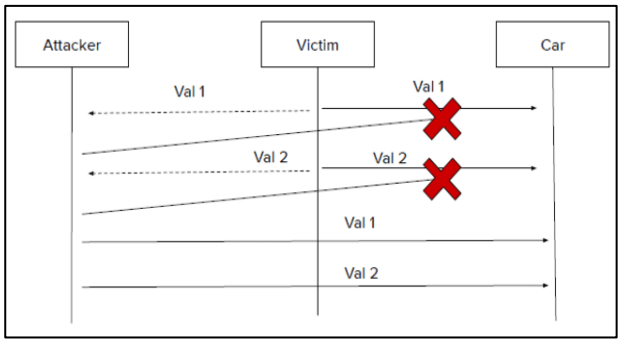
How can we attack systems based on rolling codes? (Missing Link Attack)

As an attacker, we would like the receiver not to be able to receive any of the radio transmissions from the remote. To achieve this, we can **jam (block) the communication between the key and the car**. This is a technique used by thieves to have drivers not to close their cars in the parking lot. When the owner of the car presses the key button to close his car there's a jamming that blocks the signal and the car doesn't receive the signal from the key, so the owner goes away thinking he closes the car, and the thief steals the car.

Remote jamming works, but the person walking away from the car may notice that it did not lock (no noise emitted by the car). We want to develop a technique that is indistinguishable from the standard interaction between car and owner. We integrate jamming and replay attack in the RollJam attack. We want to jam the signal, capture it, and somehow replay it as a legitimate communication between key and car.



While the signal is jammed, we can use the same window to capture the first legitimate message that the key sent to the car. This is possible because the jamming is not in the exact same frequency of the signal. A thief can store the signal and use it later on → the car doesn't receive anything (its counter does not increase), so the captured signal (current+1) is still valid.



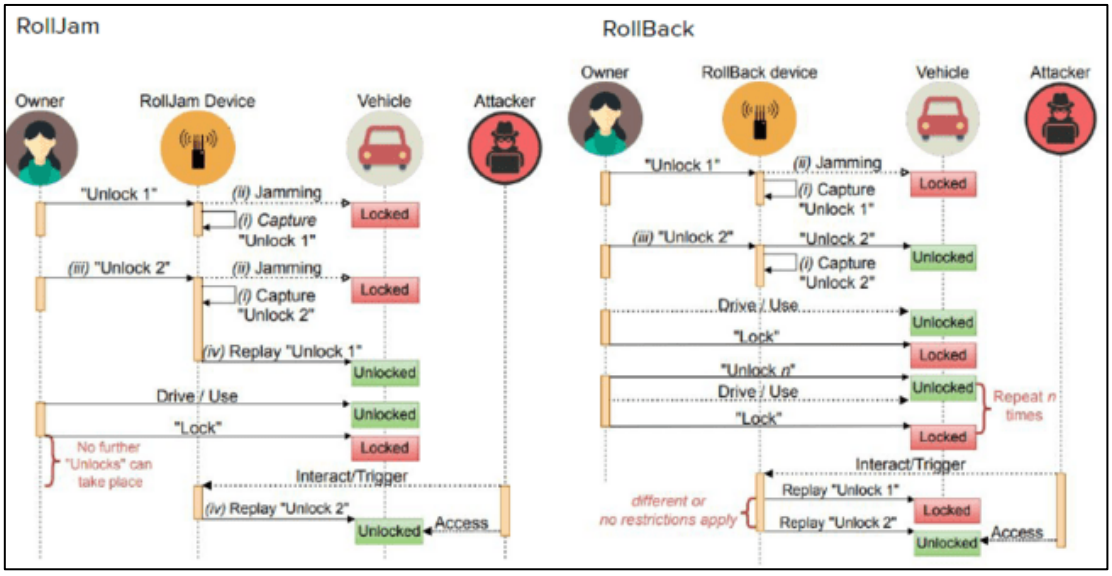
When someone cannot get a remote to work, the instinctive reaction is to press the remote again. As soon as this happens, we can get a second (successive) instance of a legitimate transmission.

We can replay the first signal to lock the car so the owner goes away and then we can use the second captured signal to unlock the car.

RollJam problem: this attack works only in key fob with lock/unlock functionalities in the same button. Anyway, many modern cars use different frequencies (different buttons) for lock and unlock.

- If the attacker only monitors the frequency of the unlock button, then they will only be able to unlock the car.
- It requires the legitimate owner to send a legitimate lock, before being able to perform the unlock, given that lock and unlock use the same rolling code.
- Solution: permanently jam the lock frequency such that the user will need to manually lock the car.

Again, RollJam requires the jammer to be always there. Although rolling codes have been designed to prevent replay attacks, there are still chances that replay might work. We said that there may be some synchronization issues between key and car for which resynchronization is needed → let's exploit this mechanism to evade the security provided by rolling codes.



1. The attacker places the RollBack-device near the target car.
2. When the victims comes back to their car, they will try to unlock it via the key fob sending $Send_V(S^i_{unlock})$
3. RollBack-device:
   - Capture the signal
   - Jam the frequency band to prevent reception
4. The victim assumes lousy reception and presses the same button again, sending $Send_V(S^{i+1}_{unlock})$
5. The RollBack-device captures the signal, but let the car receive it → the victim unlocks and drives away (no doubts).

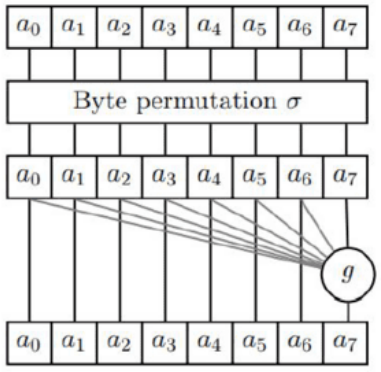
RollBack exploitation

By design, once the attacker captures two rolling codes, those codes become invalid for normal use, as the key fob and the car move forward in their rolling code sequence. The victim can

continue to lock and unlock their car without noticing any issues, as the car and key fob generate new codes with each button press. However, if the key fob is pressed out of the car's range (for example, inside the victim's home or far from the vehicle), the car will miss some of those signals, causing it to go out of sync with the key fob. To resynchronize, the car has a built-in mechanism that accepts two consecutive valid codes to realign itself with the key fob. The attacker exploits this resync mechanism by replaying the two previously captured codes in sequence. Since the car expects two consecutive codes for resynchronization, it accepts these old codes, allowing the attacker to unlock the car by tricking it into thinking the attacker's replayed codes are coming from the legitimate key fob.

3.4 AUT64 and Hitag2 Cipher

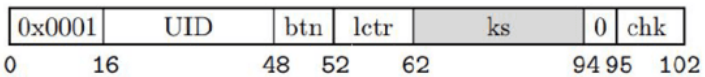
- AUT64 is an iterated cipher that operates on 8-byte blocks
- In each round, the state is first permuted
- Then, byte 7 is updated using the round function g(a_0, ..., key_i)
- Key_i = 32-bit round key
- Total of 12 rounds



Doing the math, the effective key size of AUT64 is 91.55 bit. Finding the key via exhaustive search is not practical. However, the problem is that this RKE system uses a global master key which is independent from the vehicle or remote control. We can use the master key to generate new keys.
This means that **the same key is stored in millions of ECUs and RKE remotes, without any key diversification**. The sole means by which the vehicle determines if a rolling code is valid is hence by whitelisting certain UIDs and checking if the counter is within the validity window.

RKE based on Hitag2 Cipher

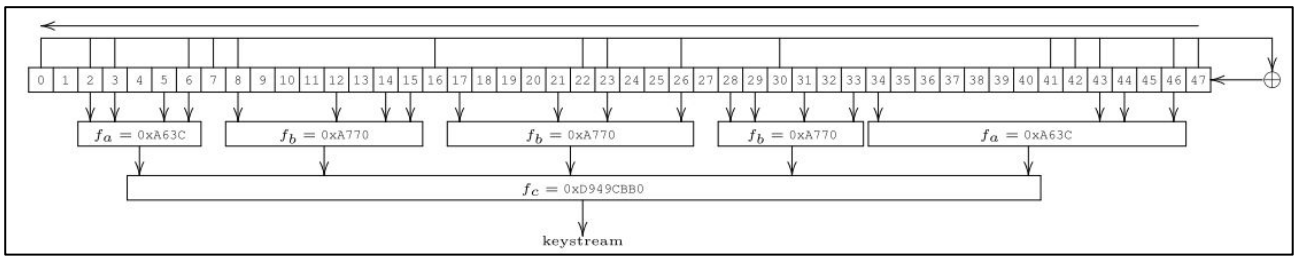
- Hitag2 consists of a 48-bit LFSR and a non-linear filter function f
- In the rolling code scheme, when a button is pressed it transmits the following message



- The initial state of the stream cipher consists of the 32-bit UID concatenated with the first 16 bits of the key k
- ctr is incremented and then initialization vector (iv) = ctr || btn is XORed with the last 32 bits of the key and shifted into the LFSR

The cipher consists of a 48-bit Linear Feedback Shift Register and a non-linear filter function f. Each clock cycle, twenty bits of the LFSR are input to f to generate a bit of the keystream. The LFSR is then shifted one bit to the left, using the feedback polynomial to generate a new bit on the right.

Definition 4.1 *The feedback function* $L\colon \mathbb{F}_2^{48} \to \mathbb{F}_2$ *is defined by* $L(x_0 \dots x_{47}) := x_0 \oplus x_2 \oplus x_3 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_{16} \oplus x_{22} \oplus x_{23} \oplus x_{26} \oplus x_{30} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{46} \oplus x_{47}$.



Rolling Code on Hitag2

- During the authentication protocol, the internal state of the stream cipher is initialized.
- The internal state consists of the 32-bits UID concatenated with the first 16 bits of the key k.
- The counter ctr is incremented and iv = ctr || btn is XORed with the last 32 bits of the key and shifted into the LFSR.
- For this point, the next 32 bits of the keystream (output by the cipher ks) are sent as a proof of knowledge of the secret k.

RKE Based on Hitag2 Cipher

- The next 32 bits of keystream, which are output by the cipher ks, are sent as proof of knowledge of the secret key k.
- L is the feedback function from the LFSR (Linear Feedback Shift Register) that generates a bit.

Definition 4.4 *Given a key* $k = k_0 \dots k_{47} \in \mathbb{F}_2^{48}$, *an identifier* $id = id_0 \dots id_{31} \in \mathbb{F}_2^{32}$, *a counter* $ctr = ctr_0 \dots ctr_{27} \in \mathbb{F}_2^{28}$, *a button identifier* $btn_0 \dots btn_3 \in \mathbb{F}_2^4$ *and keystream* $ks = ks_0 \dots ks_{31} \in \mathbb{F}_2^{32}$, *we let the initialization vector* $iv \in \mathbb{F}_2^{32}$ *be defined as*

$$iv = ctr||btn.$$

Furthermore, the internal state of the cipher at time i *is* $\alpha_i := a_i \dots a_{47+i} \in \mathbb{F}_2^{48}$. *Here the* $a_i \in \mathbb{F}_2$ *are given by*

$$a_i := id_i \qquad \forall i \in [0,31] \quad (1)$$

$$a_{32+i} := k_i \qquad \forall i \in [0,15] \quad (2)$$

$$a_{48+i} := k_{16+i} \oplus iv_i \oplus f(a_i \dots a_{i+47}) \qquad \forall i \in [0,31] \quad (3)$$

$$a_{80+i} := L(a_{32+i} \dots a_{79+i}) \qquad \forall i \in \mathbb{N}\,. \quad (4)$$

Furthermore, we define the keystream bit $ks_i \in \mathbb{F}_2$ *by*

$$ks_i := f(a_{32+i} \dots a_{79+i}) \qquad \forall i \in [0,31]. \quad (5)$$

Note that the a_i, α_i, *and* ks_i *are formally functions of* k, id, *and* iv. *Instead of making this explicit by writing, e.g.,* $a_i(k,id,iv)$, *we just write* a_i *where* k, id, *and* iv *are clear from the context.*

CORRELATION ATTACK ON HITAG2

The purpose of the attacker is to retrieve the key. It requires a minimum of four rolling codes, but it would be faster and more precise by having more traces.

Rolling codes may have an arbitrary counter value, i.e., non consecutive. However, this is good as it increases correlation. We denote as <UID, iv^j, ks^j>, j = 0,...,n-1, n>3, n authentication traces.

- The adversary first guesses a 16-bit window corresponding to LFSR stream bits a_{32},..,. a_{47} = k_0,...,k_{15}.
- Together with UID, this gives the adversary a_0, ..., a_{47}, which is constant over traces.
- The adversary can hence compute $b_0 = f(a_0, \dots, a_{47})$.
- The adversary shifts this 16-bit window to the left of the LFSR, until bits a_{32}, .. , a_{47} are on the very left of the LFSR, i.e., the point where the cipher starts outputting ks.
- The adversary computes a correlation score for this guess
- The window determines 8 input bits x_0, ..., x_7 to the filter function f_{20}, while the remaining 12 inputs remain unknown.
- The correlation is taken as the ratio of those 2^{12} input values x_8, ..., x_{19} that produce the correct keystream bit ks_0.
- Shifting the window further to the left, the adversary can perform tests on multiple keystream bits (ks_0,..., ks_{15}).
- The adversary assigns this guess the average score over all traces.
- So far this scoring computation is independent of the value iv as it happens before iv gets to have any influence on it.
- The adversary sorts guesses based on their score and stores them in a table, discarding guesses with lowest score if needed.
- Experiments show that 400, 000 guesses are usually sufficient.
- For each guess in the table, the adversary goes back to Step (1) and proceeds as before, except that she will now extend the window size by one guessing the next LFSR stream bit (a_{48}, ... , a_{51}).
- The power of this attack comes from using the window on the right of the LFSR to compute the necessary keystream bits to correct the internal state.

- On average, the attack recovers the cryptographic key in approximately 1 minute of computation
- It requires between 4 and 8 rolling codes

Manufacturer	Model	Year
Alfa Romeo	Giulietta	2010
Chevrolet	Cruze Hatchback	2012
Citroen	Nemo	2009
Dacia	Logan II	2012
Fiat	Punto	2016
Ford	Ka	2009, 2016
Lancia	Delta	2009
Mitsubishi	Colt	2004
Nissan	Micra	2006
Opel	Vectra	2008
Opel	Combo	2016
Peugeot	207	2010
Peugeot	Boxer	2016
Renault	Clio	2011
Renault	Master	2011

4. Autonomous Vehicles

An autonomous vehicle (AV) is a vehicle capable of **sensing the surrounding environment and take actions without human intervention**. These kinds of vehicles need a lot of sensors, include thermographic cameras, GPS, inertial measurement units, etc. The sensors' output is used as input for a controller which creates a model to identify the most appropriate navigation path.

Note: when we talk about an autonomous vehicle, we are not referring just to cars but also to drones, robots (in some factory), etc.

Autonomous vehicles use different types of sensors to perceive the environment (for instance to detect obstacles) and monitor their own physical parameters:

- Cameras
- LiDAR
- RADAR
- SONAR
- IMU
- GNSS

Sensor fusion techniques are used to reduce measurement uncertainties due to noise and they constitute a specific class of algorithms that controls the output from sensors to increase reliability.

PLANNING PHASE (the main goal of an AV)

Using the data obtained from the perception system, the vehicle performs behaviour planning. The optimal behaviour needs to be decided by predicting states of the ego vehicle as well as other objects in the surrounding. Based on the planned behaviour, the AV generates an optimal trajectory. The entire process is called **motion planning**.

SAE LEVELS OF AUTOMATION

The Society of Automotive Engineers (SAE) defined 6 levels of driving automation:

- **Level 0**: no automation, manual control.
- **Level 1**: driver assistance with monitoring functionalities (e.g., cruise control).
- **Level 2**: partial automation with the car taking autonomous actions such as steering and acceleration, but the human can always take back the control.
- L**evel 3**: conditional automation, where the vehicle can perform most of the driving task, but human override is still required.
- **Level 4**: high automation where the vehicle performs all driving tasks under specific circumstances and in geofenced areas. Human override is still an option.
- **Level 5**: full automation with the vehicle performing all driving tasks in all conditions. No human attention or interaction is required.

There are several limitations and barriers that could impede adoption of AVs, including: the need for sufficient consumer demand, assurance of data security, protection against cyberattacks, social distrust, and the development of economically viable AV technologies.

4.1 Control systems

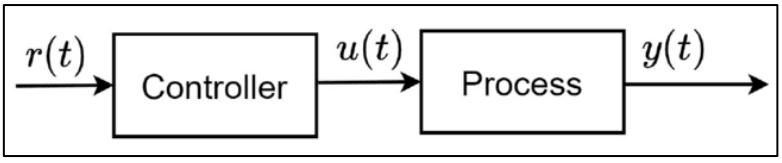
Cruise Control (**CC**) is a system that automatically controls the speed of a motor vehicle. It is a system that automatically **uses error-sensing negative feedback to correct** the throttle of the car and maintain a constant speed. CC is a simple implementation of a control system that, in control theory, is called Proportional-Integral-Derivative (PID) controller.

Control Theory is a field of engineering and applied math that deals with the control of dynamical systems. The objective is to develop a **controller**, a device that **acts on the system to achieve a desired state**. The controller monitors the process via a *process variable* and compares it with a *reference* or *set point* (in the case of CC, the speed). The difference between these values is called the *error signal*.

Example (possible scenario): if we want to maintain a certain distance from the car above, we need to communicate the distance reference value to the controller which computes the error signal and performs some operation to minimize the error to reach the given distance value.

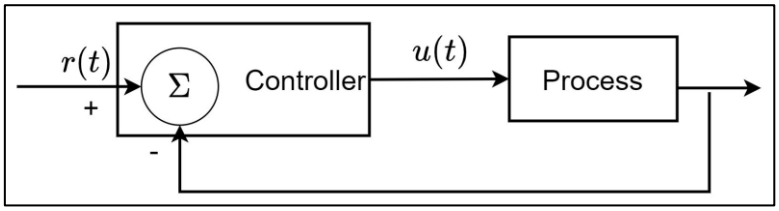
4.1.1 Types of controllers

Open Loop (Feedforward) controller



A reference signal is provided to the controller. The controller implements a function that computes an output that instruct the system on what it has to do to reach the target. At this point the process computes an output which is the current system value.

Closed Loop (Feedback) controller



It operates by comparing the system's output (process value) with the reference input (desired value). Unlike an open-loop system, the feedback controller continuously monitors the process output and adjusts the system's behavior to minimize the difference (error) between the actual and desired values. Examples of feedback control systems include a car's cruise control, which maintains a set speed, and a thermostat, which regulates temperature.

A PID control system for an autonomous vehicle is broken down in two fundamental components:

- **Longitudinal control**: control of the longitudinal motion of the vehicle, with variables being the throttle and brake inputs.
- **Lateral control**: control the lateral dynamics of the vehicles, with variables being the steering inputs to govern the steering angle and heading (two different things).

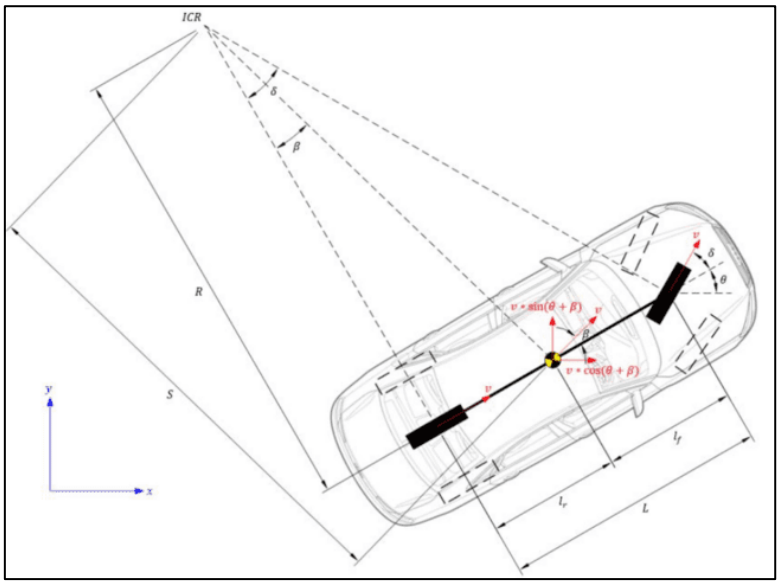
KINEMATIC MODEL (BICYCLE MODEL)

Kinematics is the study of motion of a system disregarding the forces (wind, etc) that govern it. It can be employed in situations wherein kinematics relations are able to sufficiently approximate the actual system dynamics. This model is **valid only for systems that do not perform aggressive manoeuvres at lower speeds**, driving slowly and making smooth turns. Even if quite restrictive, we start to consider those kinds of vehicle to make things simpler.

A widely adopted model to capture vehicle dynamics under normal driving conditions. It strikes a good balance between simplicity and accuracy. The idea is to define a vehicle state and see how it evolves over time based on the previous state and current control inputs, defining so its behaviour.

Let the vehicle state q comprises the x,y coordinates, heading angle θ, and velocity v:

$$q = [x, y, \theta, v]^T$$



- For controls, we need to consider both longitudinal and lateral steering commands
- Brake and throttle contribute to longitudinal acceleration in range [-a'$_{max}$, a$_{max}$]
- Steering command alters the steering angle of the vehicle $\delta \in [-\delta_{max}, \delta_{max}]$
- The control vector is hence $u = [a, \delta]^T$

- Using the distance between the rear wheel axle and the vehicle's center of gravity, we can compute the slip angle beta as $tan(\beta) = \frac{l_r}{S} = \frac{l_r}{\left(\frac{L}{tan(\delta)}\right)} = \frac{l_r}{L} * tan(\delta)$

$$\therefore \beta = tan^{-1}\left(\frac{l_r}{L} * tan(\delta)\right)$$

- Ideally $l_r = L/2 \Rightarrow \beta = tan^{-1}\left(\frac{tan(\delta)}{2}\right)$

- Resolving the velocity vector v into x and y we get

$$\dot{x} = v * cos(\theta + \beta)$$

$$\dot{y} = v * sin(\theta + \beta)$$

- In order to get theta, we first need to calculate S as

$$S = \frac{L}{tan(\delta)} \quad \rightarrow \quad R = \frac{S}{cos(\beta)} = \frac{L}{(tan(\delta) * cos(\beta))}$$

$$\dot{\theta} = \frac{v}{R} = \frac{v * tan(\delta) * cos(\beta)}{L}$$

- Finally, we can compute $\dot{v} = a$
- We hence get the continuous-time kinematic model

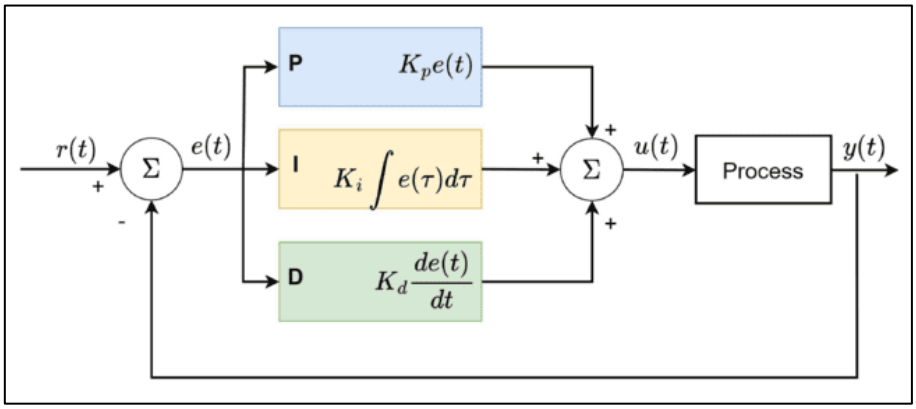
$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v * cos(\theta + \beta) \\ v * sin(\theta + \beta) \\ \frac{v * tan(\delta) * cos(\beta)}{L} \\ a \end{bmatrix}$$

- And hence the discrete-time model

State transition equation

$$\begin{cases} x_{t+1} = x_t + \dot{x}_t * \Delta t \\ y_{t+1} = y_t + \dot{y}_t * \Delta t \\ \theta_{t+1} = \theta_t + \dot{\theta}_t * \Delta t \\ v_{t+1} = v_t + \dot{v}_t * \Delta t \end{cases}$$

PID controller



As previously discussed, PID controller is a control loop mechanism using feedback to apply continuously modulated control. It continuously computes an error as the difference between a desired setpoint and a measured process variable. Based on the error value, PID controller **applies a correction based on proportional** (P), **integral** (I), and **derivative** (D) terms. Based on the type of error computed, the controller applies a correction.

- P is proportional to the current error.
- I is the integral of the error in a predefined past time window.
- D is the derivative of the error and is the best estimate of the future trend.
- K_p, K_i and K_d determine the magnitude of the correction.

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau)\,\mathrm{d}\tau + K_\mathrm{d} \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

- The integral of the speed is the distance
  - Difference with the distance it would have traveled at the target speed
  - Deals with hills

$$d(T) = \int_T v(t)dt$$

- The derivative of the speed is acceleration
  - Helps in responding quickly to changes

$$a(t) = \frac{v(t)}{dt}$$

4.1.2 Adaptive Cruise Control (ACC)

Adaptive Cruise Control (ACC) is a system that automatically controls the speed of a motor vehicle by integrating sensing capabilities on the vehicle (SAE Level 1). The vehicle uses a radar/LiDar/laser to compute the distance from a car in the front. This distance is reported to the controller, which acts on the speed to maintain a minimum safety distance.

EXAMPLE OF ACC

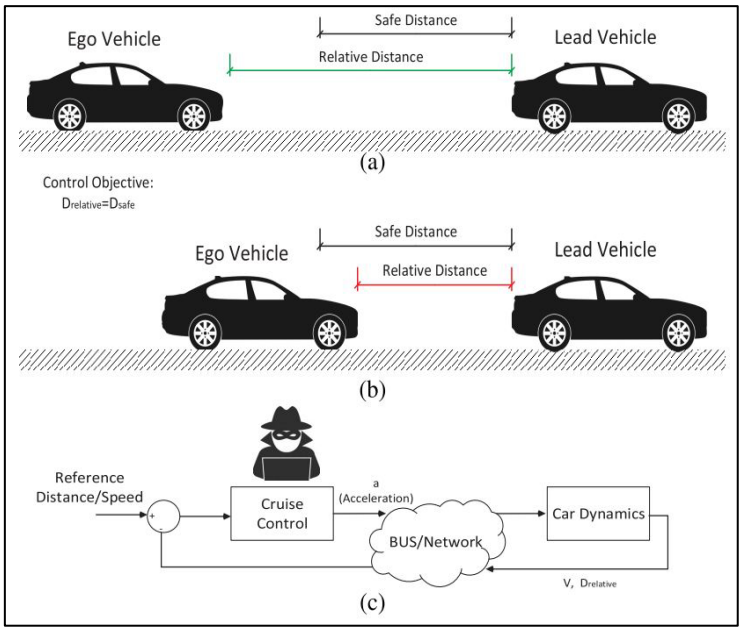
Assume we have two cars, one in the front (lead) and one in the back. We assume that the one in the back (ego vehicle) has ACC. Ego uses a radar to measure the distance to the lead vehicle.

The lead car is supposed to be driving in the same lane. Ego also uses the radar to measure the relative speed between the two cars.

Adaptive Cruise Control may be enabled by two operating modes:

- **First mode**: the ego vehicle should drive at a driver's specified speed as long as it maintains a safe distance with the lead vehicle.
- **Second mode**: the space between the two vehicles is controlled (by changing the ego vehicle's speed) so that the two vehicles do not get closer than a safe distance.

According to real-time measurements, either working modes can be enabled by the ACC system.



To determine the operating mode, the ACC applies the following rules:

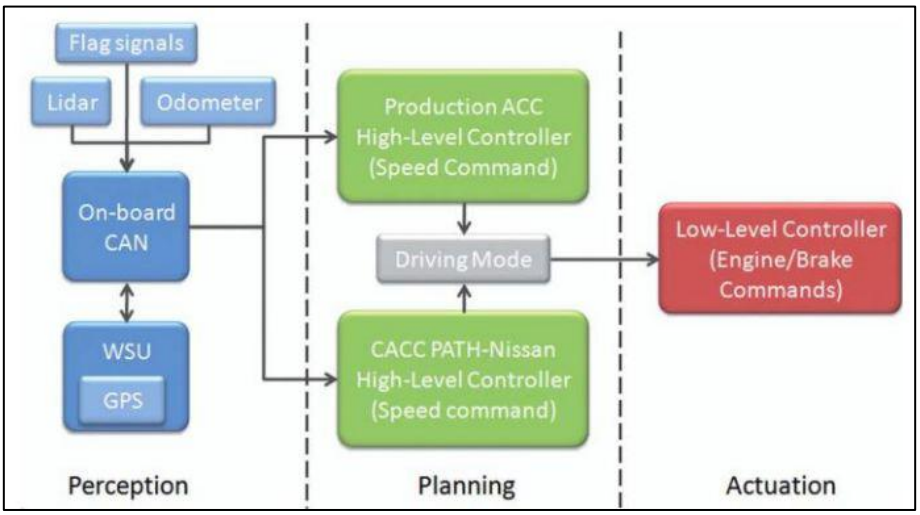
- If relative distance is >= safe distance, then use speed control (first) mode to track the driver set velocity.
- If relative distance is < than safe distance, then the space control (second) mode is active and keep track of the distance.

We explore two different attack scenarios in which we assume the attacker can compromise in some way the ACC unit:

1. **The attacker remains dormant** and monitors the measured distance to the front vehicle. At the times this distance is at its lowest (presumably near the minimum safe distance), it creates a spike (a very high signal) in the control signal and makes the vehicle accelerate. This could similarly happen when the front vehicle suddenly brakes and the compromised ACC refuses to reduce the speed.
2. Unlike the first scenario **the attacker does not hide for the attack**. Trivially lowers the ACC's reference distance. During the times ACC is in second mode and tries to maintain the safe distance, it is practically following a false reference. Since this difference is trivial and not noticeable to the driver, this attack remains covert or stealth.

4.1.3 Cooperative Adaptive Cruise Control (CACC)

Cooperative Adaptive Cruise Control (CACC) is a system that automatically controls the speed of a motor vehicle by integrating sensing capabilities on the vehicle and **communication capabilities with other vehicles**. It uses Vehicle-to-Everything (V2X) communication. In addition to what ACC does, CACC uses the preceding vehicle's acceleration in a feed-forward loop. This info passes via the Cooperative Awareness Messages, allowing to communicate its own status to the other vehicles.



Perception Phase:

- Get information from on-board sensors and include them in CAN data.
- The Wireless Safety Unit (WSU) provides:
  - Data transmitted by other CAVs (Connected and Autonomous Vehicles) in the CACC system through V2V communications.
  - Data collected by GPS with wider area augmentation system differential corrections (e.g., vehicle position assigned in the CACC system).
- Information derived from on-board sensors such as Lidar, odometer and flag signals will also be received and included on the CAN bus data structure.

Planning Phase:

- Includes the high-level controller (longitudinal control algorithms).
- The controller is usually connected and instructs the vehicle via the CAN bus.
- Very similar to ACC.

Actuation Phase:

- Execute target reference command transmitted from the planning phase.
- Low-level controller converts the target speed commands into throttle and brake actions.

The vehicle dynamics is calculated by the vehicle controller and controller acts only on its vehicle. In particular, we often use longitudinal control, we write the sum of the forces acting on the vehicle and use them to maintain the same longitudinal speed of the other vehicles in a CACC system while keeping a fixed longitudinal inter-vehicle distance. CACC technology allows CAVs to form vehicle platoons with shorter inter-vehicle distances.

Model of CACC

- Let us consider a platoon of K cars, numbered from 0 to K-1
- We assume that the cars all drive in a single straight lane and that their order can not change
- We denote the spatial position, velocity, acceleration of car i as q_i, v_i, a_i
- We indicate the distance between the front bumper of car i and the rear bumper of car i-1 as $d_i = q_{i-1} - q_i$
- We indicate the desired distance between car i and car i-1 as $d_{r,i}$
- Cars desire to follow a constant headway policy
- $d_{r,i} = h_{d,i} v_i + L_i$ where the last term is a constant distance offset and h is the desired headway of car i
- Given the error at car i $e_i = d_i - d_{r,i}$
- $e_i = q_{i-1} - q_i - h_{d,i} v_i - L_i$
- We can define the control strategy based on the desired acceleration value $u_i = u_{fb,i} + u_{ff,i}$
- Sum of feedback input and communication-received input
- We measure inter-vehicle distance with radar and use PD feedback for in-vehicle info $u_{fb,i} = k_p e_i + k_d \dot{e}_i$
- We account for a values received via DSRC stating the control strategy of vehicle i-1 $\dot{u}_{ff,i} = -h_{d,i}^{-1} u_{ff,i} + h_{d,i}^{-1} \hat{u}_{i-1}$
- This controller has been shown to work in real life
- J. Ploeg, B. T. Scheepers, E. van Nunen, N. van de Wouw, and H. Nijmeijer. Design and experimental evaluation of cooperative adaptive cruise control. In Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on, pages 260–265. IEEE, 2011

ATTACK STRATEGIES

An attacker can cause a series of abnormal behaviours in CACC system, in particular referring to the platooning case. The attacker may either be an external actor or an inner vehicle/byzantine node.

1) **Reduced headway attack**: a car intentionally ignores the recommended $h_{d,a} < h_{d,min}$ headway speed that guarantees string stability and follows closer. By ignoring the safe headway (the ideal following distance for stability), the driver can reduce wind resistance and save fuel. However, following too closely can destabilize the entire line of cars (or "platoon") and increase the risk of collisions, especially if sudden braking occurs.
2) **Joining without radar**: this attack involves a vehicle trying to join the platoon without the proper radar or distance-sensing equipment. Instead of using radar to gauge safe following distances, the vehicle relies only on information it receives through communication systems, such as DSRC (Dedicated Short-Range Communications). This setup is risky because, if there is any communication delay or disruption, the car cannot adjust its position based on actual distance data, which could lead to accidents. The attacker control strategy is hence $u_a = u_{ff,a}$.
3) **Mis-report Attack**: the attacker misinforms the vehicle that is following to increase the following car's headway or to cause a change in the following car's behaviour. The attacker mounting this attack could either follow the prescribed control law or choose an alternative control law. Assuming the attacker misreporting only its behaviour, then $u_a = u_i$. This is motivated by wanting to increase the following distance of the preceding car.
4) **Collision Induction Attack**: the attacker broadcasts an acceleration profile indicating that they are speeding up which causes the following vehicle to accelerate. The attacker starts to aggressively brake which causes the error between the attacker and following car to quickly increase. Very similar to attacks that could be mounted in the current highway system. If a driver was to jam on their breaks during rush hour while being tailgated, the vehicle would likely be rear ended.

Attacks summary

Attack	Impact	Motivation	Method
Reduced Headway Attack	Decreased String Stability	Decreased fuel consumption Increased density	Misbehavior
Joining Without Radar	Decreased String Stability Danger in wireless congestion	Decreased cost over radar equipped car	Misbehavior
Mis-report Attack	Decreased Performance	Mistrust of the system	Misinformation
Collision Induction Attack	Collision Loss of Life Property Damage	Maliciousness Terror	Misbehavior & Misinformation
Non-Attack Abnormalities	Decreased Performance Decreased String Stability	Improper Maintenance	Misbehavior

How do we detect and defend against these attacks?

We can describe our CACC system with a double integrator model with a lag constant nu for each car:

$$\begin{aligned}\dot{a}_i &= -\eta_i^{-1} a_i + \eta_i^{-1} u_i \\ \dot{v}_i &= a_i \\ \dot{q} &= v_i \\ \dot{e}_i &= v_{i-1} - v_i - h_{d,i} a_i.\end{aligned}$$

- Let us define a vector with the state of the car $x_i^T = [e_i, v_i, a_i, u_{ff,i}]$
- The state update equation of the car can be written as a linear system

  $\dot{x_i} = A_{i,i}x_i + A_{i,i-1}x_{i-1} + B_{s,i}u_i + B_{c,i}\hat{u}_{i-1},\ \forall\, i > 0$

  $\dot{x_0} = A_0 x_0 + B_{s,i}u_r$

  where

$$A_{i,i} = \begin{pmatrix} 0 & -1 & -h_{d,i} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\eta_i^{-1} & 0 \\ 0 & 0 & 0 & -h_{d,i}^{-1} \end{pmatrix}, \qquad A_{i,i-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

$$B_{s,i}^T = \begin{pmatrix} 0 & 0 & \eta_i^{-1} & 0 \end{pmatrix},$$

$$B_{c,i}^T = \begin{pmatrix} 0 & 0 & 0 & h_{d,i}^{-1} \end{pmatrix},$$

$$A_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\eta_0^{-1} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

- We alsod define a variable X representing the state of the whole system $X^T = [x_0^T, x_1^T, \ldots x_{K-1}^T]$
- We define the inputs to the system as $U^T = [u_0, \hat{u}_0, u_1, \hat{u}_1, \ldots, u_{K-1}]$
- This allows us to write the equation for the whole system

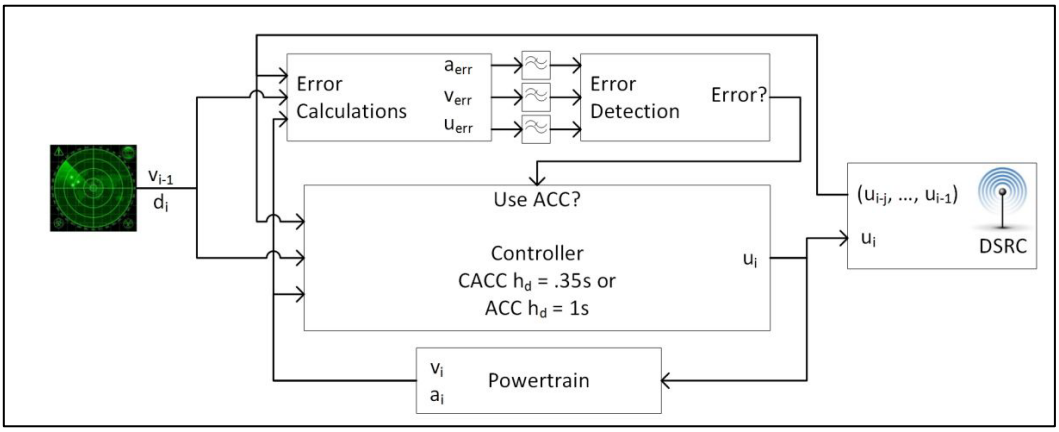
$$\dot{X} = AX + BU$$

- We assume that the controller impèlements digital control instead than analog
- The controller sampling time will depend also on the sampling time of the radar and of the communication system
- Thus, we can rewrite the update equations as $X[k+1] = A_d X[k] + B_d U[k]$
- We also rewrite the control strategy as $u_i = k_1 x_i[k] + k_2 x_i[k-1]$
- Assuming that radar update is 1ms and DSRC update is 100 ms

$$k_1 = \begin{pmatrix} k_p + \frac{k_d}{.001} & 0 & 0 & 1 \end{pmatrix} \qquad k_2 = \begin{pmatrix} -\frac{k_d}{.001} & 0 & 0 & 0 \end{pmatrix}$$

MODEL BASED DETECTION

Every car models the expected behaviour of the vehicle directly in front of them. Vehicles then compare the calculated expected behaviour with the observed behaviour. The car is then able to detect both malicious and benign abnormalities. Once abnormal behaviour is detected, the car switches from operating in a cooperative platoon framework to a radar only based adaptive cruise control framework where it is safe even if the preceding car is mounting an attack.



- Car i wants to model the behavior of car i-1 given the data packets from car i-j
- We define the modeled state of car i-1 as $x_{m,i-1}$
- We can define the state of all cars in the model as

$$X_m = [x_{m,i-j}, x_{m,i-j+1}, \ldots, x_{m,i-1}]$$

- We can write the system update equation for the model as

$$X_m[k+1] = A_m X_m[k] + B_m U_m[k]$$

- We assume all cars behave according to the control law in red
- During an update period we can hence use it to define

$$U_m[k] = \phi_1 X_m[k] + \phi_2 X_m[k-1] + \phi_3 \hat{u}_{i-j}[k]$$

$$\phi_1 = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 \\ 0 & k_1 & 0 & \ldots & 0 \\ 0 & k_1 & 0 & \ldots & 0 \\ 0 & 0 & k_1 & \ldots & 0 \\ 0 & 0 & k_1 & \ldots & 0 \\ \vdots & & & \ddots & 0 \\ 0 & 0 & 0 & \ldots & k_1 \\ 0 & 0 & 0 & \ldots & k_1 \end{pmatrix}, \phi_2 = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 \\ 0 & k_2 & 0 & \ldots & 0 \\ 0 & k_2 & 0 & \ldots & 0 \\ 0 & 0 & k_2 & \ldots & 0 \\ 0 & 0 & k_2 & \ldots & 0 \\ \vdots & & & \ddots & 0 \\ 0 & 0 & 0 & \ldots & k_2 \\ 0 & 0 & 0 & \ldots & k_2 \end{pmatrix} \quad \phi_3 = (1, 1, 0, \ldots, 0)^T$$

- During a non-update period we can hence use it to define

$$U_m[k] = \phi_4 X_m[k] + \phi_5 X_m[k-1] + \phi_6 U_m[k-1]$$

$$\phi_4 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & k_1 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & k_1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & k_1 \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}, \phi_5 = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & k_2 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & k_2 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & k_2 \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}, \quad \phi_6 = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

The modeling technique is based on double integrator and can be made complicate at wish. Considering trade-offs in accuracy, calculation cost, and time for calculation. Improvements that could be considered in the modeling include capturing non-linear behavior of the vehicles drivetrain and using terrain mapping to predict variation.

Thresholding

Once we have the prediction model, **we can predict whether the error is acceptable or not**. We indicate the measured values as $\hat{x}_{i-1,m}$, also we assume we can compute acceleration and velocity. However, we cannot measure the error since we do not have a line of sight with cars further than a single hop.

We use the model error normalized to acceleration

$$\begin{aligned} (a_{err}|\hat{u}_{i-j}) &= \left(\frac{(a_{m,i-1}|\hat{u}_{i-j}) - a_{i-1,}}{a_{i-1}} \right)^2 \\ (v_{err}|\hat{u}_{i-j}) &= \left(\frac{(v_{m,i-1}|\hat{u}_{i-j}) - v_{i-1}}{a_{i-1}} \right)^2 \\ (\hat{u}_{err}|\hat{u}_{i-j}) &= \left(\frac{(\hat{u}_{m,i-1}|\hat{u}_{i-j}) - \hat{u}_{i-1}}{a_{i-1}} \right)^2 \end{aligned}$$

A different methodology

Model predictive control can be implemented in multiple ways. In the previous example, each vehicle implements the model of the preceding vehicle. However, we can model more complex systems. For instance, it could be possible for vehicle i to simulate the entire platoon up to vehicle i-1.

4.1.4 Replay attacks in CACC

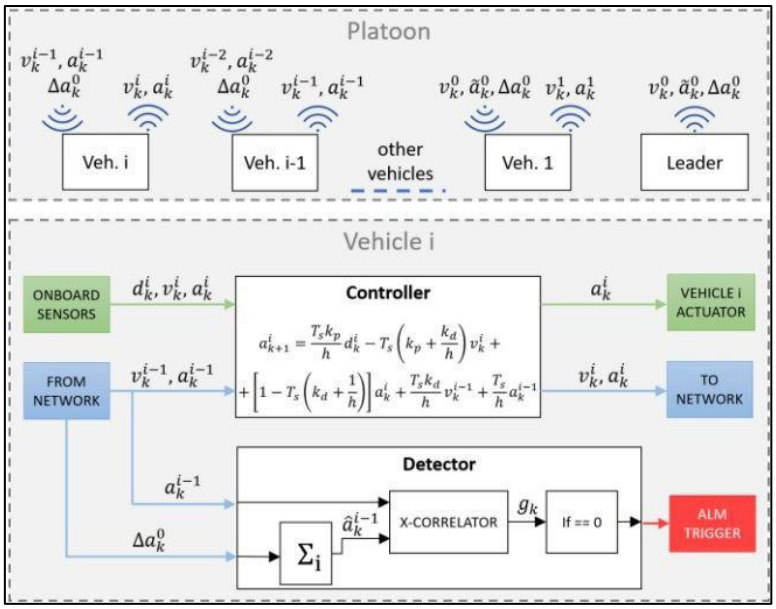
The platoon of vehicles is ideally going to reach a steady state condition where all the relative distances and the velocities are constant and where the accelerations are equal to zero. We suppose the attacker can:

- Record measurements during a steady state situation without influencing in any way the platoon behaviour.
- Replays the recorded data while acting to either degrade the platoon performance or damage the vehicles involved in the platoon.

We assume the attacker has already got full access to the cryptography keys and to the in-vehicle network and it is acting as an intelligent insider. The controller of a legitimate vehicle starts relying on malicious information. We however assume that the leader is immune to hacking and is hence secure.

Detection of replay attacks

- The leader vehicle broadcasts its velocity, acceleration, and a **noisy signal** Δa_k^0
- The following vehicles broadcast their speeds and their accelerations and read from the network the speed, the acceleration of the vehicle in front of them and the noisy signal generated by the leader.
- The noisy signal acts as a **timestamped authentication** signal which is propagated along the vehicles in the platoon formation through the control laws exploited in each vehicle. Δa_k^0 is used to update the control law in the vehicle 1, which ensures the platoon moves as a coordinated unit. Also note that the noise is dynamic, it changes over time, making difficult for an attacker to reply it in a second moment.
- The noisy behaviour is propagated by each vehicle to the following one by updating the control action and by exchanging speed and acceleration signals through the network with the next vehicle.
- The detection system combines a **control algorithm** and a **detection algorithm** in each vehicle, with a specific focus on monitoring the acceleration and authenticity of signals transmitted through the platoon.
- The propagation of noise allows each vehicle to "check" that the signal it is receiving corresponds to the latest broadcast from the leader, making it much harder for an attacker to inject previously recorded (replayed) signals without detection.
- The i-th detector runs a **virtual model** Σ_i that simulates the platoon up to vehicle i (in its steady state) with given input the authentication signal Δa_k^0. The purpose of the model is to **estimate the acceleration signal a_k^{i-1} in absence of attack starting from the noisy signal**. Since the noisy signal is continuously changing, the model's output will match the actual signal only if the data has not been tampered with.
- We use a **cross-correlator** to measure the correlation between signal a_k^{i-1} coming from the front vehicle and the estimate $\hat{a}_k^{i-1}$ output of the virtual model.
- If there is an attack, the output of the correlator should be zero.



Noisy control

- The leader vehicle generates the noisy control signal as:

$$\tilde{a}_k^0 = a_k^0 + \Delta a_k^0$$

Actual control signal — Original control signal — $\Delta a_k^0 \sim N(0, \vartheta)$

- The cross correlation is computed over a **time window** with a given length.
    - A shorter time window means the system checks for mismatches more frequently, leading to faster detection of attacks. However, it also **increases the chance of false detections**. This is because natural fluctuations or noise in the signals might temporarily look like an attack when observed over a very short period, leading the system to mistakenly trigger the alarm.
    - A longer time window allows the cross-correlation algorithm to observe the signals over a broader period, reducing the impact of temporary fluctuations or minor noise. This setup **reduces false detections** since the algorithm has more data to distinguish genuine attacks from random noise. However, it also **slows down detection**. If an attack occurs, the system will take longer to identify it because it needs to accumulate enough data over the longer window to confirm the mismatch.
- A **trade-off between false detection and time of the detection** is needed to choose the right window size with the purpose to properly detect the replay attack.

SENSORS IN AUTONOMOUS DRIVING

Autonomous vehicle act based on i) Sense, ii) Understand, and iii) Act. The **sensing layer** of vehicles is comprised of vehicular sensors that **measure the physical properties** of a vehicle's state and surroundings. By sensing information from different sensors, the car constructs a representation of its environment. We have already seen that distance measurement sensors allow for ACC and CACC.

Sensors can be divided based on their functions in safety, diagnostic, convenience, and environment monitoring as:

- **Safety sensors:** provide night vision, detect impending crashes, and tailor airbag deployment to each passenger's weight and position.
- **Diagnostic sensors:** detect vehicle malfunctions and offer malfunction alerts to drivers.
- **Convenience sensors:** maintain high air quality within the vehicle, control automatic mirror dimming, perform automatic braking and acceleration.
- **Environment monitoring sensors:** track the vehicle's surroundings, including traffic, street signage, and road conditions.

The **sensing layer is vulnerable to malicious interference** conducted both physically and remotely. Tampering requires the attacker to physically access the car to and attach something like electromagnetic actuators or sound-absorbent foam. Remote attack can be either:

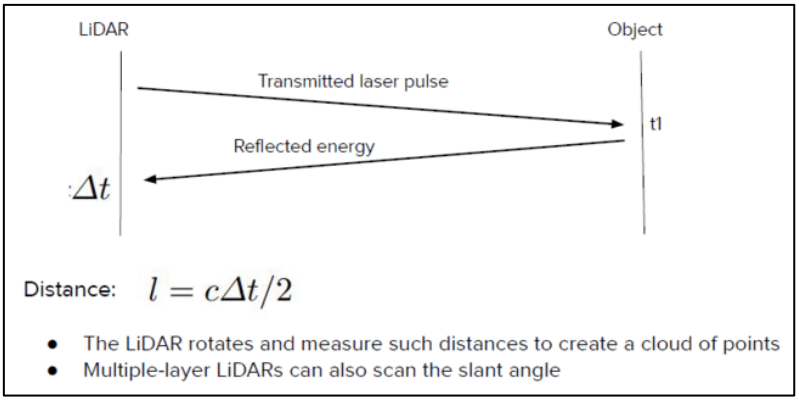
- Roadside: the attacker places stationary attack equipment on one or more locations along the roadside.
- Front/rear/side: the attack equipment is mounted on the attacker's vehicle which follows the victim.

4.1.5 LiDAR

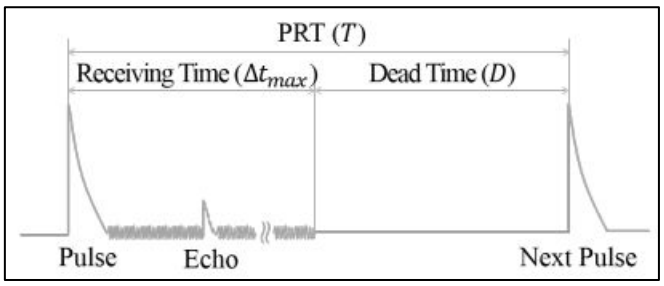
Light Detection and Ranging (LiDAR) is an active remote sensing technique that measures the distance to nearby objects by emitting laser pulses and analysing the reflected signals. This method is "active" because it involves the sensor itself emitting energy, which then interacts with the target and returns data to the sensor. By calculating the time it takes for the light to travel to the object and back, LiDAR precisely maps distances and surfaces.

Two types of LiDAR:

- **Scanning:** mainly composed of laser transceiver(s) and a moving rotary system for scanning, which allows acquiring a round view.
- **Solid state:** do not require moving parts to acquire a round view. Currently however we use scanning LiDARs, as solid state are generally more expensive and the reliability is always the same.



Since LiDAR continuously sends out pulses at regular intervals, there is a **risk of overlap when multiple echoes return**. This can create ambiguities if it becomes unclear which pulse a particular echo corresponds to, potentially leading to distance errors. To limit uncertainties, LiDARs define the **receiving time** and the **dead time**. After sending a pulse, a LiDAR waits for its echoes for the duration of the receiving time. The received echoes are considered as that of the last transmitted pulse. The LiDAR ignores all the echoes received from the end of the receiving time up to receiving time + dead time, and then transmits again. The LiDAR can hence detect objects up to a maximum distance, given by $l_{max} = c\Delta t_{max}/2$



The LiDAR does not require a wide receiving angle if well calibrated. This means it can focus its sensing capabilities within a narrow field, only covering the area where it expects reflections based on the pulse it has transmitted and reducing so noise reflections. Only echoes that fall within this defined receiving angle are considered in the sensing results. The receiving angle only needs to be aligned with the direction of the transmitted pulse for the duration of the "maximum round trip time". After this maximum time has passed, any remaining echoes are irrelevant.

We can hence derive the receiving angle from the rotating speed and the maximum distance as

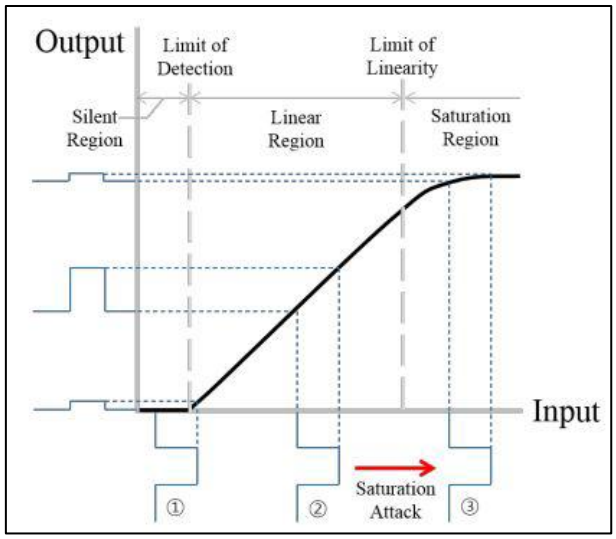
$$\Theta_R = \Delta t_{max} \cdot \omega = \frac{2l_{max}}{c} \cdot \omega \ [^\circ]$$

So, what can an attacker do?

Sensors transition curve comprise three regions:

1. The **silent region**, where no sensing occurs.
2. The **linear region**, where the sensor actively detects objects and provides accurate measurements.
3. The **saturation region**, where the sensor is overwhelmed and unable to measure effectively.

A threshold line marks the transition from silent to linear region (where actual sensing occurs) and another threshold separates the linear region from saturation.



Denial-of-Service (DoS) via saturation

By flooding the sensor with excessive input (e.g., strong signals), an attacker can push it into the saturation region, where it becomes overloaded and stops functioning properly. This disrupts the sensor's ability to gather accurate data, effectively causing a DoS attack.

Spoofing the sensor

An attacker can spoof the sensor by broadcasting a fake signal that mimics the LiDAR's expected signal. This deceptive signal tricks the sensor into interpreting a false environment or situation. **Spoofing exploits a semantic gap**: the difference between the real situation and what the sensor perceives. For example, both an actual earthquake and a child shaking a seismometer might produce similar readings, even though the underlying causes are very different.

Some active sensors use a unique ping waveform to distinguish their own echoes from other signals. This waveform acts like a signature, allowing the sensor to filter out irrelevant signals. However, an attacker could record this ping waveform and replay it with a slight delay to trick the sensor into interpreting the spoofed signal as legitimate data. This is known as **relay spoofing**.

Relay spoofing is particularly hard to detect because of the semantic gap; it can be challenging to distinguish a real reflection from a manipulated signal, as both may appear similar to the sensor.

Blinding Attack

We can **saturate LiDARs using light sources**. In particular, the attacker should point against the LiDAR a light source of the same wavelength as that used by the LiDAR. Depending on the intensity of the light, the attacker can saturate a bunch of dots or an entire direction. This type of saturation attack against LiDAR is called "blinding attack".

Saturation attacks against LiDARs have some common characteristics:

- **Stealthiness against drivers and pedestrians**: in order to not hinder human driving and for eye safety, LIDARs use infrared (IR) lasers. Invisibility of the medium also assists stealthiness in saturating. Irrespectively of the intensity, human drivers and pedestrians would be unaware, rendering the attack effective.
- **Receiving angle**: a wide receiving angle is not essential for LIDARs to sense objects in the field of view. Therefore, LiDAR receivers typically have much smaller receiving angles compared to the angle of view. This can limit the effect of saturating, because the attacking light comes from a certain direction when the LiDAR rotates. However, in practice the receiving angles of LIDARs are much larger than required.
- **Curved reception glass**: an oblique incidence of strong light onto the curved reception glass can cause the appearance of fake dots in directions other than that of the attacking source.

Effects of Saturation

Experiments on a LiDAR sensor reveal significant effects of saturation on sensor accuracy and functionality. The LiDAR device, which features a real-time visualization tool, was tested using two laser modules as light sources: a weaker 30mW, 905nm laser and a stronger 800mW, 905nm laser. **Under the weaker light source, the LiDAR detected numerous false points**, or "fake dots," concentrated in the direction of the light source. When exposed to the stronger laser at an oblique angle, the fake dots appeared not only in the source's direction but spread across other areas, indicating a broader range of interference. When the strong laser was directed at the sensor head-on, the LiDAR's vision became impaired, rendering it completely blind in a specific sector. This demonstrates how **LiDAR systems can be vulnerable to saturation effects**, particularly from strong, direct laser sources, which could compromise its reliability in detecting genuine environmental features.

Spoofing by relaying: Ideal Spoofing

LiDAR measures distances by calculating the round-trip time of light, where light pulses bounce off the first encountered object and return to the transmitter. An ideal spoofing procedure can exploit this mechanism by simulating this return process, effectively **creating false objects at specific distances from the LiDAR**. This attack setup requires three key components: a receiver, an adjustable delay unit, and a laser transmitter with the same wavelength as the target LiDAR.

The spoofing process follows these steps:

1. Align the attack equipment with the target LiDAR.
2. Capture the LiDAR's emitted pulse using the receiver.
3. Introduce a calculated delay through the delay component, which adjusts the timing.
4. Emit a pulse back to the LiDAR from the transmitter.

The above sequence ideally creates a single, precise "fake dot" that the LiDAR registers as a real object. For the spoofing to succeed, the delay must be precisely computed based on the

desired distance from the LiDAR to create an apparent object at that location. The attacker must **ensure the RTT of the pulse matches the distance that simulates an object further away than the spoofer's own position**. To ensure success, a few conditions must be met:

- The target LiDAR must be oriented toward the attacker, as only then will it capture the spoofed signal.
- The delayed pulse must reach the LiDAR within its acceptable receiving window, as it will ignore any echoes arriving after a set time. This timing constraint requires the spoofed pulse to be returned precisely within the LiDAR's response time.

Spoofing by relaying: Actual Spoofing

While the theoretical spoofing method is sound, the actual implementation introduces some complexities:

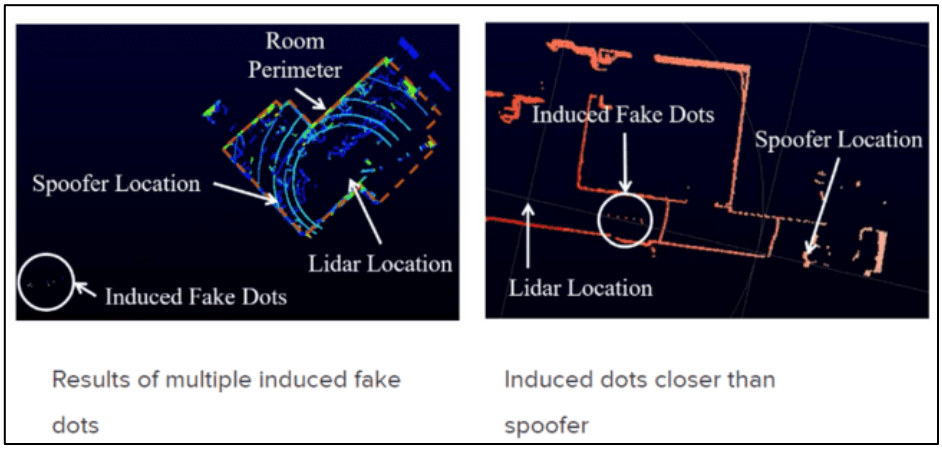
- **Laser divergence**: the LiDAR's laser pulses diverge as they travel, resulting in multiple adjacent pulses rather than a single, focused beam. As a result, only some of these pulses directly reach the attacker's receiver, requiring adjustments to handle this spread in signal detection.
- **Receiver-Transmitter separation**: even if positioned closely, the receiver and transmitter in the spoofing setup are separated by a small distance, introducing a delay S between when the receiver detects a LiDAR pulse and when the transmitter can emit its spoofed pulse. This spatial separation alters the timing of the attack, making it necessary to adjust the original delay calculations.
- **Adjusted Delay calculation**: to maintain accuracy, we must adjust the ideal delay formula to include the time offsets introduced by this setup. Specifically, the delay equation is modified to account for both the physical separation (time difference S) and any processing or propagation delays. These delays are added to the ideal delay, with the delay component triggered by the first detected pulse to initiate the spoofed signal accurately.

By incorporating these compensations for spatial and processing delays, the spoofing setup can more effectively mimic a legitimate signal at a controlled distance from the LiDAR.

Spoofing characteristics

- **Stealthiness**: as for the saturation attack, the source is invisible to humans.
- **Inducing multiple fake dots**: if the lidar rotates at a constant speed, an attacker can generate multiple fake dots with one attack tool. This can be done by periodically firing back the attacking pulses, immediately after the first attacking pulse, with the same period as the Pulse Repetition Time.
- **Receiving angle**: a small receiving angle limits the maximum number of fake dots inducible by a fixed spoofer → use multiple transmitters.
- **Curved Reception Glass**: the oblique incidence of a strong laser pulse to readily induce fake dots in sectors, other than the direction of the attacker.

Spoofing results



Results of multiple induced fake dots

Induced dots closer than spoofer

LiDAR 3D Object Detection

LiDAR is better if estimated 3D objects. This is achieved via 3D object detection models (deep learning), which output 3D bounding boxes. We can group these models into three categories:

1. **Byrd's-eye view (BEV)-based 3D object detection:** project point clouds into the top-down view and use CNN to perform the final decision.
2. **Voxel-based 3D Object detection:** VoxelNet slices the point clouds into voxels and extract learnable features by applying a PointNet to each voxel. A 2D convolution layer is applied in the final stage.
3. **Point-Wise 3D Object detection:** directly operate on point clouds for 3D object detection. Two stage architecture: i) generate high-quality region proposals in the 3D space, ii) regress bounding box parameters and classify detected objects.

Adversarial Attacks

Besides spoofing fake points at the sensor levels, attacks to LiDAR include attacks towards its post-processing pipeline. Let's consider the pristine point cloud X, and the Apollo pipeline (BeV) with hardcoded features x. Given the function phi that preprocesses feature maps and given T', t' as spoofed point coude and corresponding features maps, the attack can be modeled as

model

$$\min \quad \mathcal{L}(x \oplus t'; \mathcal{M})$$

Sensor attack capabilities

$$\text{s.t.} \quad t' \in \{\Phi(T') \mid T' \in \mathcal{A}\} \;\&\; x = \Phi(X)$$

CARLO

In LiDAR-based perception, **the number of points reflected from a vehicle can vary**, leading to situations where fewer points are observed for certain objects. This can happen due to:

- **Occlusion**: when an object (like another vehicle or obstacle) partially blocks the view of a vehicle, only parts of the occluded vehicle are visible to the LiDAR, resulting in fewer reflected points.

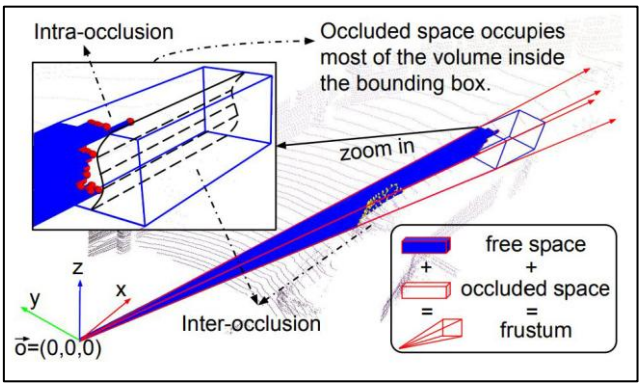
- **Distance**: as objects move farther away, the density of reflected points decreases because LiDAR's laser pulses spread out over a larger area, making distant vehicles appear with fewer points.

These scenarios can make it **challenging to accurately perceive and track objects**, as legitimate vehicles might look sparse in the point cloud, similar to how a fake vehicle might appear if someone is spoofing the LiDAR. **This opens the door for spoofing attacks**, where attackers inject fake points to create the illusion of a non-existent vehicle. Such fake vehicles might also have a sparse point count, mimicking the appearance of an occluded or distant vehicle.

A possible countermeasure is **oCclusion-Aware hieRarchy anomaLy detectiOn** (CARLO). The intuition is that we may base our anomaly detection on the intuition that some attacks violate laws of physics. Harnesses occlusion patterns as invariant physical features to accurately detect spoofed fake vehicles. It consists of two building blocks: i) free space detection, and ii) laser penetration detection.

Free Space Detection (FSD)

- Integrate inter and intra occlusion to detect spoofed values.
- Inter-occlusion: describes a causal relationship between occludee and the corresponding occluders (i.e., the occluders cause the occlude partially visible).
- Intra-occlusion: the facing surface of a solid object (e.g., a vehicle) occludes itself in the point cloud, which indicates that the LiDAR cannot perceive the interior of the object.



- The frustum as well as the straight-line path from the LiDAR sensor to any point in the point cloud is considered as free space.
- The entire 3D space can be divided into *free space* and *occluded space*.
- The former is embedded at the point level, the latter at the object level.
- Free space includes information from occluded space.
- We hence exploit this information to detect fake vehicles.

- Due to inter-occlusion and intra-occlusion, we observe that the ratio f of the volume of FS over the volume of a detected bounding box should be subject to some distribution and upper-bounded by b.
- This means that f is in (0, b].
- Since fake vehicles do not obey the occlusion pattern, their ratio should be large enough and bounded such that f is in [a,1).

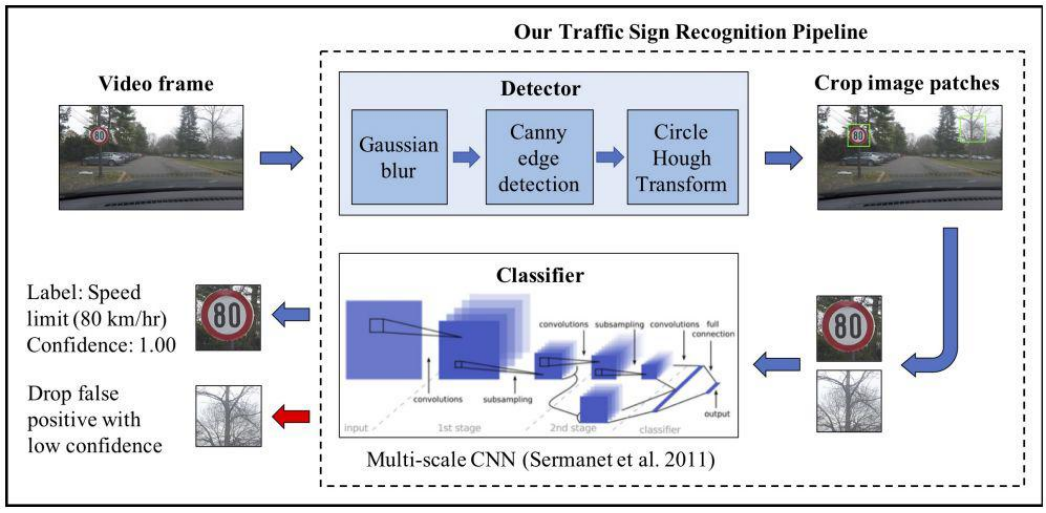
Laser Penetration Detection (LPD)

- Each point in the point cloud represents one laser ray and the boundary between free space and occluded space.
- Given a vehicle's point set, its bounding box B also divides the corresponding frustum into three spaces
    - the space between the LiDAR sensor and the bounding box
    - the space inside the bounding box
    - the space behind the bounding box
- From the perspective of the LiDAR sensor, the ratio g of the number of points located in the space behind the bounding box over the total number of points in the whole frustum should be upper bounded.

CARLO hierarchically integrates FSD and LPD. In the first stage, CARLO accepts the detected bounding boxes and leverages LPD to filter the unquestionably fake and valid vehicles by two thresholds. The remaining bounding boxes are uncertain and will be further fed into FSD for final checking.

LiDAR and radars are generally more expensive than cameras, therefore it is preferable to use cameras to save money. Autonomous cars are generally equipped with multiple cameras spaced around the vehicle. Each camera provides monocular vision and resolve azimuth and elevation angles.

Example: Sign Recognition



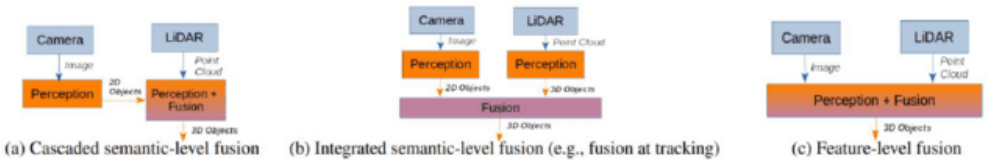
SENSOR FUSION

To improve the performance of the sensing capabilities of cars, we can use **sensor fusion techniques**. Sensor fusion refers to the process of merging information coming from different sensors related to the same scenario to reduce uncertainty. Sensors can be either of the same type or based on different technologies (i.e., camera and LiDAR).

Example

- An example of calculation via sensor fusion is *inverse variance weighting*
- Let us consider two measurements $\mathbf{x}_1$ and $\mathbf{x}_2$ with variance σ_1^2 and σ_2^2 respectively
- Then, the fused measurement is given by $\mathbf{x}_3 = \sigma_3^2(\sigma_1^{-2}\mathbf{x}_1 + \sigma_2^{-2}\mathbf{x}_2)$
- Where the variance of the combined estimate is given by $\sigma_3^2=(\sigma_1^{-2}+\sigma_2^{-2})^{-1}$

Classes of Fusion in Perception



(a) Cascaded semantic-level fusion (b) Integrated semantic-level fusion (e.g., fusion at tracking) (c) Feature-level fusion

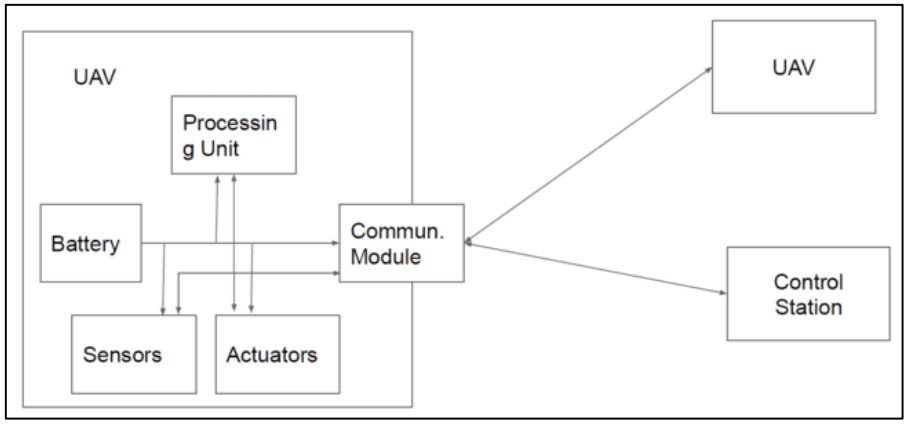
- a) using the output of perception on one or more sensors to augment the input of other single-sensor perception
- b) runs isolated perception for each sensor and fuses semantic outputs
- c) combines low-level (machine-learned) features from multiple perception sources to produce a unified output

5. Drones Security and Privacy

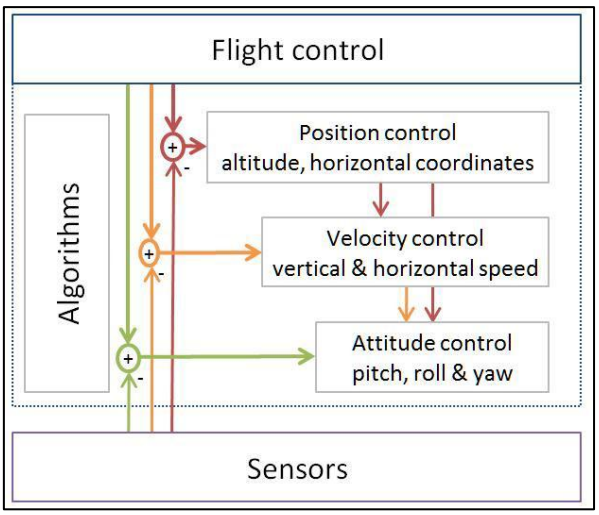
Unmanned Aerial Vehicles (**UAVs**) are flying devices without any crew or pilot on board. They come in different shapes and size and can reach different altitudes (e.g., fixed-wing rotor, quadcopter, tricopter).

UAVs operate primarily through ground-based pilots using **Remotely Piloted Aircraft Systems** (RPAS). A RPAS is formally defined as a collection of configurable components that includes the remotely piloted aircraft, its control station, command and control links, and any additional systems necessary for flight operations. While many UAVs are controlled remotely, some are fully autonomous, capable of coordinating with other drones to accomplish shared tasks. In fleet operations, these UAVs must communicate effectively within their group and with other fleets to ensure safe navigation and prevent collisions, especially in crowded airspace where multiple objectives may be pursued simultaneously.

Structure of a drone



Drones are characterized by multiple modules to acquire and process data. The communication module acts as a central hub that allows the drone to send and receive information from/to other drones or the control station. UAVs employ control loops possibly with sensor feedback.



Applications

- UAVs have been firstly proposed for military applications.
- Seventeen countries armed UAVs, and more than 100 countries use UAVs in military capacity.
- Civil applications are instead more recent.
- Thanks to their high adaptability, UAVs find applications in many different fields:
    - Infrastructure
    - Transport
    - Media and Entertainment
    - Telecommunication
    - Agriculture
    - Search and Rescue

A drone ecosystem is composed by six unique targets:

- Drone hardware: CPU, sensors, firmware.
- Drone chassis and package: all non-electronic devices.
- Ground control station (GCS): may be fixed or mobile.
- First-Person View (FPV) channel: control channel via common communication protocols.
- Pilot: person remotely controlling the drone.
- Cloud services: some drones send telemetry of flight information to a cloud server if needed.

Attacker model

The attacker aim is to **disrupt the legitimate task of flying a drone**. Attackers may be both civilian and military. Although military may have sophisticated equipment (e.g., cannons or predator birds), a civilian may still be able to impact on the drone. We consider a civilian that has access to equipment that can be purchased at a moderate cost (e.g., Software-Defined Radio, Computer, Commercial lasers, Butterfly net, Magnets).

An attacker may be of three types:

1. **With direct physical access**: access the drone or the GCS to modify the firmware or replace hardware parts.
2. **Physically proximate**: send, modify, and replay radio transmissions to hijack a drone.
3. **Distant adversary**: resides on the Internet and applies attacks against servers, drones, the GCS.

The impact of the attack can be measured according to the CIA triad:

- **Confidentiality**: attack that reveals information about drone, pilot, telemetry or collected data.
- **Integrity**: attack that modifies the information delivered to or collected by the drone, GCS, cloud server or pilot.
- **Availability**: attack that causes the pilot to lose control of the drone due to forced landing, crash or hijacking.

Drone hardware attacks

Drones are susceptible to several hardware-based attacks that exploit vulnerabilities in their sensors and navigation systems.

- **Attack to compass**: the introduction of an external magnetic field can mislead the compass reading and either hijack the drone's flight path or prevent it from taking off entirely.
- **Attack on the stabilizing algorithm via camera sensor**: drones often use optical flow analysis which interprets ground features in the camera's field of view to detect movement and adjust stability. An attacker could manipulate this feature by projecting laser beams or images onto the ground, creating artificial patterns that deceive the drone's sensors. This could trick the drone into following the projected features, potentially steering it off course or destabilizing its flight. These vulnerabilities highlight the risks in UAV sensor systems that can be exploited to interfere with or take control of drone operations.

OPTICAL FLOW

Optical flow uses a sequence of successive images to allow for the **estimation of motion**. It tries to calculate the motion between two image frames which are taken at a predefined rate at every voxel position.

UAVs use a **downward facing optical flow camera** that constantly takes pictures of the ground below. By analysing these images, the drone can detect if it has drifted. For instance, if the ground appears to "move" in the images (by a relative offset), it means the drone itself is drifting, since the ground should be stationary.

To make optical flow work, the system needs to identify specific points on the ground that are easy to track, like corners or edges. This is done with a **feature detection algorithm** that marks recognizable spots on the ground in the image. These features are then provided as input to the **optical flow algorithm** which identifies the location of these features in successive images and uses the difference to compute the displacement.
A classic configuration is the one that uses the Shi-Tomasi corner detection algorithm for feature detection and the Lucas-Kanade method to compute the optical flow.

Shi-Tomasi Corner Detection and Lucas-Kanade Optical Flow

The main idea is to use the derivative of the image to detect whether there is a sudden change of color in one direction. If only in direction, then an edge is detected. If in two directions, then a corner is detected. It is generally used because efficient, compared to more sophisticated but slower feature detectors (SURF, SIFT).

Once the features (corners) are detected, the Lucas-Kanade Optical Flow algorithm is used to track them in the next image frame. This method assumes that the change between frames is small and relatively smooth. By observing how the positions of these features shift between frames, the algorithm can estimate the displacement or how much the drone has moved.

For each feature, it looks at a small window around it and calculates the motion (velocity) by analysing the pixel changes. Using a technique called Least Squares, it estimates how far the drone has moved based on these tiny shifts.

This whole process allows the drone to **understand its motion relative to the ground**. If the optical flow system detects movement in the ground features, it can correct the drift to stabilize the flight. Optical flow helps the drone maintain a steady position by continuously analysing and adjusting its location in real-time based on these image comparisons.

ATTACKER REQUIREMENTS

A successful attack to sensors needs three requirements:

1. Environment influence: the adversary can alter the physical phenomenon that the system measures (alter pixel values).
2. Plausible input: create an input to the sensor that will be used by the system as valid input.
3. Meaningful response: the attacker can induce a behaviour on the UAV representing the control the attacker has over it.

1) **Environment influence on Optical Flow**

- The adversary must be able to alter or obscure the ground plane to alter the pixel values reported by the optical flow camera.
- Onerous example: cover part of the area with feature-rich pattern.
- More concrete example: project features on the ground via laser/light beams.
- In the first case, the attacker can use a battery-operated projector loading images (e.g., from a USB stick).
- In the second case, use an array of laser elements to create features.

2) **Plausible input to Optical Flow**

- The basic idea of optical flow is to instruct the UAV to compensate a drift by moving of the same amount in the opposite direction.
- The system assumes the image on the ground to be stationary, so it interprets feature motion along a vector as a movement in the opposite direction.
- By moving the ground feature, the attacker can control the movement of the drone.

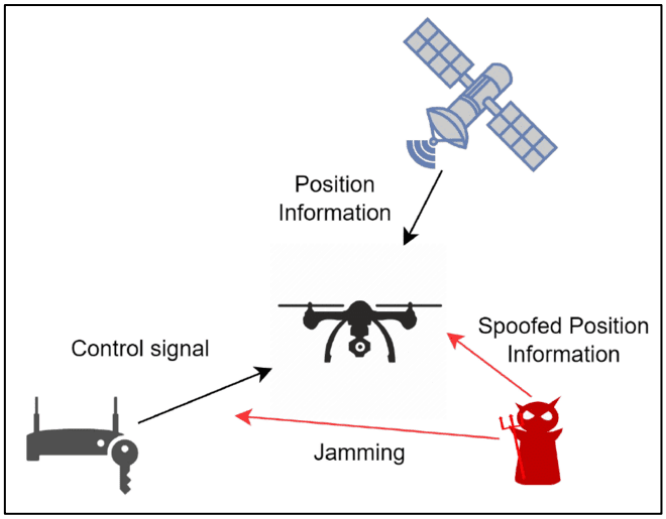
3) **Meaningful response to Optical Flow**

- The attacker's goal is to project a sharp gradient onto the ground so that the feature detection algorithm will pick up the light as a corner.
- Since the Lucas-Kanade-based optical flow computes a final displacement based on the average displacement of each feature, the attacker needs to generate a large number of corners.
- In practice, the attacker can simply sweep their projected light across the ground plane.

CAPTURING DRONES

Most UAVs are enabled with the **Return to Home** (RTH) functionality. The UAV records its location of departure, and in case of emergency it will automatically fly back to home. Emergency includes low battery or loss of the control signal for more than 3 seconds.

Most UAVs rely on the Global Navigation Satellite System (GNSS) for positioning (e.g., Europe's Galileo). However, this system has the main issue of not being authenticated. An attacker can hence spoof the GNSS signal and control the location of the drone. However, the drone is controlled by its pilot.



When jamming the signal for enough time, the drone enters the RTH mode and starts to fly back to the home location. With the help of the GPS module and the on-board compass, it can derive a trajectory based on its current location S and the home location H. The attacker aims at **sending spoofed location information to capture the drone** in the minimum time by bringing it to a controlled location D. To achieve the shortest capturing time, the attacker may want the UAV to fly along the straight line SD. However, the drone has a pre-recorded H and will try to return there. If only one could spoof an arbitrary location to the UAV, then the UAV will change its direction immediately to any desired angle.

For instance, consider a location S' such that S'H || SD (i.e., S'H is parallel to SD trajectory). The idea is to make the drone think it is at a new position S' such that when it tries to head toward H from S', it actually ends up moving in the general direction of D. However, GPS accepts only physics-reasonable coordinates. So, the attacker must carefully choose spoofed coordinates that are within a plausible range to trick the drone effectively.

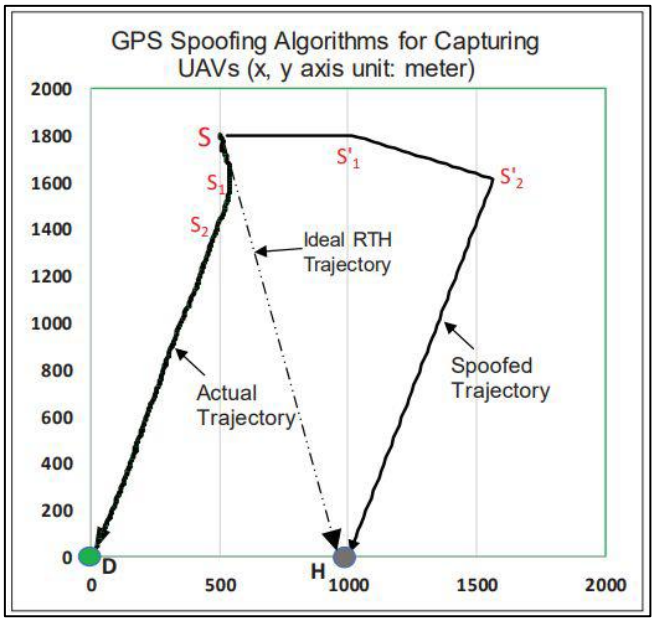
Greedy Algorithm for UAV Capture

Constraint: the **maximum speed** at which the attacker can change the drone's GPS location through spoofing is limited to V_s.

Idea: to minimize the capture time, the UAV trajectory shall be spoofed such that it points towards D as soon as possible.

- Loiter mode: the UAV tries to hold its position S by correcting any drift. We exploit this feature to have the drone gradually moving to D.
- An ideal spoofing trajectory is the one starting from the horizontal line a, where the arriving point is exactly above H.
- Besides spoofing, the UAV will always point towards the direction of H thanks to its compass.
- Whenever it believes it is on the left side of the first spoofed point S1', it moves closer to H horizontally.
- After passing S1, the UAV direction will turn left until it points precisely at D.
- Denoting S2' as the turning point in the spoofed trajectory and its corresponding point in the actual trajectory as S2, S2'H || S2D.
- The UAV, convinced to be taking the spoofed trajectory S2'H, will fly along the straight line S2D until it reaches D.

- How to determine the spoofed trajectory S1'S2' such that its actual trajectory becomes S1S2 and S2'H becomes parallel with S2D at the earliest time?
- Greedy algorithm to compute S2':
    - At the spoofed location S1', the next spoofed point Sx' is such that S1Sx' is perpendicular to S1D.
    - If SxD || Sx'H, done.
    - Otherwise, repeat.



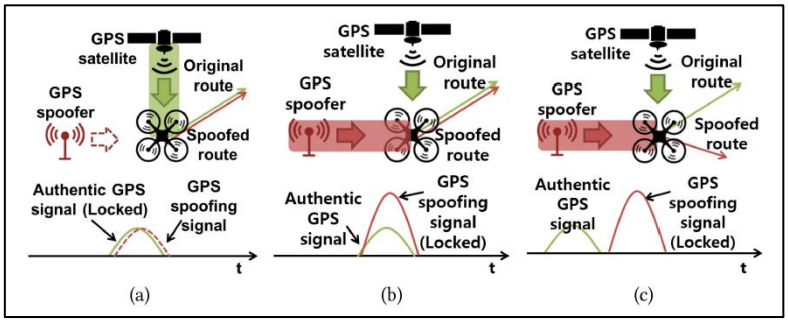
5.1 Safe Hijacking

GPS systems may occasionally drop the signal or suffer from glitches, i.e., provide significantly inaccurate position information. The **GPS failsafe** is a mechanism used to provide safety to the drone in case of GPS signal loss or glitches. In this case, the drone may either land or switch to a manual control. To protect against glitches, the drone has a short memory of GPS position and compares the new measurement with the old one (it is possible to use Kalman Filter algorithm also in this case).

Traditional anti-drone systems often rely on jamming the drone's radio control signal to disable remote controllers. However, this approach can be ineffective against drones operating in autonomous mode, guided solely by GPS. Consequently, GPS spoofing emerges as a potential strategy to **safely divert drones from unauthorized zones**.

Soft GPS Spoofing

The spoofing signal is aligned with the authentic GPS signal. The idea is not to jam the signal but try to align the drone with the spoofed signal and then increase its power to lock the drone to that signal without interruption. This happens in three steps:



Hard GPS Spoofing

Soft spoofing has some requirements to satisfy to avoid losing the lock, one of these requires emitting a stronger signal in a way the drone chooses the spoofed one instead of the authentic one (when it loses the GPS signal). When one of the requirements is not satisfied, we call this signal as hard GPS spoofing. It initially acts like a jamming signal and the victim may lose its lock on the authentic signal. The spoofing signal is stronger, thus the victim will reconnect to the spoofed signal.

Drone types according to their behaviour after GPS recovering

Drone type	GPS fail-safe flight mode	Behavior after GPS recovery	Corresponding safe-hijacking strategy	Belonging consumer drones
I	Positioning mode (non-GPS)	Positioning mode (GPS)	Strategy A	DJI Phantom 3 & Phantom 4
II		Autopilot (GPS)	Strategy B	Parrot Bebop 2
III		Continue fail-safe	Strategy C	3DR Solo
IV	Landing			—*

* We were not able to find any consumer drones that correspond to Type IV. We added Type IV for the sake of completeness without any missing consumer drone.

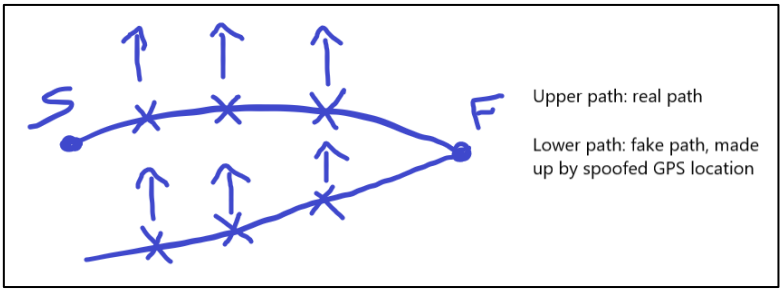
A GPS spoofer can be integrated into the drone detection system and the spoofer can adaptively generate hard and soft GPS spoofing signals according to the hijacking strategy.

Safe hijacking strategy for type 1 drones

The key to this strategy is that the type 1 drones are trying to stay over their original position. If the attacker spoofs the target drone's GPS position as if the drone is moving in a certain direction, then drones are considered to drift owing to external factors such as wind. Thus, the target drone generates speed in the opposite direction, so the drone moves in that direction in the real world. With this method, GPS spoofing is possible in all directions - 360°.

Safe hijacking strategy for type 2 drones

If the GPS position is manipulated as the drone deviates from the path, then it will move in a different direction from the original one to return to the track. The moving direction is determined by their path following algorithm and the fake position. Then, based on the analysis of the path-following algorithm, the hijacker can determine the hijacking direction and calculate the corresponding fake location.



Note: the above figure shows the main strategy used to hijack type 2 drones. The path SF represents the real (authentic) drone path. During the path we provide different spoofed GPS locations (represented by the "x" in the fake path) in a way the drone believes it is on a different path/location, represented by the line/locations below the original SF. The drone tries to return to its original path by moving up and returning to SF. Anyway, tthis results in the drone moving away from SF. The drone is hence hijacked to a fake location.

Safe hijacking strategy for type 3 drones

Since the type 3 drones in failsafe mode need to wait for a pilot command, manipulation of GPS signal is not effective. In this strategy, the focus is to prevent the target drone from losing GPS lock despite GPS signal manipulations (soft GPS spoofing). However, it might be complicated to move the drone if it uses a combination of GPS and IMU sensors (accelerometer and gyroscope) to determine its location.

Note that Strategy C can also be applied to Type 1-2 drones. However, it is the decision of the hijacker whether to apply this strategy to those types of drones, because soft GPS spoofing is more difficult and expensive than hard GPS spoofing.

THE 3DR SOLO CASE

3DR Solo does not have a fallback mechanism when losing connection with the GPS (i.e., a set of predefined actions, for instance RTH). It is complicated to hijack, because it uses both GPS and IMU for localization. It has a complicated path-following algorithm that uses **Intermediate Target Position (ITP)**, necessitating moving the spoofed GPS location accordingly to safely control the behaviour of the target drone. 3DR Solo represents an interesting case study for strategy C.

Note: Kalman filter is a mathematical tool that helps to estimate the true state of a system (like a drone's orientation) by combining noisy sensor measurements with a mathematical model of how the system should behave.

Path following algorithm

- The drone uses an Extended Kalman Filter algorithm to estimate various parameters, such as velocity, position, and magnetic field by fusing the IMU's output with GPS measurements.
- The EKF algorithm starts by predicting its position and velocity by using the IMU's output only.
- IMUs have inherent errors → periodically compare its predictions with the GPS values and correct based on errors.

ALGORITHM 1: EKF failure detection algorithm

```
// ekf_check() is called at 10  Hz by ArduCopter scheduler
Function ekf_check()
    if motors are stopped then
        bad_variance ←False
        fail_count ← 0
        return
    get innovVelSumSq from the EKF
    get varVelSum from the EKF
    velVar ←sqrt(innovVelSumSq / varVelSum)
    if velVar >= 0.8 then
        if bad_variance is not True then
            fail_count ←fail_count +1
            if fail_count >= 10 then
                fail_count ← 10
                bad_variance ←True
                change its mode to the EKF fail-safe mode
    else
        if fail_count > 0 then
            fail_count ←fail_count −1
            if bad_variance is True and fail_count == 0 then
                bad_variance ←False
```

- During flight the drone might deviate from the track because of external influences such as wind.
- The path following algorithm is what prevent the drone from completely deviating from its mission.
- The ArduCopter path following algorithm is based on the ITP position.

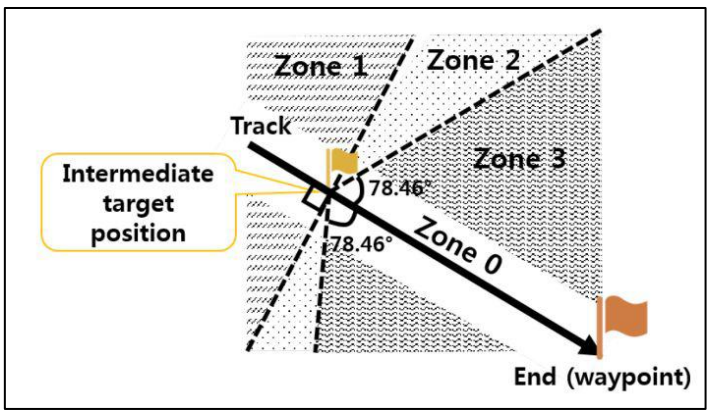
- ArduCopter periodically advances the ITP along the track in small increments and causes the drone body to move to the ITP.

ALGORITHM 2: ArduCopter path-following algorithm

```
// The original name of the function is advance_wp_target_along_track()
// advance_target() is called at 400 Hz by the ArduCopter scheduler
Function advance_target()
    // ITP is initialized with the starting point of the track
    get track from the mission uploaded by a user
    location ←the coordinates of the current location from the EKF
    distance ←the distance from the location to the track
    if distance is less than 13 m then /* Zone 0 */
        ITP advances slightly forward along the track;
    else
        perpendicular_foot ←the coordinates of perpendicular foot from the location to the track
        if ITP is closer to the end of the track than perpendicular_foot then /* Zone 1 */
            ITP stays
        else
            if distance / (the distance between location and ITP ) > 0.98 then /* Zone 2 */
                ITP advances slightly forward along the track
            else /* Zone 3 */
                ITP advances slightly forward along the track until the drone's speed along the track
                exceeds 1 m/s
                ITP halts when the drone's speed along the track exceeds 1 m/s
    controls motors to move the drone body to ITP
```

Safe Hijacking Strategy

- ITP remains unchanged if the drone is in Zone 1.
- The drone advances slightly forward along the track if in Zone 2 or 3.
- From the attacker's point of view, the ITP can be estimated to be the physical location of the drone immediately prior to GPS spoofing.
- If the drone mistakenly determines that it has deviated from the track owing to GPS spoofing, then it will move in the direction away from the manipulated location to the ITP.
- The path following algorithm uses a *leash length*, i.e., a minimum distance from the current position to the main path.
- The attacker can exploit this distance to hijack the drone's location.



Initial fake location

- The ITP is fixed or only slightly changed when the GPS location deviates from the main track by more than the leash length.
- We can hence say that the physical location of the drone immediately before hijacking is an approximation of the updated ITP.
- The attacker hence derives the possible coordinates of the initial fake location from the approximated ITP and the intended hijack direction.
- We assume the attacker begins hijacking at the physical location $p_{init} = (p_1, p_2, p_3)$.
- Objective: move the drone to p_{target}
- Hijacking direction = line between p_{init} and p_{target}
- The drone will try to move to the ITP (approximately p_{init}) if its GPS location deviates from the track more than the leash length.
- The direction from the initial fake location $a_{init} = (a_1, a_2, a_3)$ to the ITP should be the same as the hijacking direction.

$$a_{init} = p_{init} + k \cdot (p_{target} - p_{init}), \quad \text{where } k < 0 \text{ and } \|a_{init} - p_{init}\| > (leash\ length)$$

Manipulated location

ITP

Target location

- The equation is the vector form of the parametric equation of a line.
- Thus, a_{init} lies on the line that passes from pinot in the direction from target to pinot.
- In addition, the distance between pinot and a_{init} should be higher than the leash length.

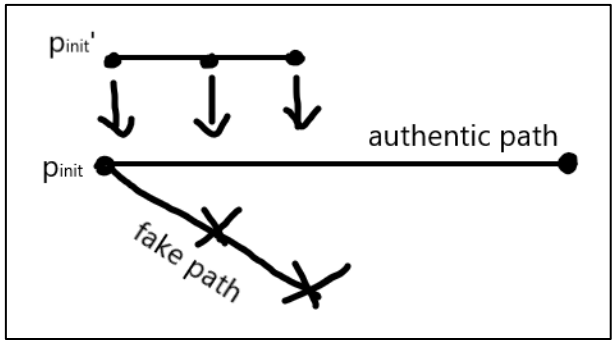
Adaptive spoofing

- The fail_count value will increase by one if the GPS location of the drone jumps to a_{init} because the GPS velocity and that measured by IMUs is not consistent.
- Therefore, the GPS velocity must be changed adaptively to be similar to the motion of the drone body to avoid triggering the failsafe mode.
- It makes the GPS location approach the drone's original track to within the leash length, but we can prevent a change in the ITP by manipulating the GPS location to a_{init} again.
- The hijacker repeats the process until the drone reaches the safe target location.
- The EKF failure count increases by one for each jump, but it will decrease to zero if the GPS velocity is consistent of that of the IMUs.
- We define as t' the current time, and as delta the GPS update period.
- We update the current fake location as

$$a_{t=t'} = a_{t=(t'-\Delta)} + (p_{t=t'} - p_{t=(t'-\Delta)})$$

Flight attitude control

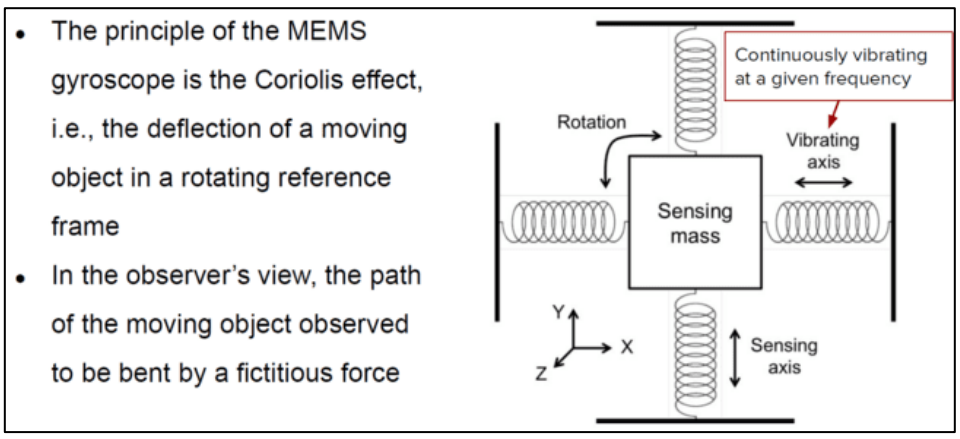
- A drone needs a controller able to adjust the horizontal speed at a rotor level.
- Indeed, multiple rotors are not always exactly the same and the center of mass cannot always be ensured.
- A flight attitude controller is hence implemented at a software level.
- Compute the proper control signal for multiple rotors based on the data from Inertial Measurement Units (IMU), including gyroscopes.



Note: the above figure represents the basic idea behind hijacking strategy for 3DR drones. The attack starts when the attacker provides an initial fake location $p_{init}' \neq p_{init}$ (the authentic position). At this point the drone tries to realign with the original path, returning to the line representing the authentic path. Anyway, providing different p_{init}'s in sequence an attacker can hijack the drone from the original path, shaping the fake path outlined in the figure. This happens because the drone tries to "move down" from the fake locations p_{init}'s to reach the authentic path but actually the drone is moving far away from that path, following the hijacking route provided by the attacker.

MEMS GYROSCOPES

An IMU is an electronic device using a combination of accelerometers, gyroscopes, and sometimes magnetometers to measure a body's specific force. In a UAV, an IMU measures the orientation, rotation, and acceleration. **Micro-Electro-Mechanical Systems (MEMS)** gyroscopes are used to make flight control modules small.



Acoustic Noise Effect

- MEMS gyroscopes are highly vulnerable to harsh acoustic noise.
- Such noise cause accuracy degradation, thus providing wrong results.
- A MEMS gyroscope has a resonant frequency related to the physical characteristics of its structure.
- Due to this resonance, if met, the gyroscope generates unexpected outputs that cause system malfunctioning.
- Usually, such frequencies should be ultrasound (above 20kHz).
- The sensitivity to noise can be exploited by an attacker.

Setup

- To effectively attack MEMS with audio, we need their resonant frequencies.
- A simple and reliable way is exhaustive search → scan with single tone sound over a chosen frequency band.
- Use a consumer grade speaker connected to a laptop and placed 10 cm above the top of the target gyroscope.
- Generate single tone noises at frequencies from 100 Hz to 30 kHz at intervals of 100 Hz.

Sound Source

- A common noise measurement unit for the loudness of sound is the Sound Pressure Level (SPL).
- Another important property is the Total Harmonic Distortion plus Noise (THD+N), i.e., the ratio of the power of the harmonics and noise components to that of a fundamental component (in %).
- Sound source = tweeter.

Noise Effect on MEMS

- Gyroscopes fixed on a stable frame in an anechoic chamber.
- In the absence of noise, the variance of the measurements should be zero → difference in standard deviation as a criterion for the resonance of the gyroscope.

Sensor	Without noise			With noise			Ratio		
	$\sigma_{X_{wo}}$	$\sigma_{Y_{wo}}$	$\sigma_{Z_{wo}}$	σ_{X_w}	σ_{Y_w}	σ_{Z_w}	$\sigma_{X_w}/\sigma_{X_{wo}}$	$\sigma_{Y_w}/\sigma_{Y_{wo}}$	$\sigma_{Z_w}/\sigma_{Z_{wo}}$
L3G4200D	3.15	2.69	2.88	12.1	22.04	4.45	**3.84**	**8.21**	**1.55**
L3GD20	2.92	2.47	2.3	62.03	76.67	3.09	**21.21**	**31.04**	**1.35**
LSM330	13.09	16.03	21.45	177.71	114.34	30.44	**13.57**	**7.13**	**1.42**
MPU6000	11.79	13.92	12.8	12.48	14.74	111.21	1.06	1.06	**8.69**
MPU6050	13.21	12.32	11.17	13.8	12.55	58.17	1.04	1.02	**5.21**
MPU6500	17.34	19.63	18.21	363.21	71.04	56.15	**20.95**	**3.62**	**3.08**
MPU9150	10.69	11.47	10.71	10.98	11.97	58.59	1.03	1.04	**5.47**

Attack Design

- The attacker, in order to design a good strategy, needs to understand what the drone is doing.
- Static analysis of the controller code to understand the reaction of sensor and actuators to noise.
- Usually, drones can support different gyroscope implementations, however the main software routing is the same for all of them.

Software Analysis

- The main processor reads the raw data from the gyroscope's registers through an I2C interface, along with the control data provided by the user.
- Raw data for each axis is stored in two 8-bit register → main inputs of the controller, which uses a PID logic to instruct rotors as follows.
    - P is proportional to the present output of the gyroscope, and if the present output is abnormally large the control from the transmitter can be ignored.
- Such raw data are them main inputs of the controller, which uses a PID logic to instruct rotors as follows.
    - I is proportional to the accumulated error between the output from the transmitter and the gyroscope, which can be ignored because usually its gain is very small.
    - D is proportional to the changes between the present output values and the gyroscope.
- Throughout the whole process the gyroscope data is not checked.

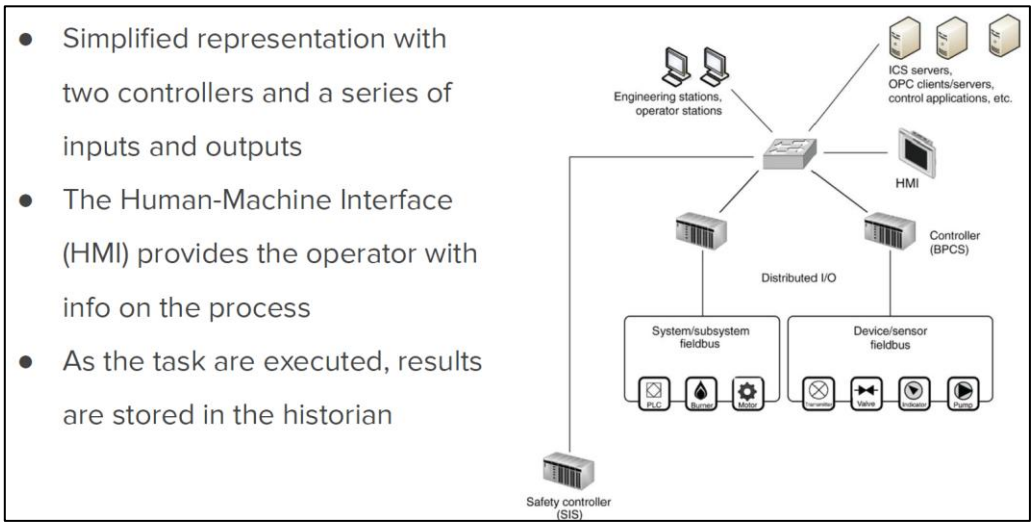
Experiment

- Attach a small Bluetooth speaker above the target system gyroscope at a distance of 10 cm to serve as attacking source.
- The sound noise is turned on while the target drones were stably maintained in the air.
- To observe the status of the drone before, during, and after the attack, the speaker is turned off, on, and off again every 10 seconds.

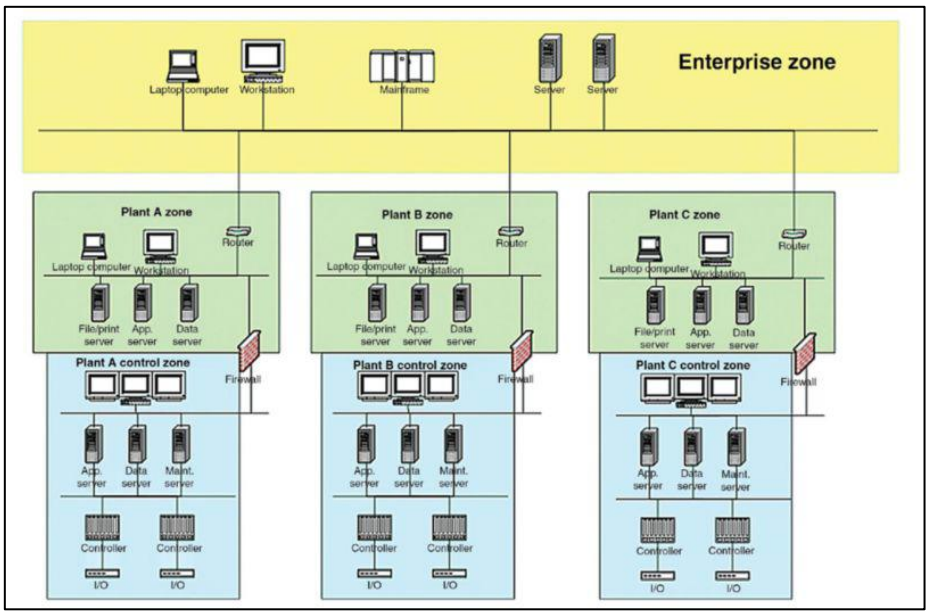
6. Industrial Systems Security

An **Industrial Control System** (ICS) is a broad class of automation systems used to provide control and monitoring functionality in manufacturing and industrial facilities. ICS is an aggregate of various system types:

- Process Control Systems (PCS)
- Distributed Control System (DCS)
- Supervisory Control and Data Acquisition (SCADA)
- Safety Instrumented Systems (SIS)



ICS may occupy very large areas, defined as **zones**, that represent groups of near devices belonging to the same ICS. A zone describes a special network that is created to expose a subset of resources to a larger, untrusted network. This is often referred to as demilitarized zone, used when enterprises want to place external-facing services, like web servers, email servers, B2B portals on the Internet while still securing their more trusted business networks.



Common recommendations to build an ICS:

- Identifying what systems need to be protected.
- Separating the systems logically into functional groups. We do not want to have a single network that connects all the ICS devices. Some devices may not need to communicate each other, leading to different logically separated zones.
- Implementing a defence-in-depth strategy around each system or group.
- Controlling access into and between each group.
- Monitoring activities that occur within and between groups.
- Limiting the actions that can be executed within and between groups.

Supervisory Control and Data Acquisition (SCADA)

SCADA is a control system architecture comprising computers, networked data applications, and graphical user interfaces for high-level supervision of machines and processes. It also covers sensing and automation devices to interface with process plants or machineries. The operator can **monitor the process and issues process commands** via the SCADA computer system. Real-time control is performed by networked modules (controllers).

SCADA can perform supervisory operations over a variety of other proprietary devices at different levels:

- **Level 0:** field devices.
- **Level 1:** industrialized input/output for direct control (PLCs/RTUs).
- **Level 2:** supervisory computers aggregating information and provide the operator control screens (plant supervision → HMI – Human Machine Interface).
- **Level 3:** production control level that monitors production and target.
- **Level 4:** production scheduling level.

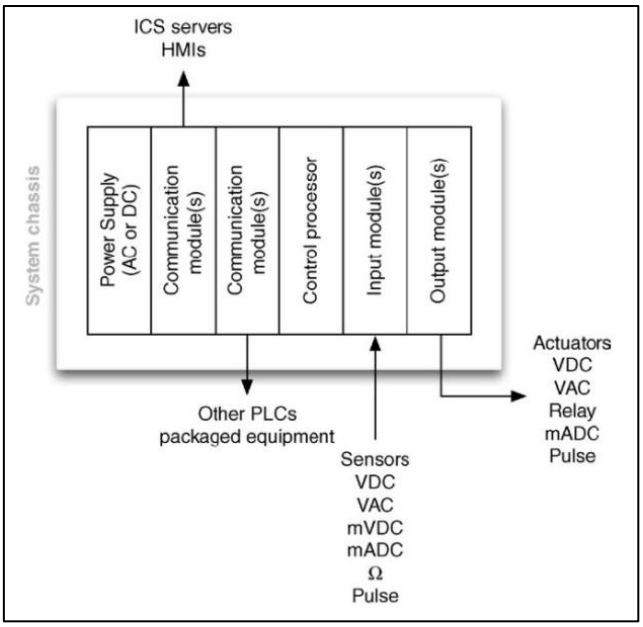
6.1 ICS components and structure

Let's see the components used in industrial networks and the role they play. Two main groups:

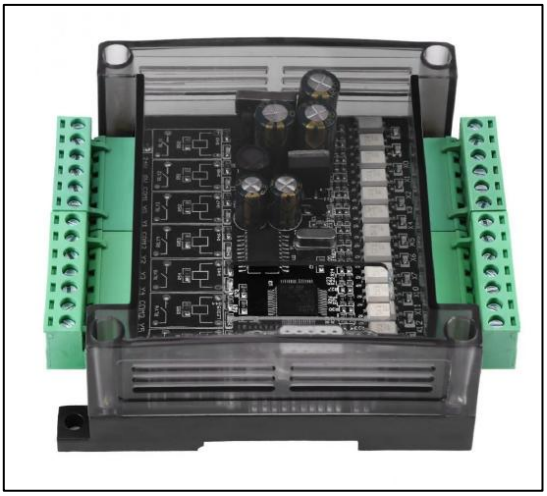
- **Field components** (simple devices), such as sensors, actuators, motor drives, gauges.
- **Control system components** (more complex devices), such as Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), intelligent electronic devices.

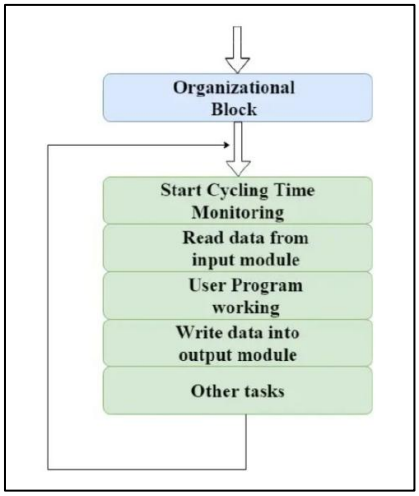
6.1.1 Programmable Logic Controller (PLC)

A Programmable Logic Controller (PLC) is a specialized industrial computer used to **automate functions** within manufacturing facilities. They are typically hardened, making them suitable for deployment in production environments. PLCs may be specialized for specific industrial uses with multiple specialized inputs and outputs. They do not use common operating systems but rely on specific application programs that allow the PLC to function **automatically generating output actions in response to specific inputs**.



PLCs are widely used in automation systems where delays or processing overhead can disrupt entire production workflows. The operation of a PLC is based on a **cyclic scanning method**, known as the scan cycle, which **continuously executes as long as the PLC is in run mode**. This involves sequentially reading inputs, executing programmed logic, and updating outputs. The logic in PLCs is implemented using standardized programming languages defined by the IEC-61131-3 standard, among which Ladder Logic are prominent. These features make PLCs an essential tool for ensuring precision and stability in industrial applications.





IEC-61131-3 standard defines both *graphical* and *textual* programming languages:

- Graphical: visual PL or block coding is a PL that lets users create programs by manipulating graphical elements.
- Textual: classical languages you always used.

LADDER LOGIC

Ladder Logic, a core programming language for PLCs, derives its name from the early electromechanical relay systems it was designed to emulate. It visually resembles a ladder, with "rungs" representing logical rules that define the relationships between inputs and outputs. **Inputs**, symbolized as relay contacts, **and outputs**, symbolized as relay coils, **are connected through a logical path**. Each rung represents a **rule** that is evaluated sequentially during the PLC's scan cycle. If all conditions along the path are true, the rung is considered true, and the output coil is energized. Conversely, if any condition is false, the output remains de-energized.

This logic is implemented using symbolic elements such as normally open contacts, represented by -[ ]-, which close when energized, and normally closed contacts, represented by -[\]-, which open when energized. Outputs, or actuators, include inactive coils, symbolized as -( )-, which energize when the rung is closed, and active coils, symbolized as -(\)-, which de-energize under the same conditions. These inputs and outputs are linked to discrete devices in manufacturing environments, such as sensors, actuators, and motor drives.

- **Logical AND:**

```
-----[ ]-------------[ ]------------------( )
  Key switch 1    Key switch 2        Door motor
```

- Door motor = key switch 1 AND key switch 2
- **Logical AND with NOT:**

```
------[ ]--------------[\]----------------( )
  Close door      Obstruction         Door motor
```

- Door motor = door closed AND NOT obstruction

- **Logical OR:**

```
--+-------[ ]-------+-----------------( )
  | Exterior unlock |                 Unlock
  |                 |
  +-------[ ]-------+
    Interior unlock
```

- Exterior unlock OR interior unlock = unlock

Industrial Stop/Start

- Latch configuration in ladder logic
- Run = (Start OR Run) AND (NOT Stop)
- In a real world example there might be hundreds or thousands of rugs
- Try it out!

```
--+----[ ]--+----[\]----( )
  |   Start |   Stop    Run
  |         |
  +----[ ]--+
       Run
```

```
-------[ ]--------------( )
       Run             Motor
```

6.1.2 Remote Terminal Unit (RTU)

A Remote Terminal Unit (RTU) **monitors the field parameters and send this data back to a central monitoring station** (e.g., an ICS server, a centrally located PLC, or HMI). RTUs typically reside in substations along a pipeline or some other remote location where there may not have easy access to electricity. They usually include communication capabilities (e.g., via modem, cellular data connections, radio or other wide area communication technology). Their communication bandwidth is limited and to maximize the information transmitted they use protocols supporting *report by exception* or other *publish-subscribe* mechanism.

An RTU is basically the interface between the physical world and the distributed control/SCADA system. They usually support the IEC 61131-3 programming standard for PLCs, and they can monitor inputs of different types:

- Digital: to acquire two state real-world information, usually accomplished by using an isolated voltage or current source to sense the position of a remote contact at the RTU side.
- Analog: can monitor analogue inputs of different types (current, voltage). RTUs can also receive analogue inputs from other on-field devices.

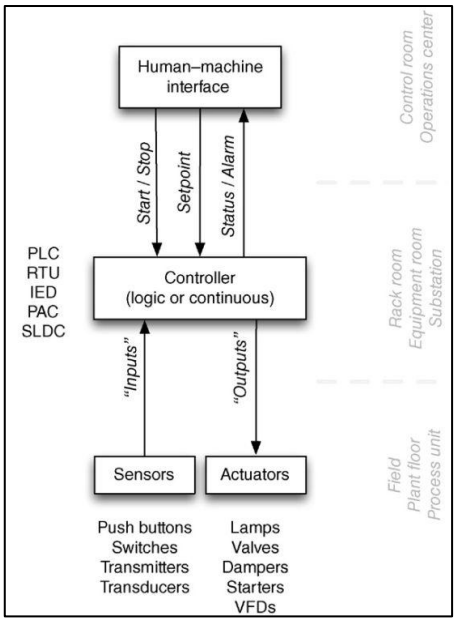
They can produce outputs of different types:

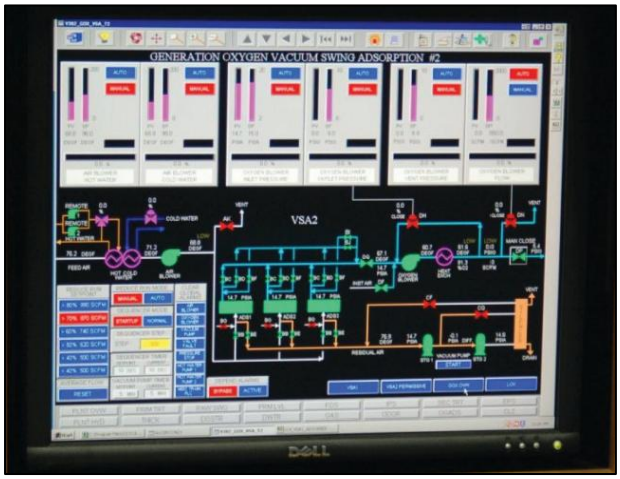
- Digital: may drive high current capacity relays to a digital output board to switch on or off on-field devices, may drive a sensitive logic input in an electronic PLC using a sensitive 5V input.
- Analog: may be included in control devices that require varying quantities.

RTUs are usually capable of executing simple programs autonomously without involving the host computers of the distributed control system or SCADA.

6.1.3 Human Machine Interface (HMI)

HMI provides the means for operators to interact with PLCs, RTUs, and IEDs. They replace manually activated switches, dials, and other electrical controls with graphical representations of the digital controls used to sense and influence the process. They allow the operator to start and stop cycles, adjust set points and perform other functions to adjust and interact with the control process. The HMI is at the top level, near the human operator.





HMI interaction mode

The above left figure shows how HMI is integrated in the overall ICS system. The human interacts with the HMI via a console, which however does not require authentication. For this reason, HMIs are hence deployed in physically strong security areas.

The HMI uses software that may come in two forms:

- The first runs on operating systems such as Windows 11 and can perform a variety of functions.
- The other form combines an industrial hardened computer, local touch panel, and typically utilize embedded operating systems. In this case, they are usually programmed with a separate computed and associated engineering software.

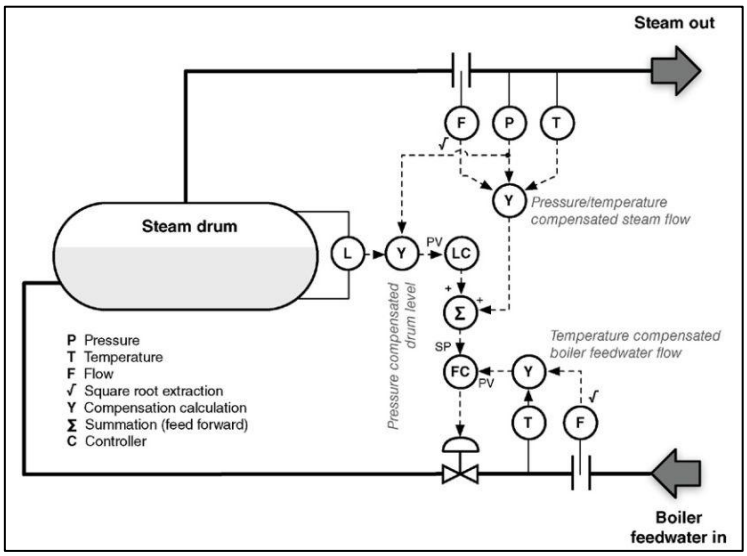
The HMI shows the process instead of the logic behind it. Since they provide supervisory data as well as control, user access control should be part of the ICS. The HMI interacts with an ICS and hence with one or more controllers.

6.1.4 Data Historian and Control Processes

Data Historian is a specialized software system that collects point values, alarm events, batch records, and other information from industrial devices and systems and stores them in a purpose-built database. Data that are historized and stored is referred to as *tags*, and car represent almost anything. Information used by both industrial operations and business management is often replicated across industrial and business networks and stored in data historians → security risk due to zones.

Control process is a general term used to define large automated process within an industrial operation. To manufacture a product or generate electricity, it might be necessary to use many control processes, each comprising one or more control loops.

Examples: one process might be to inject an ingredient into a mixer utilizing a control loop that opens a valve in response to volume measurements within the mixer, temperature, and other conditions.



6.1.5 Safety Instrumented Systems (SIS)

A **risk management strategy** should include different layers of protection to prevent a manufacturing environment to reach an unsafe operating condition. Safety Instrumented Systems (SIS) are deployed as part of a comprehensive risk management strategy. The basic process control systems are responsible for discrete and continuous control necessary for normal operations. **In the event of an abnormal situation, the SIS takes over the control**.

Risks of SISs

SIS are critical for ensuring the safe operation of industrial plants, but they also present inherent risks, particularly from cyber incidents. If compromised, the SIS may fail to perform its safety functions, potentially allowing the plant to enter a dangerous state. Moreover, since the SIS can override the Basic Process Control System (BPCS) to maintain safety, it could be exploited maliciously to cause unintended shutdowns or damage to equipment. To mitigate these risks, it is **essential to isolate the SIS from other control assets as much as possible**, ensuring it operates independently and securely to protect the plant and its operations.

SIS Isolation

Isolation is a fundamental principle in the secure and reliable operation of SIS. To minimize risks, programming of SIS is not allowed during operational mode, ensuring that unauthorized or unintended changes cannot compromise safety. Highly authorized applications, such as SIS programming tools and engineering workstations, are often disconnected from ICS networks and only reconnected when necessary for maintenance or updates. Regular testing of SIS functionality is crucial to confirm their reliability, and this process should include comprehensive cybersecurity assessments to identify and address potential vulnerabilities.

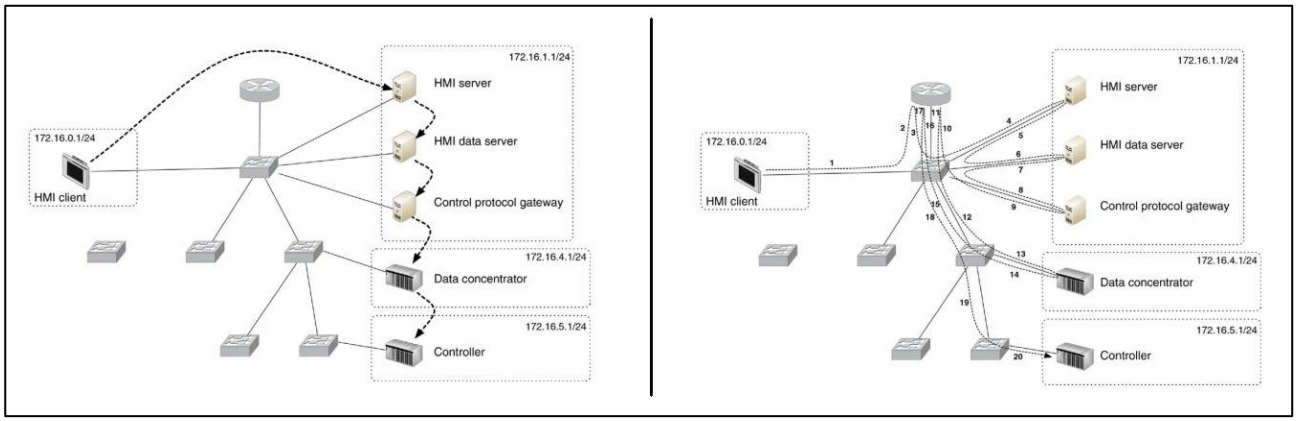
6.2 Industrial networking

An industrial network is any network that supports the interconnectivity of and communication between devices that make up or support an ICS. They may be **local-area switched networks**, as common with Distributed Control Systems, or **wide-area routed networks** more typical of SCADA architectures. Most are Ethernet and IP-based and consist of both wired and wireless connectivity.

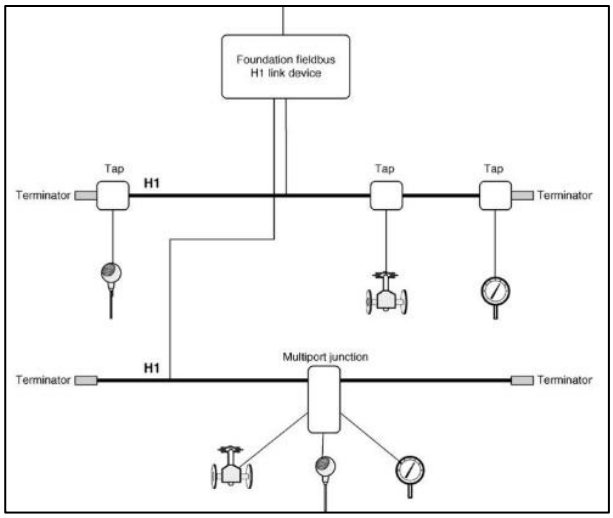
Availability is prioritized over data integrity and confidentiality. Therefore, there is a great use of real-time protocols, UDP transport, and fault-tolerant networks interconnecting endpoints and servers. Bandwidth and latency are extremely important, because the applications and protocols in use support real-time operations that depend on deterministic communication often with precise timing requirements. Ubiquitous connectivity might not be something we want to achieve.

The ICS requirements dictate the topology of the networks:

- High reliability and resiliency dictate the use of ring or mesh network topologies.
- Real-time operations and low latency require the minimization of switching and routing hops.



As we can observe from the above figure, 5 sessions require 20 paths to be traversed. It is therefore necessary to minimize latency wherever possible to maintain real-time and deterministic communications → we need to use switching unless traversing a functional boundary. We also need to use low level (Ethernet) firewalls and prefer a bridged mode, i.e., avoid relying on IP routing.

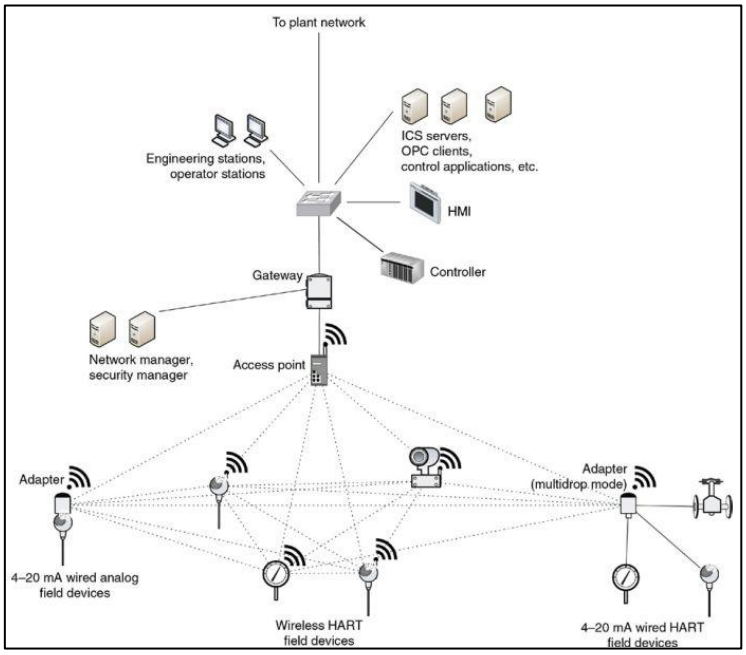


As we go lower into the control environment, functionality becomes more specialized. Introduction of open and/or proprietary protocols in either their native form or adapted to work over Ethernet.

Figure on the left: Fieldbus.

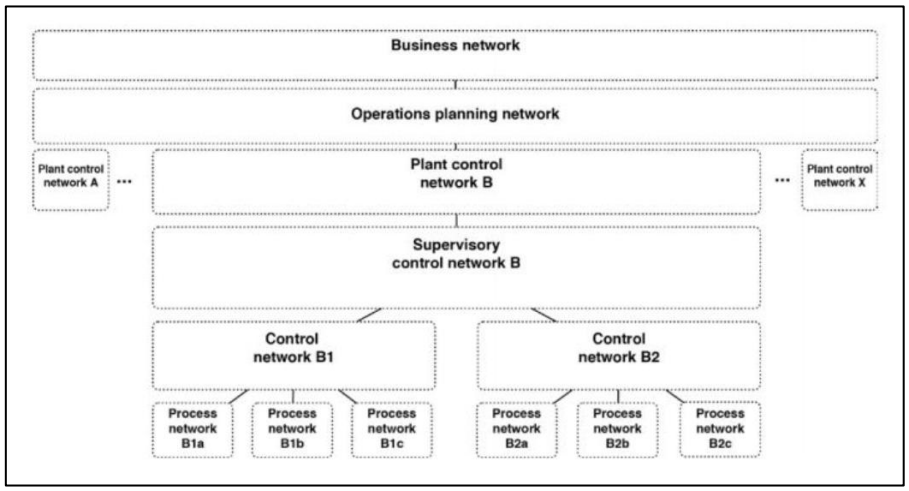
6.2.1 Wireless networks

Wireless networks are bound by the same design principles as wired networks. However, they are more difficult to physically contain because they are bound by the range of the radio wave propagation from an access point rather than by physical cables and network interfaces.
This means that **any device** that is equipped with an appropriate receiver and is within the range of a wireless access point **can physically receive wireless signals**. There is no sure way to prevent this physical (wireless) access. Instead, it is possible to block transmissions by using jammers or signal-absorbing materials, these measures are costly and rarely implemented. Typically achieved thorough radio frequency surveys in order to not only place antennas in optimal locations, but also prevent unnecessary transmission of signals into untrusted and unrestricted areas.



6.2.2 Network segmentation

Segmentation is a critical strategy in industrial networking, involving the **division of networks or zones into smaller, manageable units to enhance performance and cybersecurity**. Typically implemented at Layer 3 of the OSI model using network devices with routing functions, segmentation breaks up large, flat Ethernet networks into discrete segments by blocking broadcasts. This division not only improves network efficiency but also provides inherent access control at each boundary, reducing the risk of unauthorized communication. Supporting zone segmentation, segmentation creates hierarchical network architectures where communication between zones or networks may require traversing multiple layers. However, while such architectures enable communication flows, it is essential to carefully evaluate and restrict traffic between segments to prevent unnecessary or malicious interactions, ensuring robust network security in industrial environments.



Physical vs logical segmentation

- Physical segmentation refers to the use of two separate physical network devices (both passive and active components) to perform the isolation between networks.
- Logical segmentation refers to the use of logical functions within a single network device to achieve essentially the same result.

Physical separation of systems ("air gap" separation) is still widely used in industrial networks when talking about the coexistence of basic process control and safety systems overseeing the same process.

Creating zones from segment

Creating zones from segments in industrial networks involves tailoring the architecture to the unique operational environment and the way systems integrate with ancillary setups. The focus is not just on the equipment used but on ensuring that systems are deployed and function cohesively without causing unintended interactions. A safety level is assigned to each system, enabling the application of targeted measures to ensure reliable and secure performance. Assets at an industrial site are grouped into zones based on their relative security requirements, creating layers of protection and facilitating more effective management of risks.

Identifying zones

Identifying zones, however, presents challenges, particularly in establishing base requirements to determine the appropriate placement of assets within a zone. These requirements are guided by goals such as ensuring proper communication and interaction among systems and controlling physical access to assets. Zones based on network connectivity are more straightforward, as they align with existing segmentation practices. By systematically categorizing assets and interactions, organizations can design zones that strengthen security and operational integrity across the industrial network.

6.2.3 Industrial networks protocols

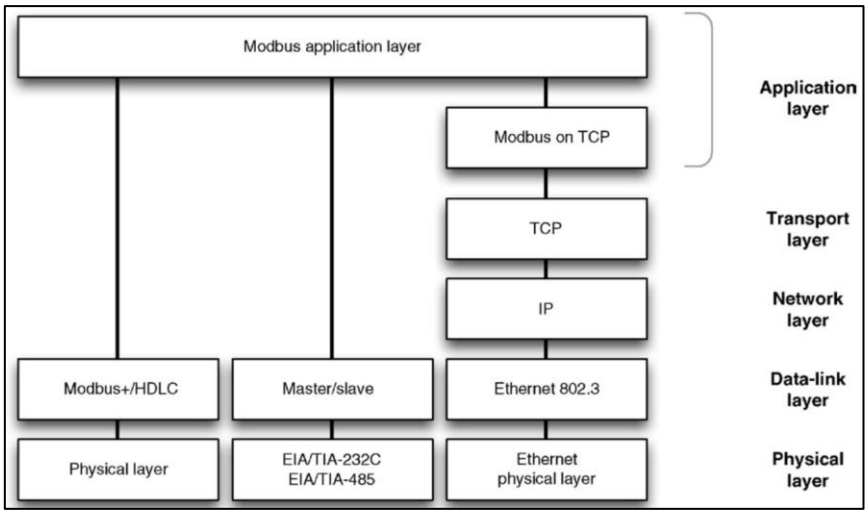
There are many highly specialized protocols used for industrial automation and control. Since they are designed for efficiency, many of them usually forgo any feature or function that is not absolutely necessary. Many of them have been adapted to run on top of Ethernet and IP. We can distinguish two main categories:

- **Fieldbus**: commonly found in process and control.
- **Backend**: commonly deployed on or above supervisory networks, and are used to provide efficient system-to-system communication.

6.2.3.1 MODBUS

Modbus is a Layer 7 application-layer messaging protocol within the OSI model, designed in 1979 to facilitate communication between process controllers and real-time computers. Over the years, it has become one of the most widely used protocols in ICS architectures, serving as a de facto standard due to its free distribution and robust support from the Modbus Organization.

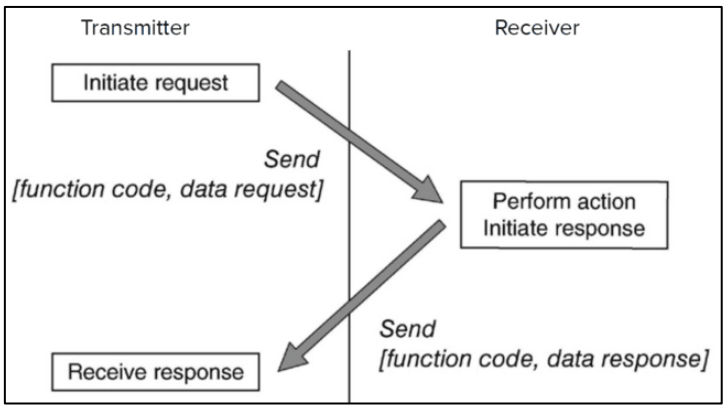
Modbus operates on a simple request/reply methodology, enabling efficient communication between interconnected devices. Its minimal processing overhead makes it ideal for low-complexity devices, such as motors and sensors, to exchange data with more sophisticated systems like PLCs and RTUs. By supporting streamlined data communication, Modbus plays a critical role in connecting supervisory ICS components with field-level devices.



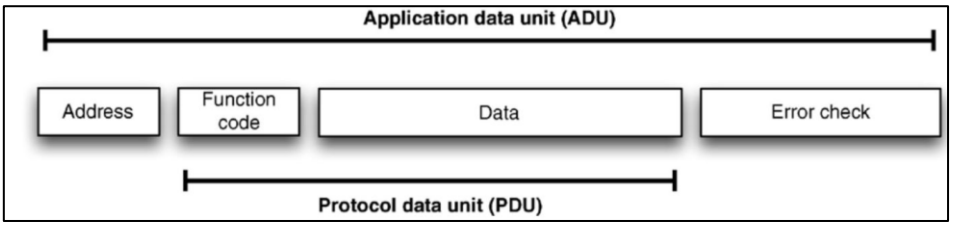
As mentioned before, it is a request/response protocol using three distinct protocol data units:

- Modbus Request
- Modbus Response
- Modbus Exception Response

Up to 32 devices can be connected on a single RS-485 serial link, requiring each device being assigned a unique address. A command is addressed to a specific Modbus address. While other devices may receive the message, only the addressed device will respond.



A “transaction” begins with the transmission of an initial Function Code and a Data Request within a Request PDU. If there are no errors, the receiver will respond with a Function Code and Data Response within a Response PDU. If there are errors, the device will respond with an Exception Function Code and Exception Code within a Modbus Exception Response.

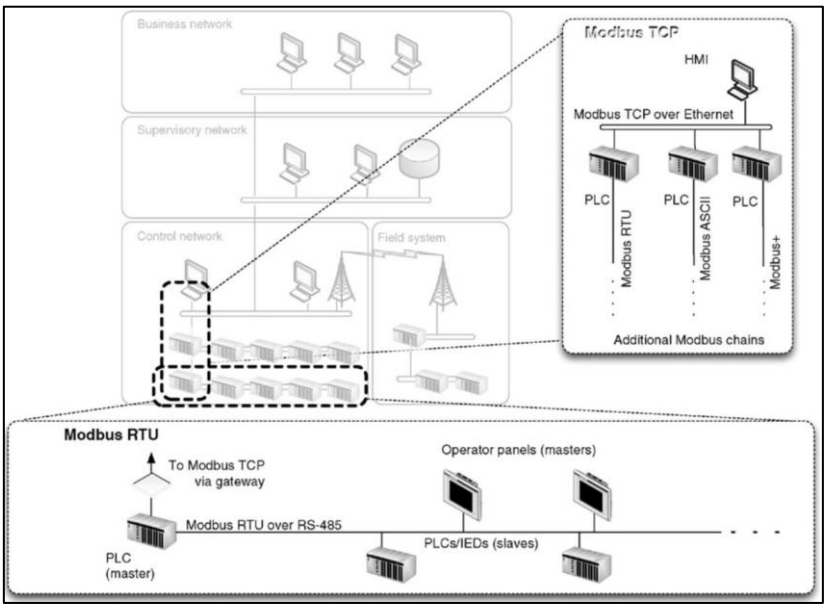


Function Codes used in Modbus are divided into three categories and provide the device vendor with some flexibility in how they implement the protocol.

- Function codes in the range of 01–64, 73–99, and 111–127 are defined as “Public” and are validated by the Modbus-IDA community and are guaranteed unique.
- “User-Defined” function codes in the range 65–72 and 100–110 are provided to allow a particular vendor to implement functionality to suit their particular device and application.

Function Codes and Data Requests can be used to perform a wide range of commands (e.g., Read the value of a single register, Write a value to a single register, Read a block of values from a group of registers, Obtain device diagnostic data).

Modbus is **typically deployed between PLCs (client) and HMIs (server)**. Modbus devices may also take the role of server in some communications, while being client in other communications.



Security concerns

Modbus presents significant security concerns due to its lack of modern protective features.
The protocol **does not incorporate authentication mechanisms**, relying solely on a valid Modbus address, function code, and associated data for communication. While this requires knowledge of the target device's registers or coils, such information can often be obtained through methods like traffic analysis.
Furthermore, Modbus **lacks encryption**, transmitting commands and addresses in clear text, making it vulnerable to interception, spoofing, and replay attacks. Communications with Modbus devices may also inadvertently expose sensitive information about device configuration and usage.
Additionally, the **absence of an application-layer message checksum** allows attackers to spoof commands by assembling Modbus Application Data Units (ADUs) with tailored parameters, as integrity verification is only performed at the transmission layer. These vulnerabilities highlight the critical need for supplementary security measures when deploying Modbus in ICS environments.

Security recommendations

- Should only be used to communicate between sets of known devices.
- In this way it can be easily monitored by establishing clear network zones and by baselining acceptable behaviour.
- This baseline behaviour can then be used to establish access controls on the conduit into the zone via appliances that provide protocol inspecting and filtering capabilities.
- Suspicious messages: Modbus TCP packets that are of wrong size or length, function codes that force slave devices into a “listen only” mode, function codes that restart communications.

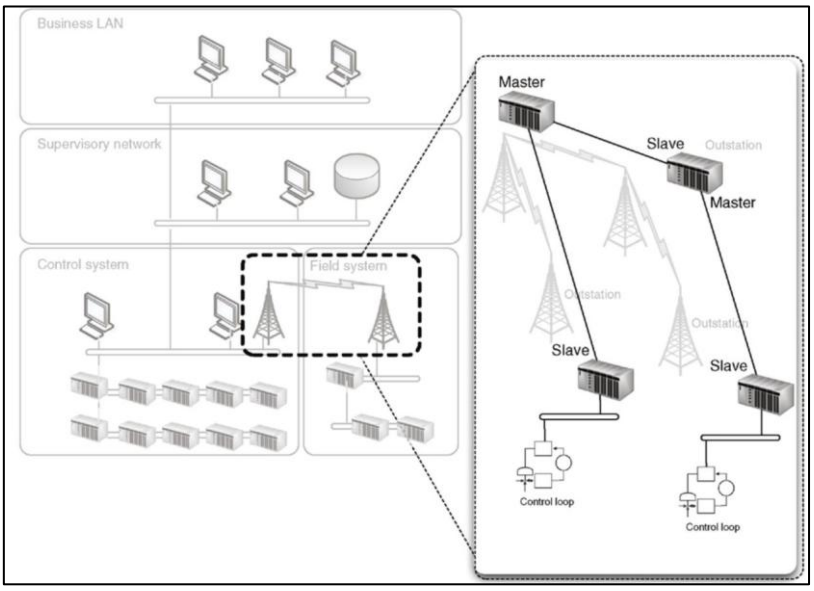
6.2.3.2 Distributed Network Protocol (DNP3)

DNP3, originally designed for reliable data transfer in industrial applications, was extended in 1998 to operate over IP by encapsulating its communications in TCP or UDP packets. This adaptation has broadened its application across various industries. Renowned for its reliability, DNP3 is also efficient and well-suited for real-time data transfer, leveraging standardized data formats and supporting time-stamped and time-synchronized data. These features enhance its capacity for precise and dependable communication. Additionally, the protocol incorporates robust error-checking mechanisms, with a single DNP3 frame capable of including up to 17 Cyclic Redundancy Checks (CRCs). Variants of DNP3 also support link-layer authentication, further strengthening its security in critical infrastructure applications.

How DNP3 works

DNP3 operates as a communication protocol that facilitates message exchange between control system devices, such as RTUs and Intelligent Electronic Devices (IEDs). It supports various functions, including sending control commands and transmitting binary or analog data for direct device interaction. The protocol efficiently identifies remote device parameters by using event data classes (1 through 3) to organize incoming messages, allowing the master station to retrieve only new information triggered by point changes or events. Initial communication often begins with a class 0 request from the master station, used to read all point values into its database. Subsequent interactions typically involve specific poll requests for targeted data classes or control and configuration commands sent from the master station to outstations, streamlining the data exchange process while ensuring that only pertinent updates are transmitted.

Where to use DNP3



Security concerns

Much attention is given to the integrity of the data frame, there is no authentication or encryption inherent within DNP3. Some examples of realistic hacks against DNP3 include the use of MitM attacks to capture addresses, which can then be used to manipulate other system components.

Examples: Turning off unsolicited reporting to suppress alarms, Spoofing unsolicited responses to the master station to falsify events, Performing a DoS attack through the injection of broadcasts.

Secure DNP3

Secure DNP3 is a variant of the standard DNP3 protocol that enhances security by incorporating authentication into the request/response process. Authentication is triggered by a challenge issued by the receiving device, which can occur at various points: during session initiation, after a preset period, or in response to critical requests, such as write operations. The authentication process relies on a unique session key, which is hashed together with the message data from both the sender and the challenger. This method ensures that only authorized devices can interact with the system, making it extremely difficult for attackers to manipulate data, inject malicious code, spoof devices, or hijack communications. By adding this layer of security, Secure DNP3 significantly reduces the risk of unauthorized access and improves the integrity of data exchanges in critical infrastructure systems.

6.3 Attacks and detection in ICS

The major distinction of control systems with respect to other IT systems is the interaction with the physical world. While measures for standard IT systems may be enough, they might not be sufficient for ICS. It is important not only to protect information, but also to protect estimation and control algorithms and how they impact the physical world.

Risk assessment

- Risk management = shifting the odds in your favour by finding the alternative that minimizes the impact of certain events.
- The best well-known metric is the average loss $R_\mu = \mathbb{E}[L] \approx \sum_i L_i p_i$.
- Multiply the loss implied by event i by the probability that event i occurs (in our case this event is represented by a possible attack on sensor i).
- There are also other measures, for instance the variance of the losses.
- $R_\chi = \mathbb{E}[L^2] - R_\mu$
- However, this is important over long periods of time so not very suitable for our case, average loss is more than enough.

The main idea is to study each sensor's loss in the ICS in a way to assess the impact of possible attacks.

ATTACK MODEL

- We focus on attacks on sensor networks and their effect on process control.
- In our case, p_i denotes the likelihood that an attacker will compromise sensor i, and L_i denotes the loss associated with that particular compromise.
- We assume that the likelihood is the same for all sensors, and we focus on the estimate of the potential loss.
- Let's consider a network of p sensors with measurement vector $y(k) = \{y_1(k), y_2(k), \cdots, y_p(k)\}$
- All sensors have a dynamic range that defines the domain of y for all k.
- I.e., $\forall k, y_i(k) \in [y_i^{min}, y_i^{max}] = \mathcal{Y}_i$
- We assume each sensor has a unique identity protected by a cryptographic key.
- Let $\tilde{y}(k) \in \mathbb{R}^p$ denote the received measurement by the controller at time k.
- Based on these measurements, the control system defines control actions to maintain certain operational goals.
- We assume that attacked signals lie within $\mathcal{Y}_i$
- If the malicious value is outside the legitimate range, it can be simply detected (as an attack) by fault-tolerant algorithms.
- Let $\mathcal{K}_a = \{k_s, \ldots, k_e\}$ denote the attack duration.
- Given attack signal $a_i(k)$, the general model for the observed signal.

$$\tilde{y}_i(k) = \begin{cases} y_i(k) & \text{for } k \notin \mathcal{K}_a \\ a_i(k) & \text{for } k \in \mathcal{K}_a, a_i(k) \in \mathcal{Y}_i \end{cases}$$

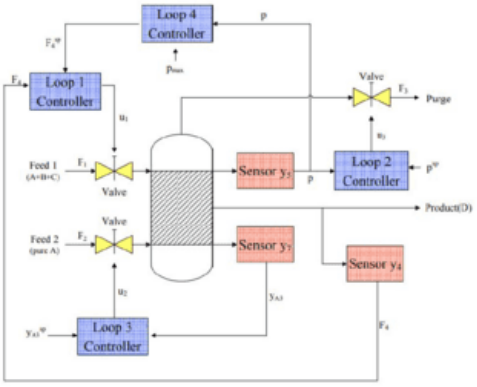
- This model can be used to represent two types of attacks.
- **Integrity attacks**: in this case the attacker compromises the senor and the value it reports, therefore $a_i(k)$ can be some arbitrary non-zero value.
- **DoS attack**: the controller will notice the lack of new measurements and will react accordingly. An intuitive response is the use of the last received valid signal.

Experiments

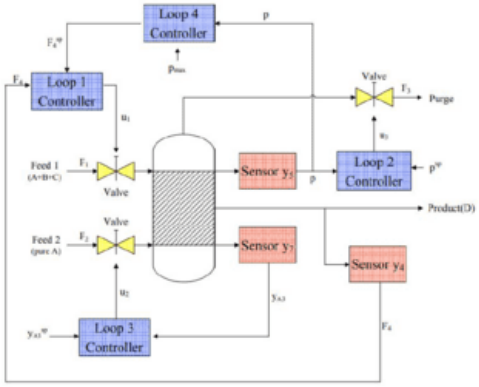
Controllers are not stupid, so let's see what effect these attacks might have on an industrial process and decide which attack has the most impact. **Tennessee-Eastman process control system (TE-PCS)** and multi-loop PI control are used to collect benchmark values and test ICS via simulations. Chemical process introduced for education purposes and for benchmarking.

- Consists of an irreversible reaction which occurs in the vapour phase inside a reactor of a fixed volume V
- Non-condensable reactants A and C react in the presence of an inert B to form a non-volatile liquid D

$$A + C \xrightarrow{B} D$$



- F_i = i-th flow
- F_1 contains A, C and trace of B
- F_2 is pure A
- F_3 is the purge containing vapours of A, B, and C
- F_4 is the exit for liquid product D



Control objectives

- Regulate the rate F_4 of production of product D at a set point F_4^{sp}
- Maintain the operating pressure of the reactor P below the shutdown limit (safety).
- Minimize C, the operating cost, that is a linear function of the purge loss of A and C relative to the production rate of D $r_D = k_0 y_{A3}^{v1} y_{C3}^{v2} P^{v3}$
- Where y_{A3} and y_{C3} denote the respective fractions of A and C in the purge, the vs are given constants.

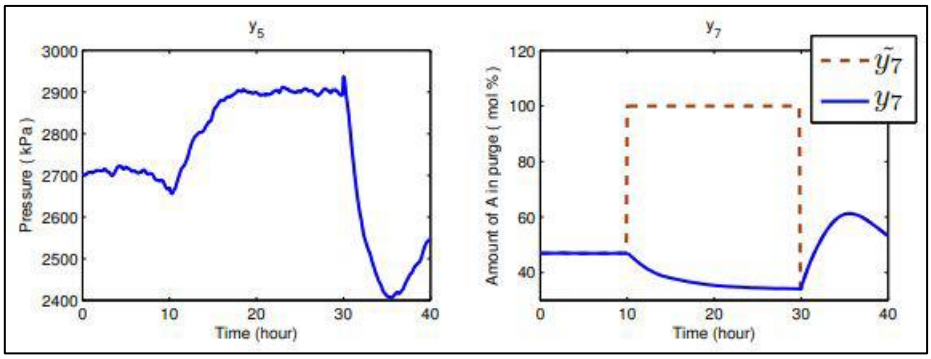
Input variables

- Four input variables (command signals) available to achieve the control objectives
- u_1, u_2 and u_3 are triggers of the actuators that can change the position of the respective valves
- u_4 Is the set point for the proportional controller for the liquid inventory
- The input variables are used by the controller as follows
- The production rate $y_4 = F_4$ is controlled using Feed 1 u_1 by loop-1 controller
- Pressure $y_5 = P$ is controlled using the purge rate u_3 by loop-2 control

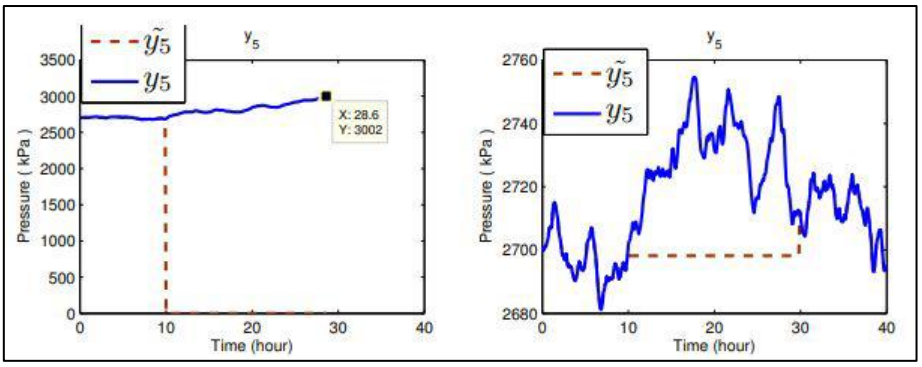
- Partial pressure of product A in the purge $y_7 = y_{A3}$ is controlled using Feed 2 u_3 by loop-3 controller
- When u_3 saturates, the loop-4 controller uses u_1 to control the pressure P
- In steady state, the production rate F_4 is $100\ kmol\ h^{-1}$, the pressure is P is $2700\ KPa$ and the fraction of A in the purge is $47\ mol\%$
- We assume an attacker wanting to obtain an unsafe pressure in the chemical reactor, i.e., > 3000 kPa and having access to a single sensor at the time

ATTACKING TE-PCS

- We assume an attacker wanting to obtain an unsafe pressure in the chemical reactor, i.e., > 3000 kPa and has access to a single sensor at the time.
- We also assume that $L_i > L_j$ when an attack i can drive the system to an unsafe state and an attack j cannot.
- Spoiler: the most effective attacks are max/min attacks, i.e., when $a_i(k) = y_i^{min}$ or $a_i(k) = y_j^{max}$, although not all of them can drive the system to un unsafe state.
- By attacking a sensor, we expect the controller to respond with incorrect control signals: by pushing y_7^{max} from t=0, to 30 the controller believes there is a large amount of component A in the tank.
- However, the controller can bring back the system to the steady state when the attack finishes.
- Furthermore, the pressure never reaches the critical 3000 kPa value.



- By launching attack y_5^{min} the attacker turns down the purge valve to increase the pressure and prevent the liquid products from accumulating.
- The pressure of the tank keeps increasing past 3000 kPa and the system operates in an unsafe state.
- However, it takes about 20 hours to cause an explosion.
- This delay (slow-dynamics) gives human system operators enough time to observe the unusual phenomenon and take proper action.



Attack detection

Detecting attacks to control systems can be formulated as an anomaly-based intrusion detection. Instead of modelling traffic or software behaviour, we model the physical system. If we know how the output sequence of the physical system should react to the control input sequence, then any attack to the sensor data can be potentially detected by comparing the expected output with the received signal.

Needs: model of the system and anomaly detection algorithm.

System model

- Developing a model for the system under analysis is not trivial.
- First principles model: derived from the laws of physics.
- Empirical input and output data: learn the model.
- To facilitate this task, most industrial control vendors provide identification packages, i.e., tools to develop models from physical data.
- The most common models are linear systems, which model dynamics that are linear in state x and control u $x(k+1) = Ax(k) + Bu(k)$
- Assuming that the system is monitored by a sensor network of p senrs, we obtain the measurement sequence from the observation equation.
- $\hat{y}(k) = Cx(k)$
- These values represent the estimated measurements collected by the sensor, and C is the output matrix $\hat{y}(k) = (\hat{y}_1(k), \ldots, \hat{y}_p(k))$

Detection methods

- Sequential detection theory: methodology to detect anomalies in real-time without considering a fixed measurement time, but relying on an online-chosen detection time
- Problem formulations of this kind are called optimal stopping problems
- Two such problems formulation are sequential detection (sequential hypothesis testing) and quickest detection (change detection)

Optimal stopping problems

- Let us consider a given time series sequence $z(1), z(2), \ldots, z(N)$
- Goal: determine the minimum number N of samples to observe before making a decision on one of two hypotheses: H_0, i.e., normal behavior or H_1, i.e., attack.

- Sequential detection strategy: we assume that the time series originated from either the normal or attack hypothesis and we want to decide on the hypothesis in the minimum amount of time.
- Change detection: the sequence originated from the normal hypothesis and changed to the other. We want to detect this change ASAP.

Sequential detection

- *False alarm probability*: detect an attack when there is none.
- *Missed detection probability*: probability of not detecting an attack.
- Goal: given fixed false alarm and detection probability, minimize the number of observations needed to make a decision.
- Solution given by the Sequential Probability Ratio Test (SPRT).
- Also referred to as Threshold Random Walk (TRW) in security papers and used to detect portscans, worms, and botnets.

Sequential Detection: SPRT

- Assume that observations $z(k)$ under hypothesis H_j are generated with a probability distribution p_j
- The SPRT algorithm can be described by the following equations

$$S(k+1) = \log \frac{p_1(z(k))}{p_0(z(k))} + S(k)$$ Cum.Sum. of LLRs

$$N = \inf_n \{n : S(n) \notin [L, U]\},$$

- With S(0) = 0
- Decision rule $d_N = \begin{cases} H_1 & \text{if } S(N) \geq U \\ H_0 & \text{if } S(N) \leq L, \end{cases}$
- Where $L \approx \ln \frac{b}{1-a}$ and $U \approx \ln \frac{1-b}{a}$, a = desired P of FA and b= P of MD

Change Detection

- Detect a possible change in the generation problem at an unknown change point k_s
- CUSUM and Shiryaev-Roberts statistics are the two most commonly used algorithms for change detection.
- Given a fixed false alarm rate, CUSUM aims at minimizing the time N > k_s for which the test stops and decides that a change has occurred.
- Very similar to SPRT.
- Let S(0) = 0
- CUSUM statistics is updated according to

$$S(k+1) = \left(\log \frac{p_1(z(k))}{p_0(z(k))} + S(k) \right)^+$$

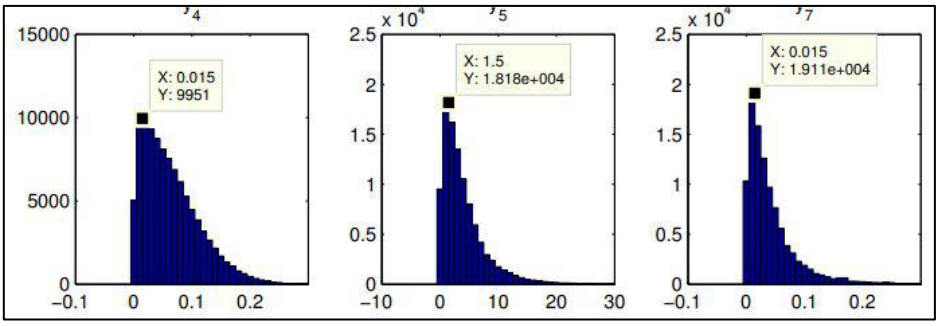
- Where $(a)^+ = a$ if $a \geq 0$, 0 otherwise
- The stopping time is $N = \inf_n \{n : S(n) \geq \tau\}$
- Where the threshold tau is selected based on the false alarm constraint
- Notice: CUSUM is SPRT with L = 0, and $U = \tau$, and whenever the statistic reaches the lower bound L it restarts

Detecting compromised sensors

- In general, we do not know the probability distribution for an attack
- The attacker can indeed select arbitrary and non-stationary sequence, hence assuming a fixed probability limits our detection capabilities
- We hence use ideas from nonparametric statistics
- We do not assume a parametric distribution for p_j, j = {0, 1}
- We place mild constraints on the observation sequence
- One of the simplest constraints is to assume the expected value of the random process $Z_i(k)$ that generates the sequence $z_i(k)$ under H_0 is less than 0 (i.e., $\mathbb{E}_0[Z_i] < 0$), and the expected value of $Z_i(k)$ under hypothesis H_1 is greater than zero (i.e., $\mathbb{E}_1[Z_i] > 0$)
- To achieve this condition, we define $z_i(k) := \|\tilde{y}_i(k) - \hat{y}_i(k)\| - b_i$ where the last terms is a small positive constant such that $\mathbb{E}_0[\|\tilde{y}_i(k) - \hat{y}_i(k)\| - b_i] < 0$

Selecting b

- To select the value b_i for each sensor, we need to estimate the expected value of the distance between the linear model estimate and the sensor measurement (i.e., without attack), respectively.
- Run experiments to estimate probabilities.

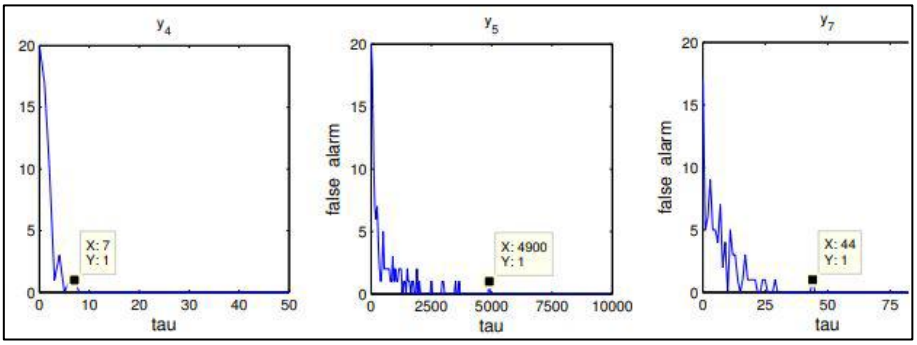


Detecting compromised sensors

- The nonparametric CUSUM statistic for sensor i is then
$$S_i(k) = (S_i(k-1) + z_i(k))^+, S_i(0) = 0$$
- The corresponding detection rule
$$d_{N,i} \equiv d_\tau(S_i(k)) = \begin{cases} H_1 & \text{if } S_i(k) > \tau_i \\ H_0 & \text{otherwise.} \end{cases}$$
where tau is the threshold selected based on the false alarm rate for sensor i
- P. of FA exponentially decreases with increasing tau
- Time to detect attack inversely proportional to b

Computing tau

- Run the system many times without attacks and count how many times the system detects ana attack.
- In general we would like to select tau as high as possible to avoid false alarms.
- This however increases the detection time.



Stealthy Attacks

- Sometimes attackers are able to adapt to the system and develop solutions to evade detection schemes.
- We now consider an attacker aware of the existence and working principle of our detection system.
- We assume the attacker knows: the exact linear model that we use (i.e., matrices A, B, and C), the parameters tau and b, and the control command signals.
- Goal: raise the pressure in the tank without being detected (i.e., keeping the controlled statistics below threshold tau).

Surge Attacks

- The attacker tries to maximize the damage as soon as possible
- However, when the statistics reaches the threshold it stays at the threshold level for the remaining time of the attack
- Means $S_i(k) = \tau$
- To do this, the attacker needs to solve $S_i(k) + \sqrt{(\hat{y}_i(k) - \tilde{y}_i(k))^2} - b_i = \tau_i$
- The resulting attack for y_5 and y_4 is

$$\tilde{y}_i(k) = \begin{cases} y_i^{min} & \text{if } S_i(k+1) \leq \tau_i \\ \hat{y}_i(k) - |\tau_i + b_i - S_i(k)| & \text{if } S_i(k+1) > \tau_i \end{cases}$$

- For y_7

$$\tilde{y}_7(k) = \begin{cases} y_7^{max} & \text{if } S_{y_7}(k) \leq \tau_7 \\ \hat{y}_7 + |\tau_7 + b_7 - S_{y_7}(k)| & \text{if } S_{y_7}(k) > \tau_7 \end{cases}$$

Bias Attacks

- In a bias attack the attacker adds a small constant c_i at each time step

$$\tilde{y}_{i,k} = \hat{y}_{i,k} - c_i \in \mathcal{Y}_i$$

- In this case, the nonparametric CUSUM statistic can be written as

$$S_i(n) = \sum_{k=0}^{n-1} |\hat{y}_i(k) - \tilde{y}_i(k)| - nb_i$$

- Assuming the attack starts at time 0, and that wants to be undetected for n steps, the attacker needs to solve

$$\sum_{k=0}^{n-1} c_i = \tau_i + nb_i \quad \rightarrow \quad c_i = \tau_i/n + b$$

- This attack creates a bias of $\tau_i/n + b_i$ for each attacked signal.
- If an attacker wants to maximize the damage, the attacker needs to select the smallest n.
- Since $\tilde{y}_i \in \mathcal{Y}_i$, it is reduced to an impulse attack.
- If the attacker wants to attack for a long time, then n will be very large.
- If n is large, the bias is smaller.

Geometric Attacks

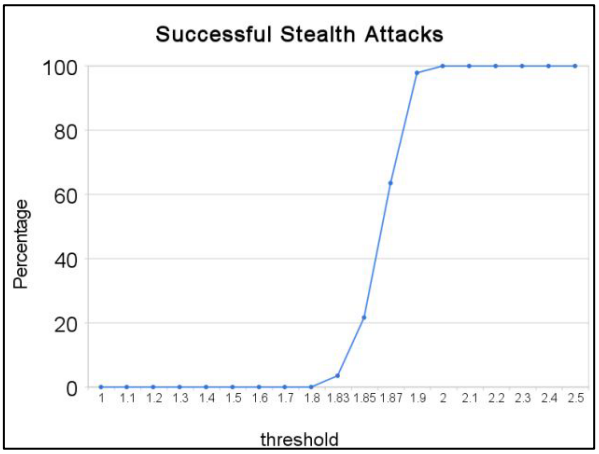
- The attacker wants to drift the value very slowly at the beginning and maximize the damage at the end.
- Combine slow initial drift of the bias attack with a surge attack at the end to cause maximum damage.
- Given $\alpha \in (0, 1)$, the attack is $\tilde{y}_i(k) = \hat{y}_i(k) - \beta_i \alpha_i^{n-k}$
- We need to find alpha and beta such that $S_i(n) = \tau_i$
- Assume the attack starts at time k = 0 and that the attacker wants to be undetected for n steps
- Need to solve the following equation $\sum_{k=0}^{n-1} \beta_i \alpha_i^{n-k} - nb_i = \tau_i$
- The addition is a geometric progression

$$\sum_{k=0}^{n-1} \beta_i \alpha_i^{n-k} = \beta_i \alpha_i^n \sum_{k=0}^{n-1} (\alpha_i^{-1})^k = \beta_i \frac{1 - \alpha_i^n}{\alpha_i^{-1} - 1}$$

- By fixing alpha, the attacker can select proper beta to satisfy the above equation

Detectability

- The value tau impacts on how good we are in detecting attacks.
- Thus, a relevant information is the percentage of successful stealthy attacks for different values of tau.
- Parameterization of threshold as $\tau_i^{test} = p\tau_i$
- Success of geometric stealthy attack in bringing pressure above 3000kPa.
- Attack undetected.



Although attack may go undetected, they usually do not impact on the safety. The most successful attack is the geometric one.

6.4 Stuxnet

There are many examples of things that can go wrong in an ICS:

- **Computer-based accidents**
  - A whole nuclear power plant was forced into emergency shutdown for 48 hours after an operator installed a software update on the business network used to monitor chemical and diagnostic data.
- **Non-targeted attacks**
  - Incidents caused by the same attacks that any other internet-connected PC may suffer, such as worms infecting computers, or controllers sending spam messages.
- **Targeted attacks**
  - The attacker knows that the target is a control system and hence tailors the attack strategy with the aim of damaging the physical system under control.
  - Although physical attacks were already famous, cyber-attacks are nowadays more and more exploited as they are cheap, they have long range, and are easy to replicate and coordinate.
  - There have been many attacks, however none of them has shown the actual vulnerabilities of control systems as Stuxnet did.

Stuxnet is a malicious computer worm first discovered in 2010 that targets Supervisory Control and Data Acquisition (SCADA) systems.

- It is the first concrete example of a cyberweapon.
- It specifically targets PLCs.
- It exploited four zero-day flaws.
- Targets machines using Windows OS, then Siemens Step7 Software, and finally the PLCs.
- Compromised PLCs, collecting information on industrial systems and causing the fast-spinning centrifuges to tear themselves apart.

Attack scenario

- As each PLC is configured in a unique manner, the attackers would first need the ICS's schematics (stoel from an insider or retrieved by an early version).
- Once having that, the attacker can develop the latest version of Stuxnet.
- In Stuxnet the attacker's malicious binaries container driver files that needed to be digitally signed. The attackers compromised two digital certificates to achieve it (stole them).
- To infect their target, Stuxnet would need to be introduced into the target environment → infect a willing or unwilling party.
- The original infection may have been introduced by removable drive.
- Once Stuxnet had infected a computer within the organization it began to spread in search of Field PGs, which are typical Windows computers but used to program PLCs.
- Since most of these computers are non-networked, Stuxnet would first try to spread to other computers on the LAN through a zero-day vulnerability.
- While attackers could control Stuxnet with a command-and-control server, the key computer was unlikely to have outbound internet access.
- All functionalities required to sabotage a system were directly embedded in the Stuxnet executable.
- Updates to the executable would be propagated throughout the facility through a peer-to-peer method established by Stuxnet.
- When Stuxnet finally found a suitable computer, one that ran Step 7, it would then modify the code on the PLC.

Stuxnet architecture

- The heart of Stuxnet consists of a large .dll file that contains many different exports and resources.
- Stuxnet also contains two encrypted configuration blocks.
- The dropper component of Stuxnet is a wrapper program that contains all of the above components stored inside itself in the "stub" section.
- When executed, the wrapper extracts the .dll file from the stub section, maps it into memory as a module, and calls one of the exports.

- During the execution, Stuxnet runs predefined actions.
- The original stub section is continuously passed around between different processes and functions as a parameter to the main payload.
    - A pointer to the original stub section is passed to the export as a parameter.
    - This export in turn will extract the .dll file from the stub section, which was passed as a parameter, map it into memory and call another different export from inside the mapped .dll file.
    - The pointer to the original stub section is again passed as a parameter.
- Stuxnet also uses an alternative way to call exports besides loading the .dll file into memory and directly calling the export.
- The alternative is to read an executable template from its own resources, populate the template with appropriate data, such as which .dll file to load and which export to call, and then to inject this newly populated executable into another process and execute it.
- The newly populated executable template will load the original .dll file and call whatever export the template was populated with.

DLL Exports	
Export #	**Function**
1	Infect connected removable drives, starts RPC server
2	Hooks APIs for Step 7 project file infections
4	Calls the removal routine (export 18)
5	Verifies if the threat is installed correctly
6	Verifies version information
7	Calls Export 6
9	Updates itself from infected Step 7 projects
10	Updates itself from infected Step 7 projects
14	Step 7 project file infection routine
15	Initial entry point
16	Main installation
17	Replaces Step 7 DLL
18	Uninstalls Stuxnet
19	Infects removable drives
22	Network propagation routines
24	Check Internet connection
27	RPC Server
28	Command and control routine
29	Command and control routine
31	Updates itself from infected Step 7 projects
32	Same as 1

Injection Technique

- Whenever an export is called, Stuxnet typically injects the entire DLL into another process and then just calls the particular export.
- When injecting into a trusted process, Stuxnet may keep the injected code in the trusted process or instruct the trusted process to inject the code into another currently running process.
- The trusted process consists of a set of default Windows processes and a variety of security products (Kaspersky, McAfee, Symantec,..).
- In addition, the registry is searched for indicators that the following programs are installed (KAV v6 or v9, McAfee, Trend PcCillin).

- If one of the above security product processes are detected, version information of the main image is extracted.
- Based on the version number, the target process of injection will be determined or the injection process will fail if the threat considers the security product non-bypassable.

Behaviour Blocking Bypass

- Stuxnet uses a special method to load DLLs and be able to bypass behavior-blocking and host intrusion-protection that monitor LoadLibrary calls.
- Stuxnet calls LoadLibrary with a specially crafted file name that does not exist on disk and normally causes LoadLibrary to fail.
- The filenames used have the pattern of KERNEL32.DLL.ASLR.[HEXADECIMAL] or SHELL32.DLL.ASLR.[HEXADECIMAL], where the variable [HEXADECIMAL]is a hexadecimal value.

Distribution and Target Determination

- Stuxnet is bigger and more complex than conventional worms.
- However, the attackers relied on local distribution via USB stick and local networks.
- While Stuxnet infected any Windows PC on its path, it only targeted Siemens' controllers.
- The infection from PC to controller happens via Ethernet, Profibus, or Siemens' proprietary protocol.
- It uses a complex process of fingerprinting to ensure that it is on target.
- Fingerprinting includes model number, configuration details, downloading the program code of the controller.
- If match, then Stuxnet drops rogue code to the controller.
- The rogue driver DLL contained three controller code sets: two (infection A and B) destined to a Siemens 315 controller, and one (infection C) to a 417 controller.
- The latter is more complicated and four times the size of the former.
- 417 is Siemens' top-of-the-line product.

Controller Hijacking

- The rogue code was logically structured in a set of subfunctions.
- The attackers' goal was to get any of these functions called.
- Stuxnet achieved this goal by injecting code into an executive loop.
- Both 315 and 417 attack codes used the main cycle for this purpose, the 315 also used the 100-ms times (interrupt handler).
- Stuxnet only occasionally takes over the main code and normal behavior.
- Whenever an export is called, Stuxnet typically injects the entire DLL into another process and then just calls the particular export.

- The trusted process consists of a set of default Windows processes and a variety of security products (Kaspersky KAV, Mcafee, AntiVir, BitDefender, Etrust).
- In addition, the registry is searched for indicators that the following programs are installed (KAV v6 to v9, McAfee, Trend PcCillin).
- If one is detected, version information of the main image is extracted, and decision is made on whether is possible or not to infect.

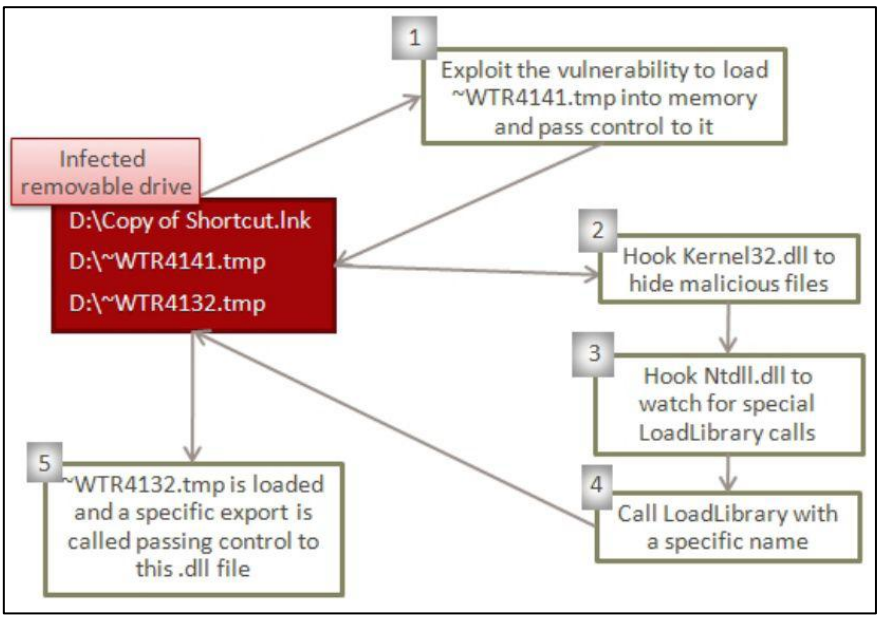
Removable Drive Propagation

- One of the main propagation methods Stuxnet uses is to copy itself to inserted removable drives.
- ICS are commonly programmed via Windows computers that are non-networked and operations usually exchange data using removable drives.
- Stuxnet used two methods to spread to and from removable drives.
    - vulnerability that allowed auto-execution when viewing the removable drive using an autorun.inf file

LNK Vulnerability (CVE-2010-2568)

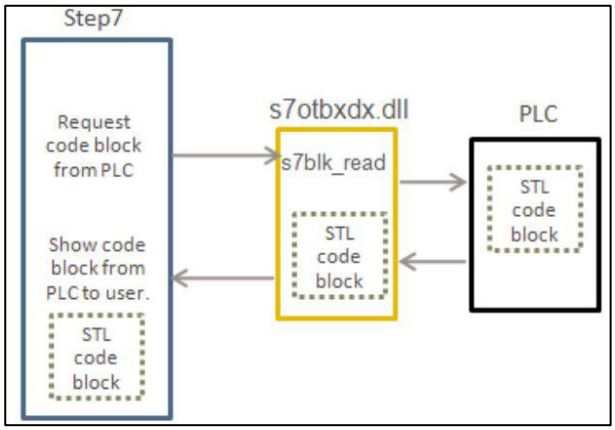
- The removable-drive copying is implemented by exports 1, 19, and 32.
- Export 19 must be called by other code and then it performs the copying routine immediately.
- Exports 1 and 32 both register routines to wait until a removable drive is inserted.
- Exports 1 and 32 both register routines to wait until a removable drive is inserted.
- If called from Export 1 or 32, Stuxnet will first verify it is running within services.exe, and determines which version of Windows it is running on.
- It creates a new hidden window with the class name ‘AFX64c313’ that
    - waits for a removable drive to be inserted (via the WM_DEVICECHANGE message)
    - verifies it contains a logical volume (has a type of DBT_DEVTYP_VOLUME)
    - is a removable drive (has a drive type of DEVICE_REMOVABLE)
- It created a list of .lnk files that contain an exploit that will automatically execute a .tmp when viewing the folder and hook kernel APIs.

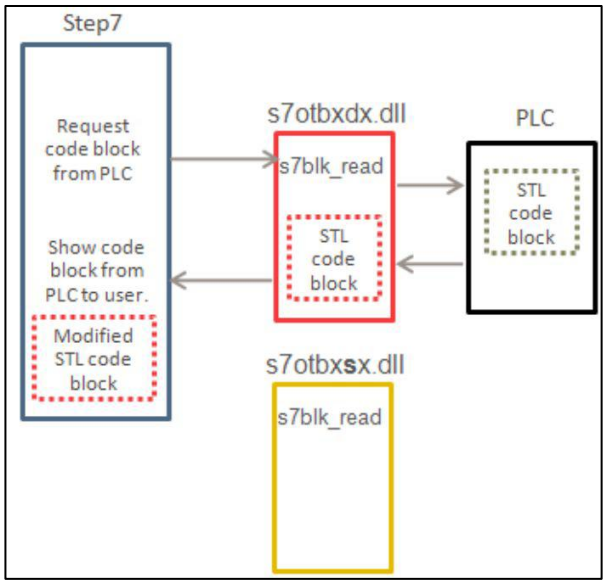
USB Execution Flow



Modifying PLCs

- The end goal of Stuxnet is to infect specific types of Simatic programmable logic controller (PLC) devices.
- PLC devices are loaded with blocks of code and data written using a variety of languages, such as STL or SCL.
- The compiled code is an assembly called MC7.
- To access a PLC, one needs a specific software (e.g., WinCC/Step 7).
- With this software installed, the programmer can connect to the PLC with a data cable and access the memory contents, reconfigure it, download a program onto it, or debug previously loaded code.
- Step 7 uses a library file called s7otbxdx.dll to perform the actual communication with the PLC (block exchange).
- Stuxnet aims at replacing the s7otbxdx.dll file responsible of handling PLC block exchange between a programming device and the PLC.
- By replacing this file, Stuxnet can
    - Monitor PLC blocks being written to and read from the PLC
    - Infect a PLC by inserting its own blocks and replacing or infecting existing blocks
    - Mask the fact that a PLC is infected





- Stuxnet renames the original s7otbxdx.dll file to s7otbxsx.dll
- It then replaces the original .dll file with its own version.
- It can now intercept any call that is made to access the PLC from any software package.
- Stuxnet's s7otbxdx.dll file contains all potential exports of the original .dll file.
- The majority of these exports are simply forwarded to the real .dll file.
- The trick is in 16 non-forwarded but intercepted exports, where intercepted exports are the routines to read, write, and enumerate code blocks on the PLC, among others.
- By intercepting these requests, Stuxnet can modify the data sent to or returned from the PLC without the operator of the PLC realizing it.
- Thanks to these routines, Stuxnet is able to hide the malicious code on the PLC.
- Types of blocks:
    - Data blocks contain program-specific data, such as numbers, structures.
    - System Data Blocks (SDB) contain information about how the PLC is configured.
    - Organization blocks are the entry point of programs cyclically executed by the CPU.

Attack Behaviour

- The attacks were triggered by complex timer and process conditions.
- Initial state: attack code just stays put and let the legitimate controller code do its thing.
- Strike time: take over control without the legitimate controller noticing.
- For the 315 attack, execution of legitimate code simply halted during the strike condition.
- 417 attack implemented a man-in-the-middle attack on the controller, intercepting the interaction between the legitimate control program and physical I/O.

- Similar to a computer application that does not directly access any physical interface registers of an I/O chip (such as Ethernet, USB, or serial), but would use data from the driver's buffer Providing the legitimate program code with a fake process image.
- The fake data that the 417 attack code fed to the legitimate controller program was recorded from real, unsuspicious inputs.
- Stuxnet replayed prerecorded input to the legitimate code during the attack.

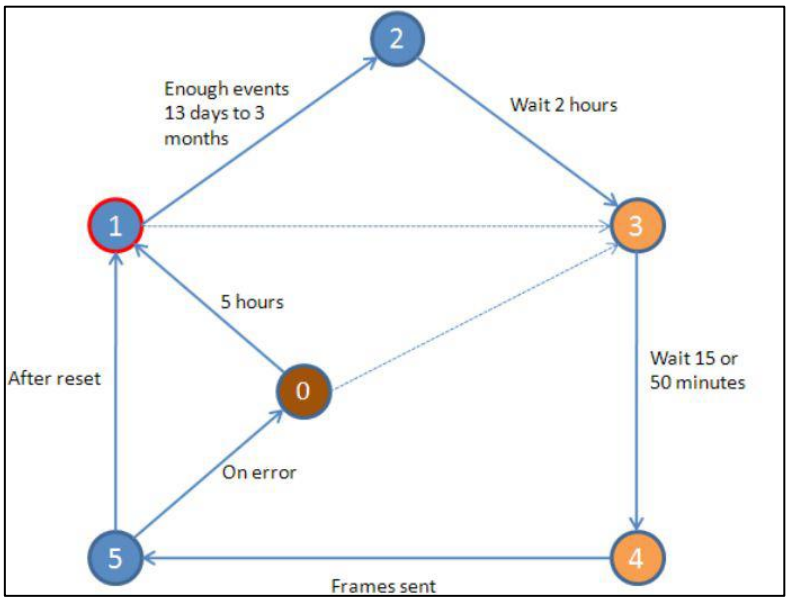
Infection Thread A and B

- This thread runs the infection routine every 15 minutes.
- Check if PLC type is 6ES7-315-2 via API call.
- The SDB blocks are checked to determine whether the PLC should be infected (check whether it is using Profibus identification numbers 7050h and 9500h, i.e., frequency converters).
- The DP_RECV block (used to receive Profibus frames) is copied to FC1869, and then replaced by a malicious block embedded in Stuxnet.
- The malicious blocks of the selected infection sequence are written to the PLC.
- Organization Block (OB) 1 is infected so that the malicious code sequence is executed at the start of a cycle.
    - Increase the size of the original block
    - Write malicious code to the beginning of the block
    - Insert the original OB1 code after the malicious code
- OB35 is also infected. It acts as a watchdog, and on certain conditions, it can stop the execution of OB1.

BEHAVIOR OF INFECTED PLC BY A/B

- DP_RECV is the name of a standard function block used by network coprocessors, used to receive Porfibus network frames.
- The infection of a 315-2 is organized as follows.
- The replaced DP_RECV block (referred to as DP_RECV monitor) is meant to monitor data sent by the frequency converter drives to the 315-2 CPU via Profibus.
- Up to 6 CP 342-5 Profibus communication modules are supported.
- The addresses of the CP 342-5 modules are recorded.
- Frames sent over Profibus are inspected. They are expected to have a specific format.
- Each frame should
    - have 31 records - one for each slave - of either 28 or 32 bytes as the format differs slightly for the two different frequency converter drives.
    - Some fields are stored.
- The other blocks implement a finite state machine

- In state 1 fields recorded by the DP_RECV monitor are examined to determine if the target system is in a particular state of operation. When enough fields match simple criteria, a transition to state 2 occurs.
- In state 2 a timer is started. Transitioning to state 3 occurs after two hours have elapsed.
- In states 3 and 4, semi-fixed network frames are generated and sent on the Profibus to DP slaves
- The content partially depends on what DP_REECV monitor has recorded
- State 5 initiates a reset of various variables used by the infection sequence before transitioning to state 1. Transitioning to state 0 may also occur in case of errors.
- In state 0, a 5-hour timer is started



- The initial state is 1 and transitioning to state 2 can take a fair amount of time.
- The code specifically monitors for records within the frames sent from the frequency converter drives that contain the current operating frequency (speed of device being controlled) → referred to as PD1.
- Frequency values can be represented in Hz or deciHz, and are in the range 807-1210 Hz.
- If value(PD1) > 1210, the code assumes is represented in deciHz and adjusts all frequency values by a factor of 10.
- The data in the frames are instructions for the frequency converter drives.
- E.g., one of the frames contains records that change the maximum frequency (the speed at which the motor will operate).
- The frequency converter drives consist of parameters, which can be remotely configured via Profibus.
- Thus, one can write new values to these parameters changing the behavior of the device.
- For sequence A, the maximum frequency is set to 1410 Hz in sequence.
- 1a, then set to 2 Hz in sequence 2a, and then set to 1064 Hz in sequence 2b.

- The normal operating frequency at this time is supposed to be between 807 Hz and 1210 Hz.
- Thus, Stuxnet sabotages the system by slowing down or speeding up the motor to different rates at different times.
- When a network send (done through the DP_SEND primitive) error occurs, up to two more attempts to resend the frame will be made.
- During states 3 and 4, the execution of the original code in OB1 and OB35 is temporarily halted by Stuxnet.
- Likely used to prevent interference from the normal mode of operation while Stuxnet sends its own frames.
- short-circuits, used to transition directly through states 0 and 1 to state 3, were planned and implemented.
- However, they appear to be disabled, which means the wait period for the transition from 1 to 2 cannot be avoided.
- When reaching states 3 and 4, the original PLC code is halted and the malicious PLC code begins sending frames of data based on the recorded values during the DP_RECV monitor phase.
- The purpose of sending the frames is to change the behavior of the frequency converter drives.
- Each record sent changes a configuration, such as the maximum frequency on the frequency converter drive.

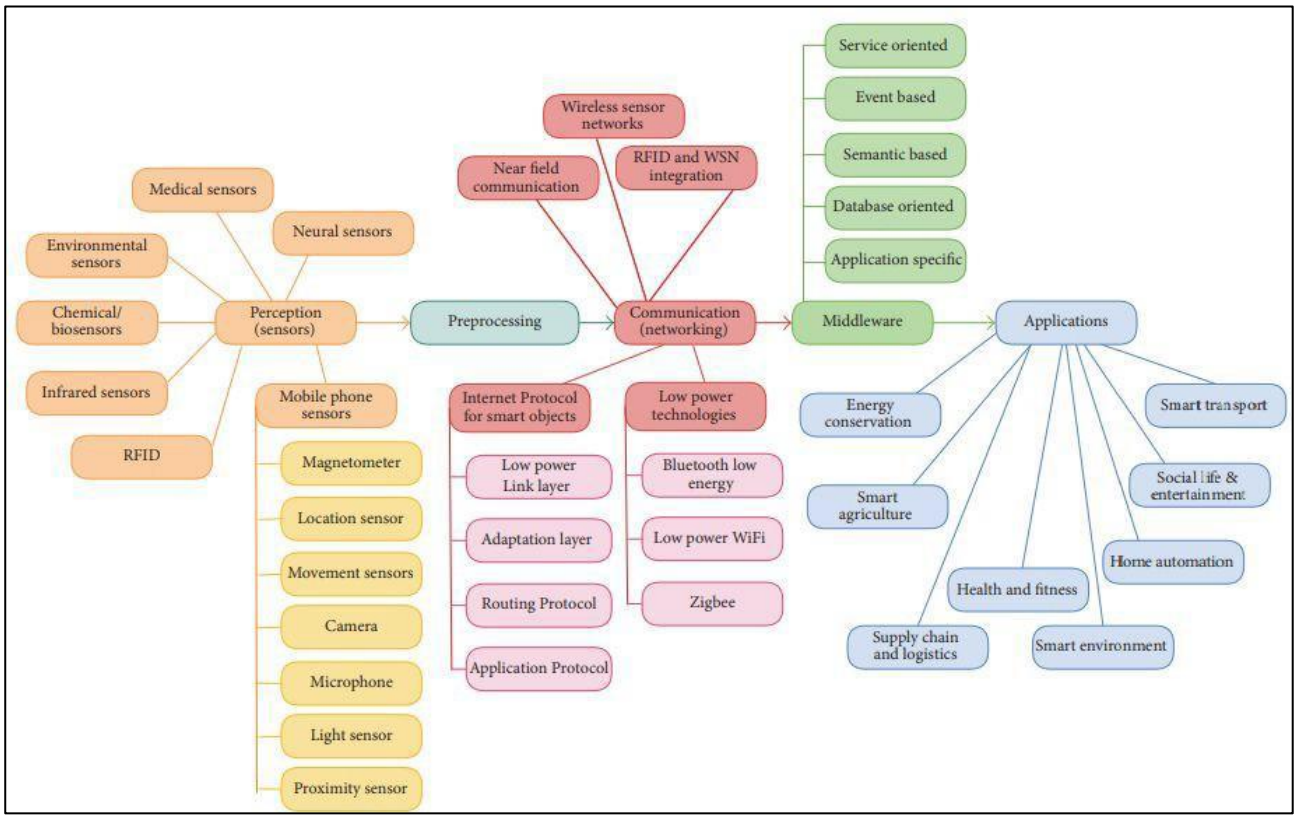
Summary

- The PLC is infected.
- Frequency converter slaves send records to their CP342-5 master, building a frame of 31 records The CPU records the CP-342-5 addresses.
- The frames are examined and the fields are recorded.
- After approximately 13 days, enough events have been recorded, showing the system has been operating between 807 Hz and 1210 Hz.
- The infected PLC generates and sends sequence 1 to its frequency converter drives, setting the frequency to 1410Hz.
- Normal operation resumes.
- After approximately 27 days, enough events have been recorded.
- The infected PLC generates and sends sequence 1 to its frequency converter drives, setting the frequency to 1410Hz.
- Normal operation resumes.
- After approximately 27 days, enough events have been recorded.
- The infected PLC generates and sends sequence 2 to its frequency converter drives, setting the frequency initially to 2Hz and then 1064Hz.

7. IoT Security and Privacy

The term “Internet of Things (IoT)” describes a group of physical devices equipped with sensing, processing, and communication capabilities able to exchange information with each other over the Internet or other communication networks. It is a result of development in different fields, including embedded devices, sensor networks, automation, and control systems.

IoT Taxonomy



An IoT system consist of three main layers:

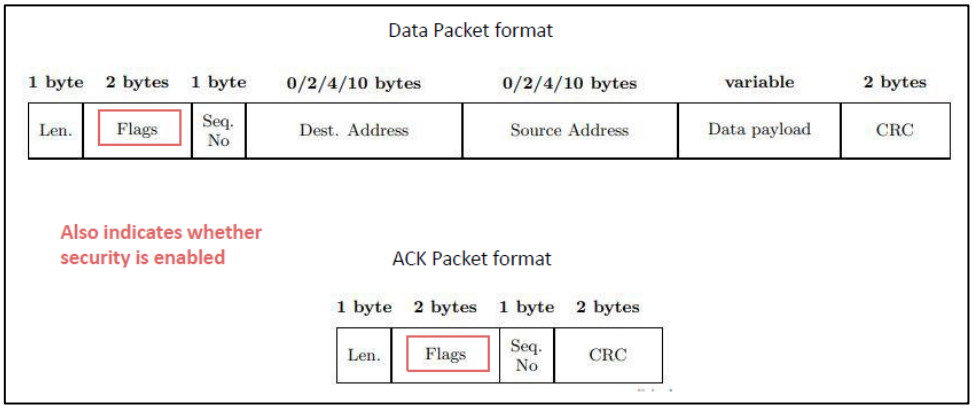
1. **Devices** are the things, i.e., those devices equipped with sensors and actuators that collect data and report it to the gateway.
2. **Gateway** is a data aggregation system to pre-process data, securing connectivity to cloud, the event hub, and sometimes fog computing.
3. **Cloud** contains the applications built using microservices, storage, event queuing, and messaging systems.

Network architecture

IoT networks are expected to include a vast number of devices, utilizing the IETF IPv6 over Low-Power Wireless Personal Area Network (**6LoWPAN**), which typically operates on IEEE 802.15.4 standards designed for low-rate PANs. For industrial applications, the IPv6 over TSCH model of IEEE 802.15.4e (**6TiSCH**) is employed. Data transport in these networks is facilitated by protocols like the IETF Constrained Application Protocol (CoAP), ZeroMQ, and MQTT.

IEEE 802.15.4

The Low Rate PAN standard defines the lower protocol layers, including PHY and MAC layers, and uses a 64-bit node ID along with a 16-bit network ID for addressing. Its basic channel access mode is carrier-sense multiple access with collision avoidance (CSMA/CA), where the channel is checked for occupancy before sending a request-to-send (RTS) packet and waiting for a clear-to-send (CTS) response. Upon receiving the CTS, the data packet is transmitted, followed by acknowledgment packets for confirmation.



IEEE 802.15.4 SECURITY

A link layer security protocol needs to provide four basic security services: access control, message integrity, message confidentiality, and replay protection.
In 802.15.4 **security is handled at the MAC layer**. The application specifies the security stack and sets the appropriate control parameters. Security is not enabled by default. An application has a choice of *security suites* that controls the type of security protections provided for the transmitted data.

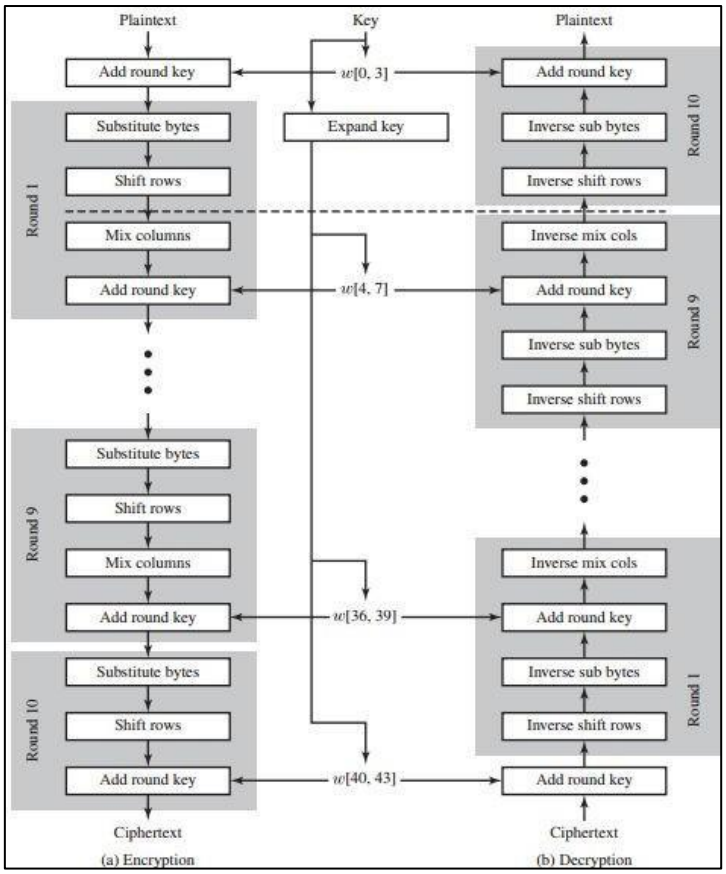
IEEE 802.15.4 standard defines different *security suites*:

- No security
- Encryption only (AES-CTR)
- Authentication only (AES-CBC-MAC)
- Encryption and authentication (AES-CCM)

Each category that supports authentication comes into three variants depending on the size of the MAC: 4, 8, or 16 bytes.

7.1 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is one of the most famous block ciphers. It is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. For AES, NIST selected three members of the Rijndael family, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits.



The input of the encryption and decryption block is a single 128-bit block. This block is copied into the state array, which is modified at each stage of encryption and decryption. After the final stage, state is copied to an output matrix. The 128-bit key is depicted as a squared matrix. The key is then expanded into an array of key schedule words, each 4 bytes long and the total key schedule is 44 words for the 128-bit key.

A key schedule process is a strategy to derive multiple keys from a single longer key.

- **Key Expansion**: the process of generating subkeys from the original key. It involves applying various operations to the original key to produce a set of round keys, which are used in multiple rounds of encryption or decryption.
- **Round Keys**: the subkeys generated by the key schedule are often called round keys because they are used in each round of the encryption or decryption process. These round keys are derived from the original key using the key schedule algorithm.
- **Round Constants**: in some key schedule algorithms, additional round constants are used to enhance the security of the key expansion process. These constants are added to the intermediate key values to introduce additional entropy and prevent attacks such as key recovery.

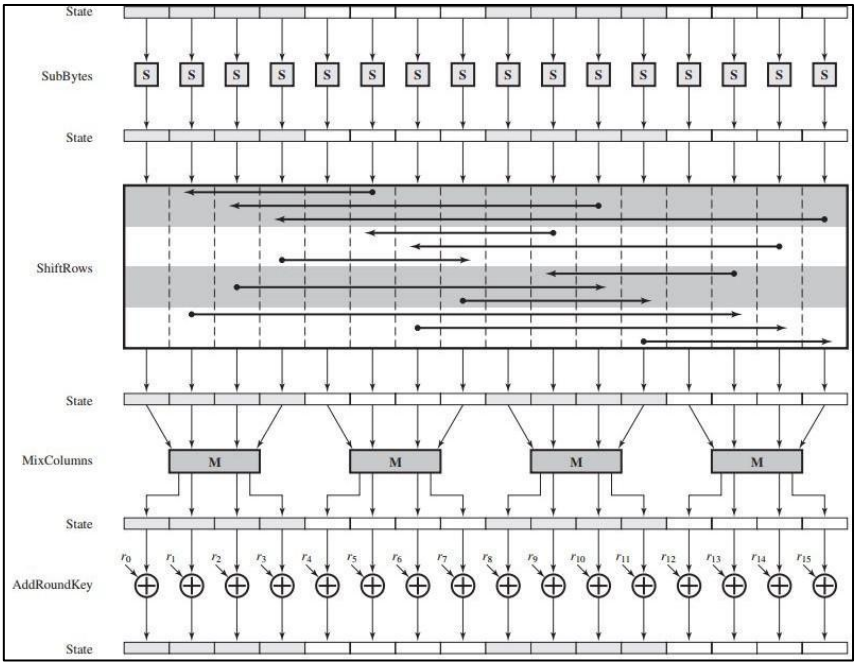
Four different stages are used, one permutation and three substitutions:

- **Substitute bytes**: uses an S-Box to perform a byte-by-byte substitution of the block.
- **Shift rows**: simple permutation performed row by row.

- **Mix columns**: substitution that alters each byte in a column as a function of all of the bytes in the column.
- **Add round key**: bitwise XOR of the current block with a portion of the expanded key.

Input	Output
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	0111
0111	1001

An S-box is essentially a lookup table that maps input bit patterns to output bit patterns according to a fixed substitution rule. Each possible input value corresponds to a unique output value.

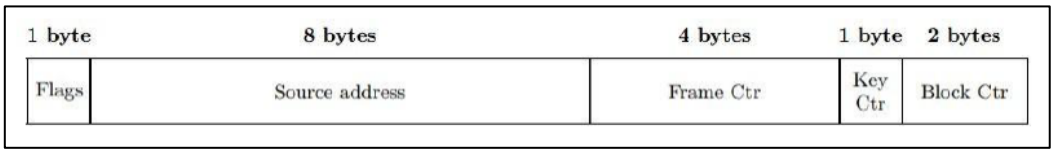


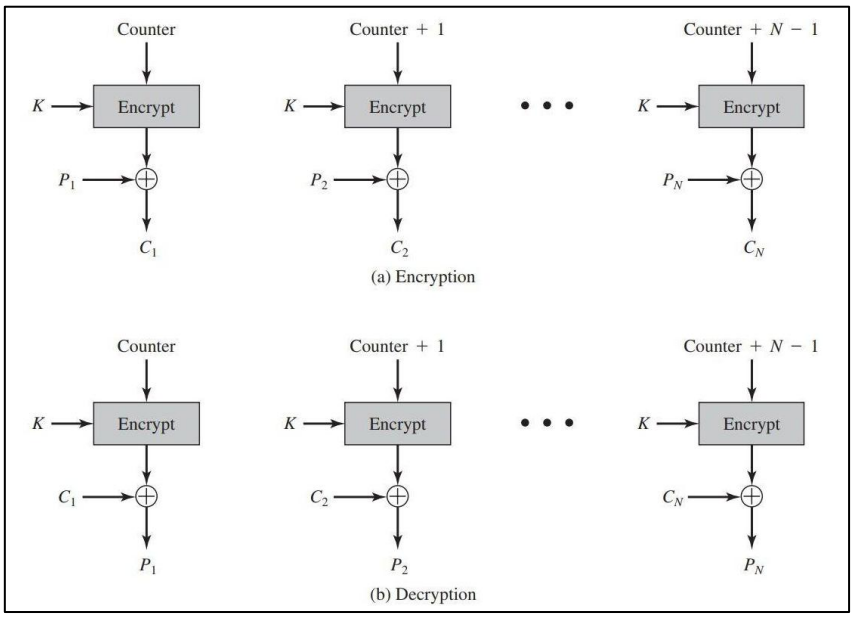
For both encryption and decryption, the cipher begins with an Add Round Key stage, followed by nine rounds that each includes all four stages. This is followed by a tenth round of three stages. Only the Add Round Key makes use of the key, hence the cipher begins and ends with an Add Round Key stage. Any other stage applied at the beginning or end is reversible without knowledge of the key.

The Add Round Key itself is not formidable. The other three stages together scramble the bits, but by themselves would provide no security because they do not use a key. Overall, the scheme is a sequence of XOR and scrambling operations. Thus, it is both highly efficient and secure. Each stage is easily reversible: for the substitute byte, shift row, and mix columns stages an inverse function is used in the decryption algorithm. For the add round key, the inverse is achieved by XORing the same round key to the block.

AES-CTR

- Confidentiality protection using AES block cypher with counter mode.
- The sender breaks the cleartext packet into 16-byte blocks $p_1, \ldots, p_n$ and computes $c_i = p_i \oplus E_k(x_i)$ where each block uses its own counter x.
- The receiver recovers the plaintext as $p_i = c_i \oplus E_k(x_i)$
- The counter, known as nonce or IV, is composed of a static flags filed, sender's address, and three separate counters.





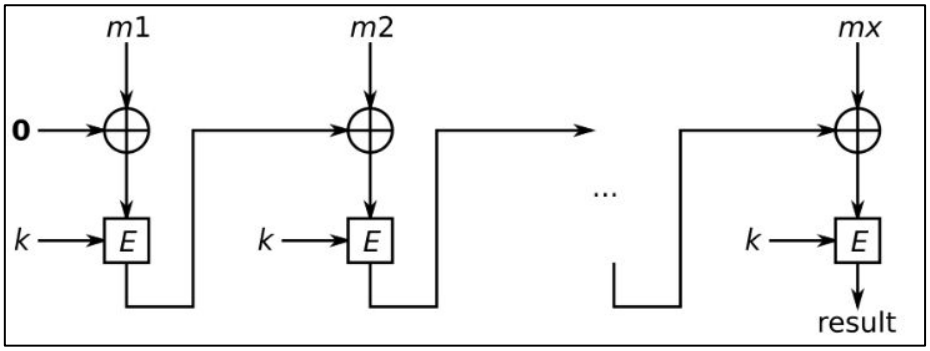
- The frame counter is maintained by the hardware radio and the sender increments it after encrypting each packet.
- The key counter is under application control.
- Requirement: nonce must never repeat within the lifetime of any single key and frame and key counter should prevent nonce reuse.
- The 2 bytes block counter ensures that each block will use a different nonce value.

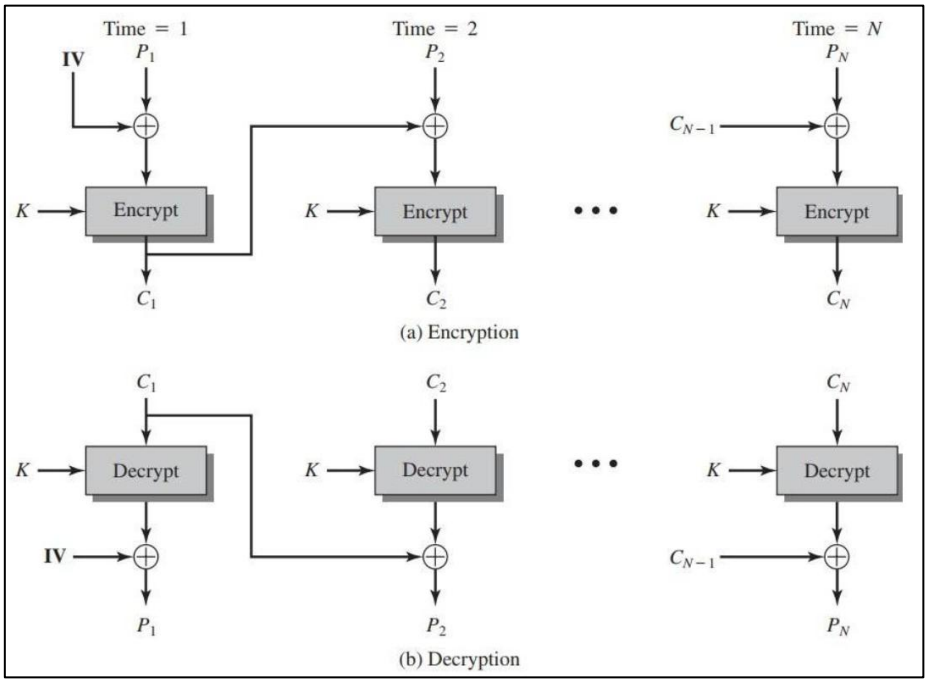
Advantages

- Hardware efficiency: it uses no chaining, so can be done in parallel over blocks.
- Software efficiency: for the same reason, processors supporting parallelism can be effectively utilized.
- Random access: possible to decrypt just a single block.

AES-CBC-MAC

- Provide integrity protection via CBC-MAC algorithm.
- It can only be computed by parts having symmetric key.
- MAC protects the packet headers and data payload.





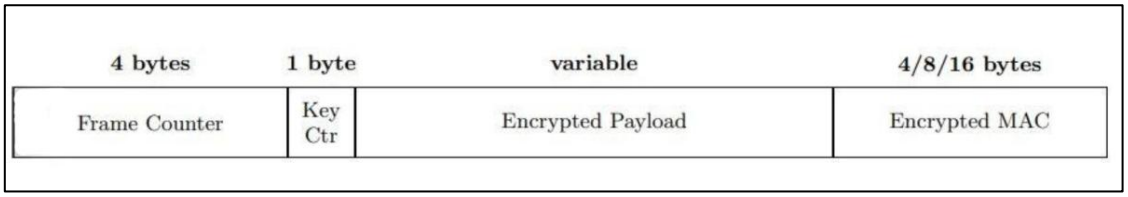
- To produce the first block of ciphertext, an IV is XORed with the first block of plaintext.
- On description, the IV is XORed with the output of the decryption algorithm to recover the first block of plaintext.
- The IV should be protected as the key.
- An adversary fooling the receiver into using a different IV may trigger selective bit flips.
- For decryption, each cipher block is passed through the decryption algorithm.
- The result is XORed with the preceding ciphertext block to produce the plaintext block.
- Given $C_j = \mathrm{E}(K, [C_{j-1} \oplus P_j])$

  We have

$$\mathrm{D}(K, C_j) = \mathrm{D}(K, \mathrm{E}(K, [C_{j-i} \oplus P_j]))$$
$$\mathrm{D}(K, C_j) = C_{j-1} \oplus P_j$$
$$C_{j-1} \oplus \mathrm{D}(K, C_j) = C_{j-1} \oplus C_{j-1} \oplus P_j = P_j$$

AES-CCM

- Provides both encryption and authentication.
- It first applies integrity protection over header and data payload using CBC-MAC.
- Encrypts data payload and MAC using AES-CTR mode.



Keying models govern what key a node uses to communicate with another node.

- **Network shared keying**: single network-wide shared key.
  - Key management becomes trivial, and memory requirements are minimal.
  - However, vulnerable to insider attacks and single key compromise.
  - A single compromised node can undermine the security guarantees of the entire network.
  - If we expect nodes to be occasionally compromised or captured, not a good approach.
- **Pairwise keying**: limit the scope of every key.
  - Each pair of nodes shares a different key.
  - Thus, if a node is compromised, only the security of communication with its pair is undermined.
  - Comes with an increased overhead.
  - Each node must store a key for every other node it communicates to.
  - Select the proper key when communicating with a node.
  - IoT nodes have limited resources.
- **Group keying**: compromise between pairwise and network keying.
  - A single key is shared among a set of nodes and is used on all links between any two nodes in that group.
  - Groups can be created based on locations, network topology, or similarity of function.
  - Partial resistance to node compromise and partial improved management of resources.

Secure Group Pairing

IoT devices need a pairing mechanism which establishes shared cryptographic keys between devices. Traditional pairing methods employ a centralized approach where a user pairs each device with a trusted IoT gateway through external helper device (e.g., type a password on the smartphone). However, the central gateway is a single point of failure, sometimes preventing devices from communicating → move towards decentralized networking (e.g., OpenThread).

Context-Based Pairing

In order to limit as much as possible, the human intervention, there is an increasing interest in context-based pairing. Co-located sensors establish shared keys based on the entropy extracted when they observe common events.
Although promising, it is limited to homogeneous devices which need to have the same sensing modalities. Furthermore, if based on events, it might take some hours or days to have pairing.

7.2 Zigbee

Zigbee is a higher layer protocol based on IEEE 802.15.4 to create PAN networks. It is usually leveraged for home automation, medical device data collection, and small-scale projects. It has a range of 10-100m in line of sight. Longer distances are achieved via multi-hop in a mesh network of intermediate devices.

Three types of devices in Zigbee networks:

- **Zigbee Coordinator (ZC)**: root of the network tree and bridge with other networks. Only one ZC, since it is the originator of the network. Trusted node containing e.g., keys.
- **Zigbee Router (ZR)**: act as intermediate device to pass data to other devices. They are usually mains powered to always be available.
- **Zigbee End Device (ZED)**: minimal functionalities to talk to the parent node. Battery powered and woke up only when has something to say.

Zigbee security builds on top of IEEE 802.15.4 security. Keys and modes discussed for 802.15.4 are basic for Zigbee. A momentary exception exists for the addition of a previously unpaired and unconfigured device. We need to assume trust in the initial installation of keys. Within the protocol stack, we need access policies to cope with the lack of cryptographic separation between different layers.

Here the security architecture implemented by Zigbee:

- Zigbee uses 128-bit keys to implement its security mechanism.
- A key can be associated to a network or to a link, acquired via pre-installation, agreement, or transport.
- There should be an initial master key obtained via a secure medium. Establishment of link keys is based on a master key.
- **Trust center**: special device in the network which other services trust for the distribution of secure keys. Ideally, all devices will have the trust center address and initial master key.
- The security architecture is distributed to different layers:

Layer	Capabilities
MAC	• Single hop reliable communications • Security level specified by upper layers
Network	• Outgoing frames use the appropriate link key according to routing
Application	• Key established and transport services to both ZDO and applications

Device authentication

After joining the network, an end-device needs to exchange security information with the trust center. Needs to obtain the current network key from the trust center and establish a new end-to-end trust center link key. Device authentication consists of four steps:

1. **Establish the Trust Center Link Key (TCLK)**: each device has a pre-installed TCLK typically obtained from the device installation code. This key is provided to the TC through out-of-band means.
2. **Establish the transport key**: the TC and node can derive a transport key from the TCLK.
3. **Distribute the network key**: the TC can send to the new node the network key encrypted via transport key.
4. **Establish new link key**: as soon as the join procedure is completed, the TC updates the TCLK of the joining device.

7.3 LoRaWAN

PANs sometimes are not sufficient for IoT purposes. Long Range (LoRa) is a proprietary radio communication technique. LoRa Wide Area Network (LoRaWAN) defines the communication protocols and system architecture to create larger networks, LPWANs (Low Power Wide Area Networks). Also in this case, we consider battery powered resource constrained devices. LoRa is a cloud-based MAC layer protocol. It manages communications between LPWAN gateways and end-node devices.

The LoRa alliance designed security measures for LoRaWAN accounting for low power consumption, low complexity, low cost, and high scalability. As part of the network join procedure, a LoRaWAN end-device establishes a mutual authentication with the LoRaWAN network. MAC and application messaging are origin authenticated, integrity and replay protected and encrypted (end-to-end encryption for application payloads).
LoRaWAN uses AES, and **each device has a unique 128-bit AES key** and a globally unique identifier (EUI-64-based DevEUI). Allocation of EUI-64 identifiers require the assignor to have an Organizationally Unique Identifier from IEEE registration authority.
LoRaWAN networks are identified by a 24-bit globally unique identifier assigned by the LoRa Alliance.

Mutual Authentication

The Over-the-air activation (or join procedure) tests whether both devices know the AppKey. The proof is obtained by computing an AES-CMAC(AppKey) on the device's join request and by the backend receiver. CMAC is a One-Key MAC that fixes security deficiencies of CBC-MAC, i.e., the fact that the latter is secure only for fixed-length messages. Nevertheless, a variation of CBC-MAC. Two keys are derived by LoRaWAN authentication:

- Providing integrity protection and encryption of the LoRaWAN MAC commands (**NwkSKey**). NwkSKey is distributed to the LoRaWAN network to prove and verify packet integrity and authenticity.

- E2E encryption of application payloads (**AppSKey**). AppSKey is distributed to the application server to encrypt/decrypt the application payload.

7.4 Network formation

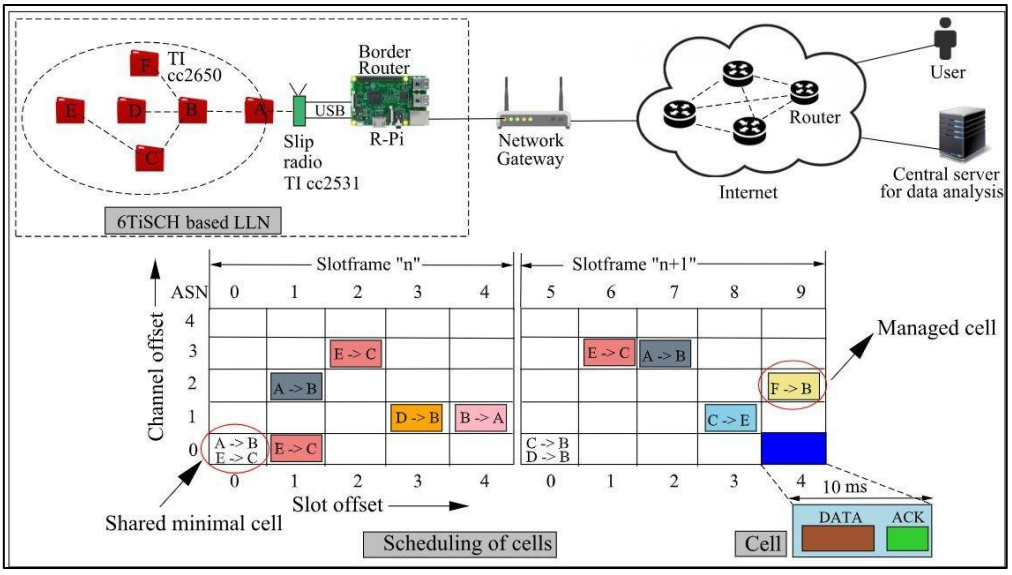
Network formation refers to the process that nodes in wireless network use to join a network. Again, focusing on IEEE 802.15.4, we focus on the ***e* version** and the **time slotted channel hopping** (**TiSCH**). 6TiSCH refers to the use of IPv6 addressing over the time slotted channel hopping medium access control methodology.

Time Slotted Channel Hopping (TiSCH)

TSCH divides time into small and fixed duration **time slots** that repeats over time. A single timeslot is long enough to transmit a packet (e.g., 10 ms for max 127 bytes long) and receive its ACK (if needed). Several timeslots collectively form a slotframe.
A node might be in a receiving, transmitting or idle state in a timeslot. Nodes use a scheduling function to decide their communication timeslot, resulting in deterministic channel access. Sometimes, a single slot may be shared by multiple users. However, in this case we use **TSCH-CSMA/CA**.
To deal with interference and multi-path fading, IEEE 802.15.4e leverages the availability of **multiple channels for communications**. The mechanism of changing the physical communication channel after each timeslot is called **channel hopping**. Nodes need to schedule one transmission channel along with the timeslot. The nodes deterministically compute their physical channel using:

$$f = F\,[(ASN + channel\ offset)\ mod\ N_{channels}]$$

where F denotes a lookup table for channels, ASN is the absolute slot number (total number of timeslots elapsed from starting of the network), channel offset is a fixed integer assigned by the scheduling algorithm, and $N_{channels}$ denotes the number of channels used in the network.



6TiSCH network formation

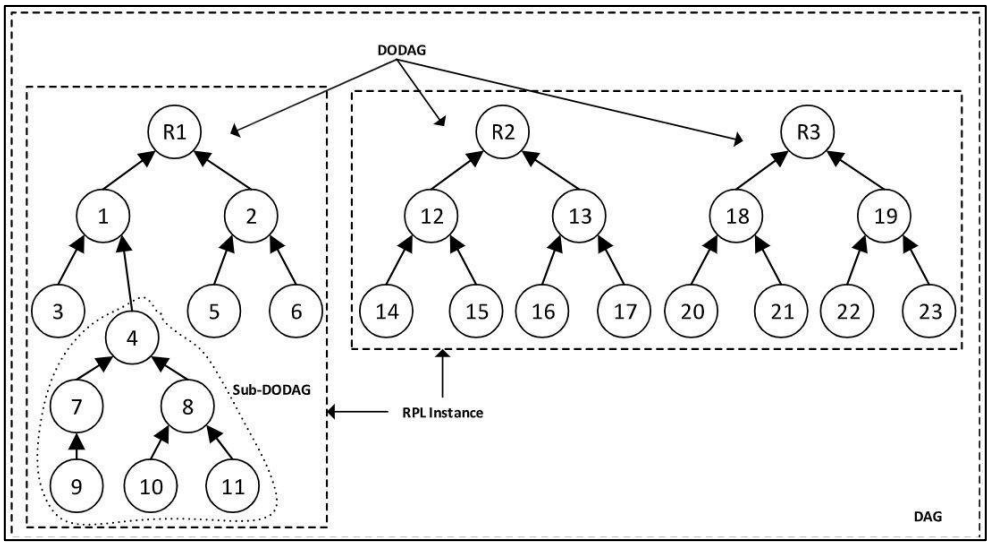
We consider the formation of a 6TiSCH network. There exists a root node, called **Joint Registrar/Coordinator** (**JRC**) which periodically broadcasts **Enhanced Beacon** (**EB**) frames. EBs contain basic network information such as the JRC ID, duration of a timeslot, number of time slots in a slot frame, channel hopping sequence, location of the shared cell. **Pledges** are new nodes willing to join a 6TiSCH network.

IPv6 Routing Protocol (RPL)

RPL was developed by the IETF Routing Over Low Power and Lossy Networks (ROLL) working group to provide routing services in low-power and lossy networks. It is a **distance vector routing protocol** that organizes network devices into **Directed Acyclic Graphs (DAGs)**. A DAG represents a network structure where all nodes are connected in such a way that there are no round-trip paths, ensuring efficient data transmission towards one or more root nodes.

Within a DAG, there can be one or more **Destination-Oriented DAGs (DODAGs)**, where each topology has a single root node. To support multiple applications simultaneously and independently within the network, several RPL instances can coexist within a DAG. Nodes in an RPL network use Objective Functions to define essential configurations such as routing metrics, optimization objectives, rank calculation, and parent selection within the DODAG.

Each node is assigned a **rank**, which is an unsigned integer value that defines its relative position to the DODAG's root node. This rank is used to determine the set of parent nodes a node can choose from. The rank value is influenced by the Objective Function and increases strictly as it moves away from the DODAG root.



Different types of RPL messages are used to maintain the network topology:

- **DODAG Information Object (DIO)**: advertised by each node when it wants to join a DODAG, create one, or maintain one. It contains information that is used to identify RPLs instances (RPLInstanceID), DODAG ID, DODAG version number, RPL mode of operation, rank of the sending node, DODAG configuration. Usually, DIO messages are multicast by nodes when they receive DIO messages from other nodes (except for the root node).

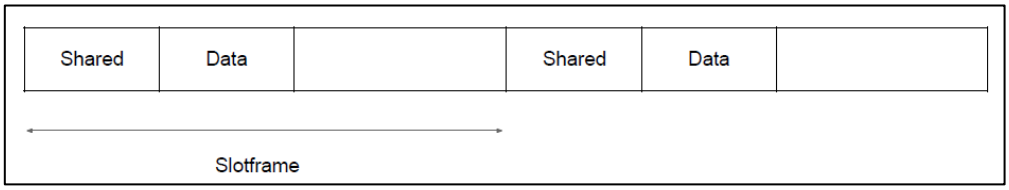
- **DODAG Information Solicitation (DIS)**: used by nodes when they want to join a DODAG and they did not receive any DIO message for a period of time (neighbour probing). The response depends on how it was sent (if unicast, or multicast). Anyway, the use of DIS messages can however introduce a vulnerability that can be exploited by an attacker as we will see later.
- **Destination Advertisement Object (DAO)**: while DIS messages are used to create and maintain upward routes (towards the root), DAO are used to find downward routes. It contains path information for reachable nodes by its sender, used to create routing tables at receiving nodes.
- **Consistency Check (CC)**: used by RPL to synchronize security counters/timestamps between each pair of nodes and provide a challenge response mechanism as a basis for control messages replay attack protection. Always sent as secure messages and hence only available in RPL secure mode.

Formation and Maintenance

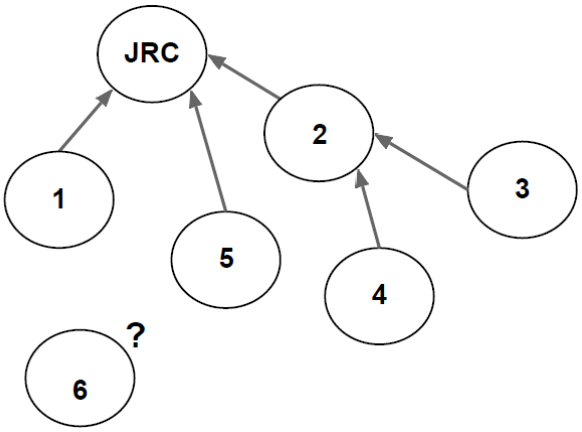
- As previously discussed, DODAG formation is done via DIO messages and always starts from the root node.
- The root node should set most of the DIO base fields: RPL instance ID; DODAG ID; DODAG version; RPL mode of operation.
- Upon receiving DIO messages, each node should:
    1) calculate its own rank;
    2) decide on which DODAG to join;
    3) select at least one preferred parent from a set;
    4) multicast is own DIO message (changing the rank).
- To maintain the topology, DIO messages should be exchanged on a regular basis and nodes can discard them if they do not result in any change to the current configuration.
- RPL uses the **trickle algorithm** to control when to send DIO messages.
    - Each node maintains a DIO counter (with a threshold) and a trickle timer.
    - DIO messages are sent when the trickle timer reaches its time or upon reception of a DIO message that causes a configuration change.
    - Whenever a DIO message is received and discarded, the DIO message counter is increased.
    - If the counter reaches a threshold value, both the counter and the trickle timer are reset and the trickle time is doubled.
    - DIO counter and trickle time will return back to their original setting when a change is made due to a received DIO message.
    - This procedure allows for a fewer broadcast of DIO messages when the network is stabilized.

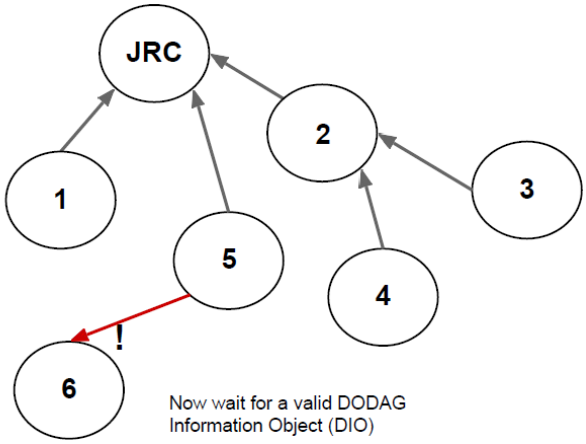
When pledges want to join the network, they start scanning until they receive a valid EB. When a new node receives an EB from an already joined node, it becomes a TSCH synchronized node.

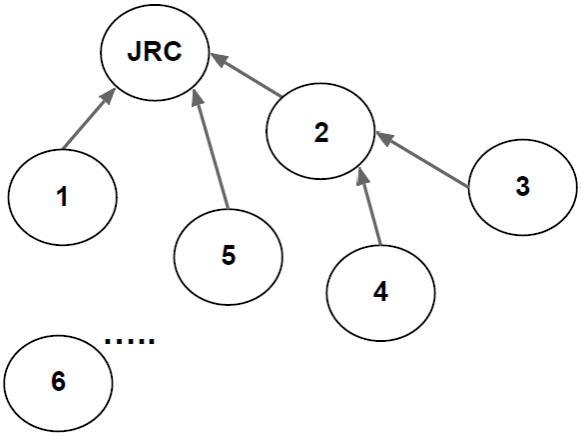
The channel is slotted and is divided into control slots and shared slots:

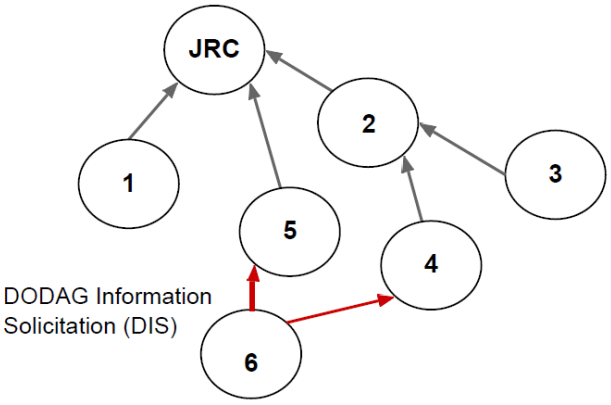


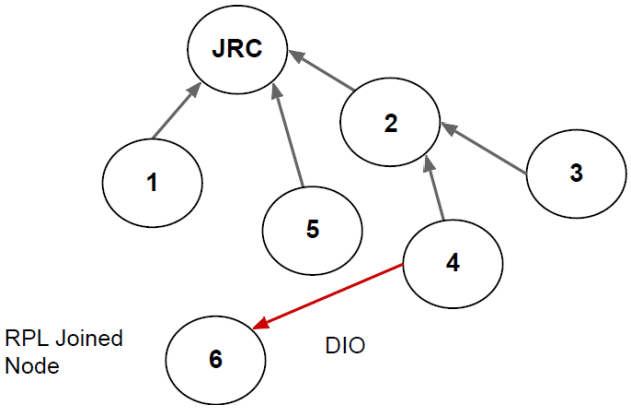
When to join?









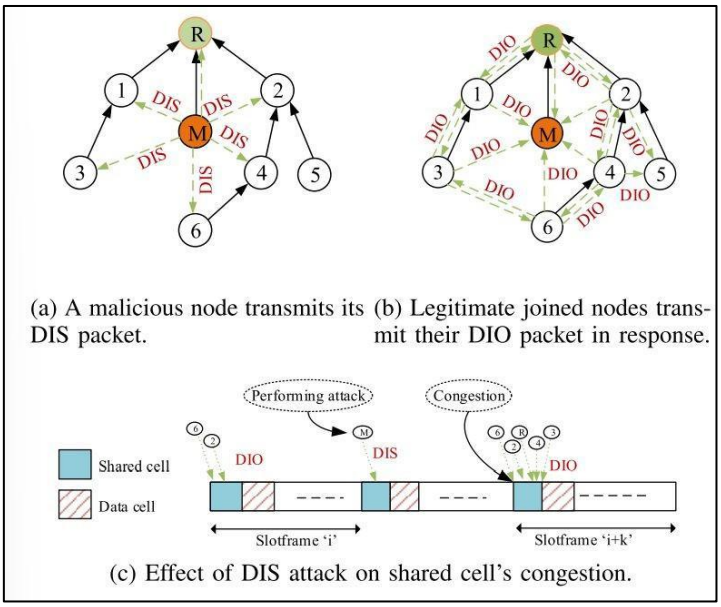


7.5 Attacks in IoT networks

DIS ATTACK

DIS packets can be sent by arbitrary nodes to solicit the sending of DIO packets. DIS attack aims at increasing the number of DIO packets transmissions in the network. The goal is therefore **increasing energy consumption and congest the shared slot**.

Note: attackers prefer DIS messages over DIO because DIS triggers multiple legitimate nodes to respond with DIOs, amplifying the attack's impact while consuming minimal resources. Unlike DIOs, DIS messages avoid validation checks as they do not require consistent topology-specific fields, making them harder to detect or reject.

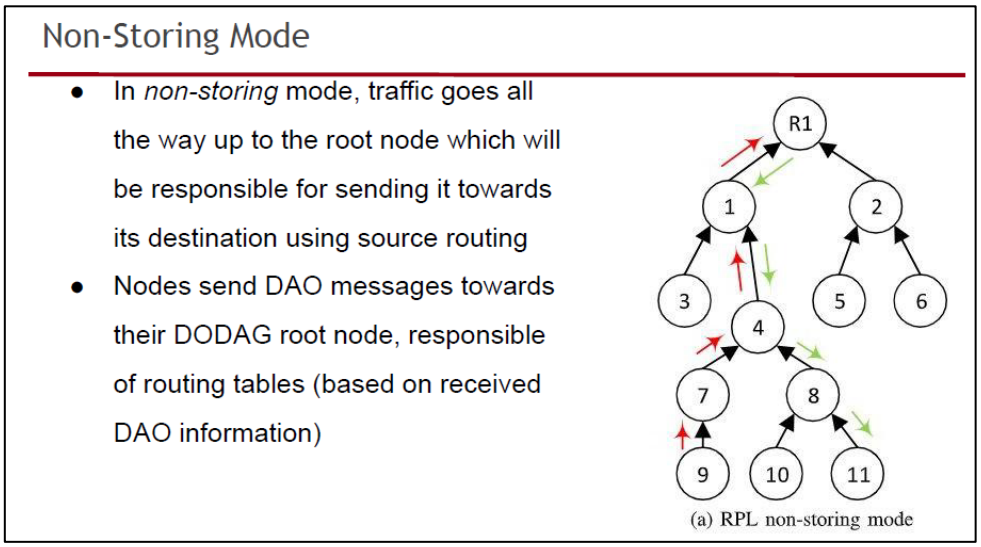


(a) A malicious node transmits its DIS packet.

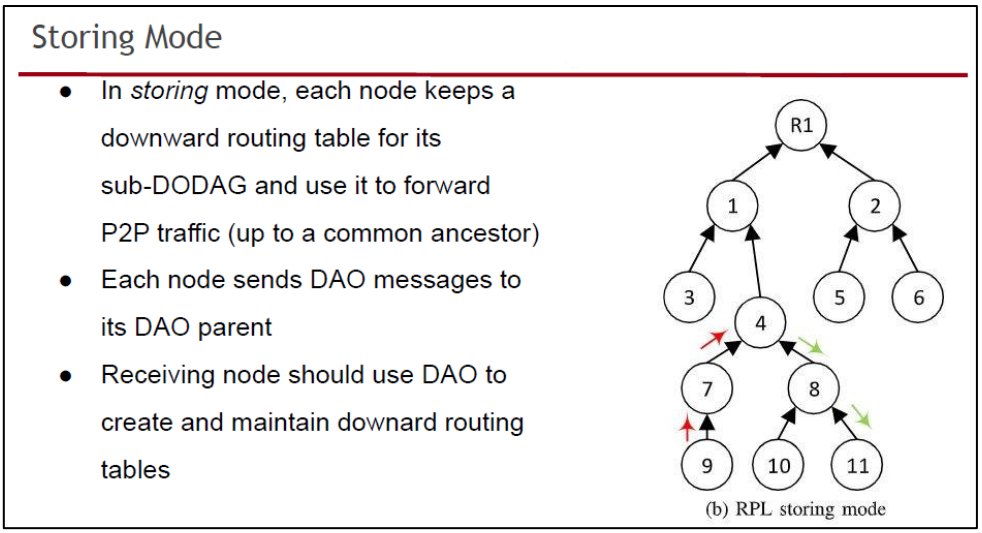
(b) Legitimate joined nodes transmit their DIO packet in response.

(c) Effect of DIS attack on shared cell's congestion.

RPL supports three types of communications in 6LoWPAN: multipoint-to-point (resembling upward traffic), point-to-multipoint (resembling downward), and point-to-point (resembling traffic between two non-root nodes). P2P has to be done using one of three ways: non-storing mode, storing mode, or one-hop P2P.



(a) RPL non-storing mode



RPL (optional) security features

Designed to use link-layer security mechanism when they are available to secure message transmission. Besides link layer, RPL has three security modes:

- **Unsecured**: default mode which depends on link layer security.
- **Preinstalled**: preinstalled symmetric keys are manually configured on the nodes which use these keys to handle and generate the secure version of RPL control messages.
- **Authenticated Mode**: nodes that want to join as leaves need to use pre-installed keys to join, but nodes with routing capabilities must use preinstalled keys to obtain another key from authentication authority.

RPL has also optional *replay protection*, called Consistency Check (CC). These checks use a non-repetitive value and the stored status information to check if the received CCM nonce value has been used before from the original node.
In the second and third mode, to support confidentiality, integrity and authenticity, nodes used AES/CCM with 128-bit keys to generate 32-bit and 64-bit MACs. Also, RPL uses RSA with SHA-256 for digital signatures of the messages to provide confidentiality and authenticity with optional 2048 and 3072-bit signatures.

RANK ATTACK

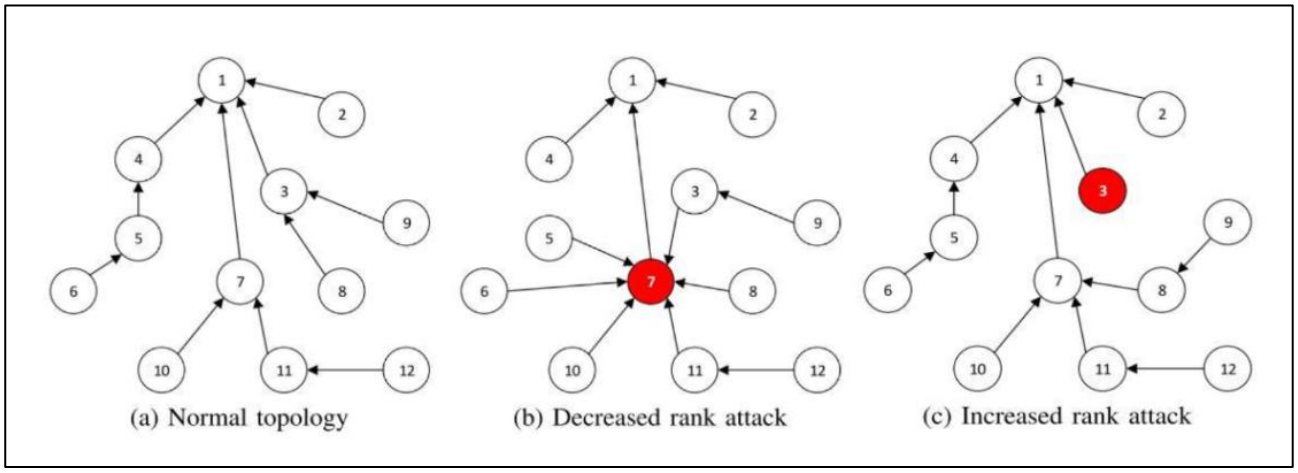
Each node chooses its parent based on two values: the rank and objective function (e.g. throughput, delay or whatever metric retained fundamental). The rank should increase going downward in the DODAG, and the role of the preferred parent selection is to select the one with the best rank. The objective of the attacker is to manipulate these values to affect the network topology. Manipulation can be performed in two ways:

- The attacker changes its rank by a specific value based on its neighbour rank value.
- The adversary manipulates its rank using a different objective function to deceive legitimate nodes into giving the malicious node a better rank (the malicious node in fact may declare “I have a better throughput” even if not true).

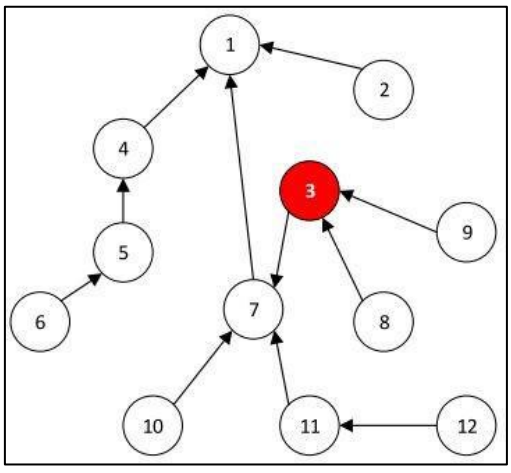
The main objective of a rank attack is to change the network topology to break security.

Types of Rank Attack

- **Decreased rank attack**: malicious nodes advertise lower rank to other nodes resulting in many of them selecting the adversary as preferred.
- **Increased rank attack**: the attacker is near the routing node and advertises higher rank and worse routing metrics. The idea is to cause topology disruptions and delays, as nodes will need to select further nodes as parents.



- **Worst parent attack**: the attacker advertises its true rank but selects the worst parent for itself. Deceive nodes into connecting to the attacker and cause delays due to the worst path they unwillingly select.



Neighboring Attack

- In this attack, the attacker node will forward any received DIO message it gets to its neighboring nodes (no modification).
- This creates the illusion that the original sender is in the range of the neighboring nodes.
- Worst case scenario: the original sender has a good rank and adversary's neighbors choose it as preferred parent although being out of range.
- Alone, the neighboring attack only causes a slight increase in the end-to-end delay.

- However, suitably combined with other attacks gets more dangerous.
- An adversary could launch a DIS attack to get DIO messages with better metrics, then selecting one of these messages to perform a neighbor attack, increasing the effect of such an attack.

RPL can work in a Point-to-Point fashion, i.e., create traffic between two nodes that are not root nodes of the DODAG. In storing mode, each node keeps a downward routing table for its sub-DODAG and use it to forwards P2P traffic. In practice, traffic goes upward up to a common ancestor of sender and destination that routes the packet to the destination node.

RPL Storing Mode Attack

- Routing table overload: the adversary sends many bogus routes (via DAO) until the node saturates.
- Route table falsification: a malicious node advertises fake routes to other nodes that might exists but not be part of the attacker's sub-DODAG causing packet losses or longer delays.
- All these attacks also cause resource exhaustion due to the increased overhead and repetitive repair attempts.

Attacks Inherited from WSN

Wireless Sensor Networks (WSNs) are the networks from which IoT was born. Therefore, IoT inherited part of the routing attacks that existed in WSNs. Although the working principle is the same, attacks needed to adapt to the new IoT paradigm.

Blackhole and Selective Forward

In a blackhole attack, a malicious node(s) will drop all packets it receives instead of forwarding them (DoS). To be less detectable, an attacker may decide to selectively drop packets (i.e., only forward RPL control messages) → selective forward or greyhole attacks.
Selective-forward attacks cannot be detected nor mitigated by the self-healing mechanisms of RPL because they pass control messages.

Sinkhole attacks

Malicious node(s) try to be sink for as much nodes as possible by advertising a fabricated link with better metrics. Sinkhole attacks by themselves are not very powerful, they need to be combined with other attacks.
These attacks can be performed by advertising DIOs with better metrics or having several adversaries directing all passing traffic toward another adversary.

Wormhole attacks

To create this attack, two adversaries need to cooperate to create a tunnel between them and transmit traffic through it instead of the regular path. Three ways to create a wormhole:

- Packet encapsulation: malicious nodes use a legitimate path between them and encapsulate packets to hide the hop count.

- Relay: deceive nodes to be neighbours.
- Out-of-band link: create links that are not part of the network.

Clone ID and Sybil attacks

In Clone ID attack, a malicious node(s) takes the identity of another legitimate node. In Sybil attacks, each malicious node takes several identities from legitimate nodes.
With sybil attacks an attacker can submit forged information to manipulate the system, disturb the routing topology and reputation-based systems. It can be mitigated by adding location information and DHTs.

To detect some of these attacks (or their declinations) there have been many proposals in terms of **Intrusion Detection Systems** (**IDSs**):

- Signature-based IDSs: use a database of signature patterns of the attacks.
- Anomaly-based IDSs: create a normal behaviour profile and compare the current observations with the normal behaviour.
- Specification-based: create a normal profile based on protocol specification.
- Hybrid IDSs: combine two of the aforementioned methods.

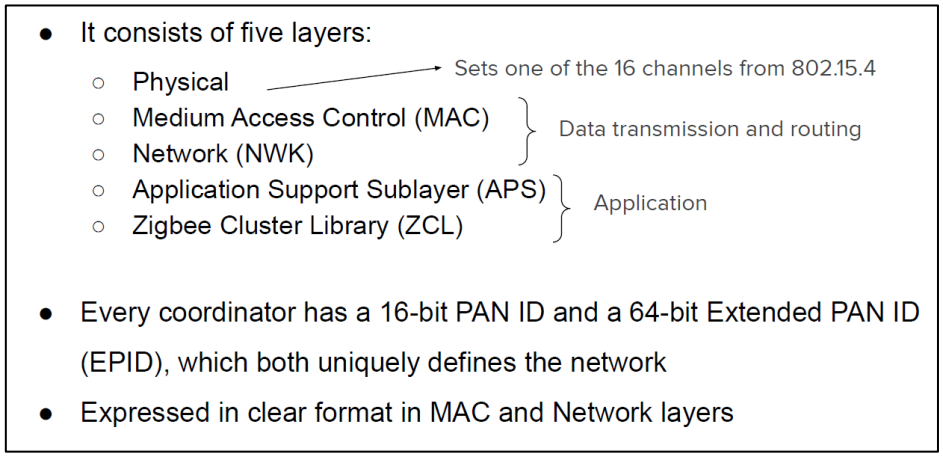
Placement of IDSs

- Centralized IDS: the IDS resides either on the root node or on a dedicated host and uses the traffic passing by to detect attacks.
    - In many cases, it is required that the central node of the IDS send periodic request for updates to unmonitored areas.
    - Advantage: most of the heavy works occurs inside a powerful node, usually capable of performing firewall functionalities as well.
    - On the other hand, challenging to monitor the network during the attack.
- Distributed IDS: each node will have a full IDS implementation, making it responsible for detecting attacks around it.
    - Usually, nodes collaborate to increase the efficiency of the detection.
    - However, this approach consumes a lot of resources throughout the network.
    - It is usually required to optimize the IDS periodically to minimize its effects.
- Hybrid IDS placement: to get the best of both centralized and distributed types.
    - Central nodes with more resources are responsible for computationally intensive tasks (analyzing data, decision making).
    - Normal nodes are responsible for lightweight duties (e.g., monitoring neighbour nodes, send info about traffic passing through them, responding to requests from central nodes).
    - Requires optimization.

7.6 IoT Authentication

Attacks on authentication systems typically exploit weaknesses in the assumption that cryptographic keys remain secret and inaccessible to adversaries. In closed networks, such as those utilizing ZigBee protocols, devices are secured by distributing keys through a secure out-of-band channel, making unauthorized access significantly more challenging. Zigbee, widely regarded as secure due to its closed nature, incorporates two key factors that bolster its resilience. First, its closed nature mandates a dedicated commissioning process to integrate new devices into the network, often requiring physical user intervention, such as pressing a button on the controller. Second, outside of this commissioning phase, the network remains isolated, with the controller rejecting any unauthorized joining requests. These measures collectively fortify the network against common attack vectors.

ZigBee Protocol Stack



Encryption and Authentication Recall

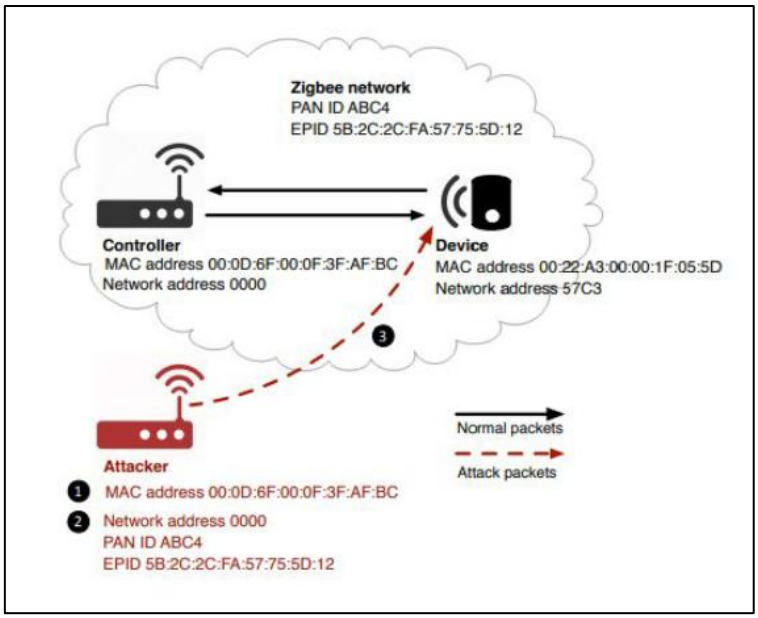
- AES encryption and CCM mode of operation (authenticity and confidentiality).
- 32-bit Message Integrity Code (MIC) is calculated and appended after the encrypted payload for integrity protection.
- Brute forcing a 32-bit MIC is infeasible in terms of time consumption → with a packet transmission rate of 200 packets/s the brute force would take more than 248 days.

Factors making ZigBee networks challenging to compromise:

- Closed network with a dedicated commissioning process to add new devices in the network (press a button on the controller).
- Zigbee uses encryption with AES-CCM → without keys you cannot infiltrate the network.

Attack to authentication: mimic the real network behaviour

- Simplified Zigbee network with two nodes: one device, one controller.
- Attacker not authorized, not part of the network.
- Attacker sniffs publicly available Zigbee network information: MAC, net addresses, PAN ID, EPID (unencrypted in MAC and network layer headers).



IMPERSONATION ATTACK

The attacker impersonates a node that is already in the target Zigbee network using the public collected information. Since the controller has the most capabilities, we focus on impersonation of the controller. The following attack steps can be launched at arbitrary time during the closed normal operations of Zigbee networks:

- **Step 1**: the attacker device needs to overwrite its manufacturer-produced physical address and pretend to be the controller.
    - This controller's address can be obtained by sniffing Zigbee packets, since the MAC address is contained in plaintext in the header.
- **Step 2**: the attacker further imitates the network identifiers.
    - Extracts the controller network address and network PAN ID by eavesdropping regular Zigbee packets.
    - To get EPID, the adversary broadcasts a beacon request.
    - The controller will send a beacon reply with EPID and state that the network is closed and does not accept join requests.
    - The adversary selects a target device and obtain its address via packet sniffing.
- **Step 3**: the attack device constructs packets and injects them into the Zigbee network.
    - The goal is to cause the target device to process forged control packets and end up in dysfunctional statuses.
    - Though Zigbee uses encryption on the network layer payload, packets crafted with specific control fields and commands can induce vulnerabilities.

Power-on Phase

- We now manipulate MAC packets during the power on phase.
- Every time a device boots up, it uses its manufacturer-provided MAC.
- We can simply change the code, and replace the one fetched from one of sources including non-volatile memory, flash, or random generation.

Network Setup Phase

- More challenging, as the adversary needs to interact with a closed network.
- During normal operations the network does not accept association requests, and authorized nodes have their roles in the network.
- A new device is not recognized and cannot play any role.
- Instead of trying to access the network, we can **impersonate the controller** and create a new network, as a twin of the original one.
- We exploit the Zigbee network formation to enforce network-setup manipulation.
- Network formation needs three steps from a controller:
    1) Enabling the radio antenna.
    2) Setting PAN ID and EPID.
    3) Adding routing path of the target device without commissioning.
- To avoid the original controller to interfere with the network formation, we assume that the setup is conducted outside of the transmission range of the target Zigbee network (after needed sniffing).

PAN ID Manipulation

- The 16-bit PAN ID is contained in every Zigbee packet to identify the associated network.
- We aim at modifying the PAN ID of the new Zigbee network that the module (attacker) creates and make it the same as the target network.
- The PAN ID is stored in a configuration file that can hence be changed upon sniffing the target one.

EPID Manipulation

- Zigbee mainly uses PAN ID in regular communications, while EPID is used in commissioning or rejoining process.
- An adversary can acquire the EPIDs of the nearby Zigbee networks by simply broadcasting a beacon request.
- During network formation, we assign the EPID to the newly created network by setting the specific value.

Routing Insertion

- Although we now have a twin network with the proper identifiers, it does not contain any node yet.
- We need to induce the framework module to believe that a target device is associated with this new network and stand-by for communications.
- At the end of the network formation, the module will be in open mode for a couple of hundred seconds to accept new nodes to join.
- However, a device has no reason to actively sending association requests.

FUZZING PACKETS

Now that we created a malicious network, we can start exploring whether it is possible to send packets that cause malfunctioning. We first need to understand what is the structure of packets we can send without legitimate keys. Then we can start exploring how to randomly generate packets to find vulnerabilities.

A first approach would be to randomly put content into the generated packets and blindly test whether they cause the Zigbee network to malfunction. However, this is highly inefficient and results in many non-valid packets.

Two challenges that we want to address in the fuzzing process:

- Zigbee uses encryption.
- Packets have varied length and formats according to header values.

Managing Encryption and Auth

- With our attack, the malicious coordinator does not know the key. Therefore, the packets it generates cannot be accepted unless being encrypted with the proper key.
- We hence examine unencrypted fields in Zigbee packets like MAC or network headers.
- We want to transmit plaintext messages that get processed.
- To have the payload of a layer encrypted, there are three security-related fields on that layer: security enabled bit, security AUX header, and Message Integrity Code (MIC).
- The security bit plays a decisive role: if set to 0, no security mechanism, so no AUX header, nor MIC → forged packets are processed at the receiver!
- Packets with security bit to 1 may have impact if the system has implementation flaws.
- We can develop strategies to fuzz this.
- If security bit is set to 1, use AES with CCM.
- The 128-bit AES-CCM allows generating encryption output with arbitrary length.
- AES encrypts an incremental nonce and then XOR with plaintext.
- The ciphertext maintains the same length as the plaintext.

Packet formats

- In addition to encryption, Zigbee has varied packet formats.
- First, the different header values cause different header lengths and consequently change the packet structure.
- Second, in the payload, commands and attribute parameters are correlated and require different lengths.
- For instance, attribute IDs have different data types to achieve various functionalities.
- We hence need to actively enumerate each individual field by sending packets and examining the format changes of captured ones.
- Two steps for Zigbee analysis:
    - Decide the header fields to find the fuzzing locations.

    - Retrieve the fuzzing ranges of commands and parameters.
- First, every layer header has a frame control field which decides the other header fields and whether the packet contains upper layers.
- We enumerate the frame control values bit by bit to find the corresponding structure changes and later construct protocol-compliant lower layers when we fuzz the upper layers.
- Second, we focus on cluster ID, command ID, and attribute ID in different upper layers for our fuzzing.
- These fields are correlated across different layers.
- For example, different command IDs require different lengths of attribute IDs, and the cluster ID in the APS header determines the ZCL layer command ID.
- We test their minimum and maximum values to get ranges.

Network Layer Fuzzing

- We first configure the network layer header and payload.
- The header can use three types of address settings (mix of network and MAC addresses) and two security bit settings (0, 1).
- Network layer packets include command ID, and each command has a one-byte attribute that can be fuzzed.
- There are 13 valid network commands.
- If security bit = 0, $13 \times 2^8 = 3328$ combinations to try.
- If security bit = 1, network payload is encrypted and MIC added for integrity check.
- Since encryption preserves the content length, we prioritize cases that could bring to meaningful results.
- All network commands are 1 byte and add another byte as attributes.
- We set random MIC values and fuzz the payload only with the length of possible commands.
- We fuzz encryption payload lengths of 8 and 16 bits → $2^8 + 2^{16} = 65792$ combinations.
- NWK header has 3 unencrypted bits that can be fuzzed for different packet settings.
- Total fuzzing number is $65792 \times 2^3 = 526336$.

Example of Attack: Key Leakage

- This attack causes vulnerable systems to leak security information.
- Some of the ZCL cluster ID fuzzed can cause key leakage in Zigbee networks.
- The attack packet will cause the device to send rejoin requests.
- During the rejoin process, the controller will resend the network key which is only encrypted by the default public link key.
- With the publicly known link key and the frame counter in the packet (plaintext), we are able to decrypt and retrieve the network key of the network.

7.6.1 Group Pairing and Key Exchange

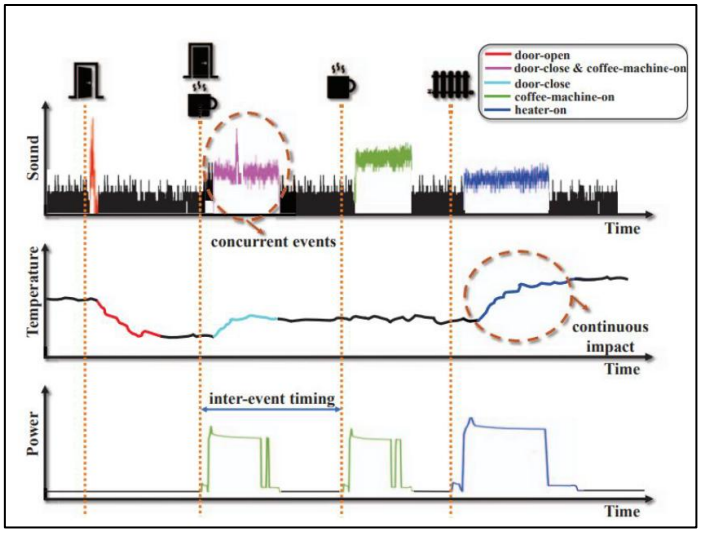
Traditional approach for key management envisions a central device with authoritative capabilities handling them all. However, this represents a single point of failure that may compromise the overall network security.
IoT platforms have recently been pushing towards decentralized IoT networking protocols (e.g., OpenThread). Past efforts at **decentralized IoT device pairing** include two approaches: human-in-the-loop, and context-based pairing.

In human-in-the-loop approaches a human need to be physically involved in the pairing process. For instance, the user should touch or press a button, shake two devices at the same time, enter password, or read QR codes.
With context-based pairing we can increase scalability, as the human is not needed. In this, co-located sensors establish shared keys based on the entropy extracted when they observe common events.

An example of events

- We consider an IoT deployment with three devices.
- Each device is equipped with a microphone, a power meter, a temperature sensor.
- User A opens the door to go out, in the meanwhile user B turns on the coffee machine.
- While the coffee machine is on, user A returns and closes the door.
- User A prepares a cup of coffee for herself and turns on the heater.

Event	Sensors Impacted
door-open/close	air pressure, humidity, illuminance, microphone, motion, temperature
coffee-machine-on/off	microphone, power
window-open/close	air pressure, humidity, illuminance, motion, temperature
oven-on/off	humidity, power, temperature
light-on/off	illuminance, power
AC-on/off	air pressure, humidity, microphone, power, temperature
heater-on/off	humidity, microphone, power, temperature
TV-on/off	illuminance, microphone, power
dryer-on/off	humidity, microphone, power, temperature

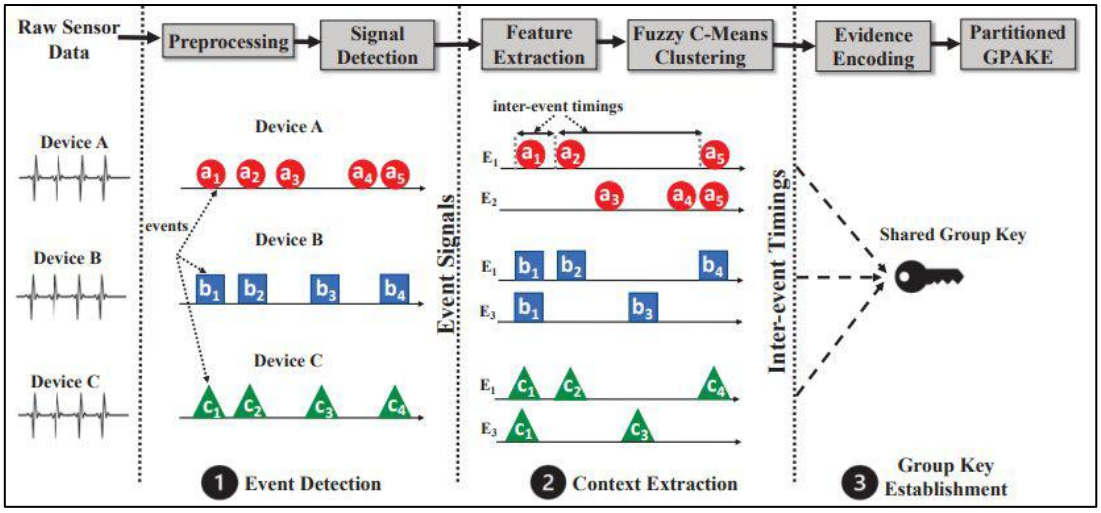


Threat Model

- We consider an attacker aiming at eavesdropping the communication between IoT devices and learn private information about users.
- Devices are deployed within an indoor closed physical space and controlled by a common trusted entity.
- The attacker is not present within the boundaries of the indoor IoT environment and cannot access, add devices or control devices inside the network.
- The attacker has complete knowledge of the pairing protocol and has access to the communication channels.

IoTCupid

- We now present IoTCupid, a solution for context-based IoT device pairing.
- The first step is to process the raw time-series data collected in real-time and perform a threshold-based signal detection to separate the sensor data corresponding to events from background noise.
- The second step is to extract distinctive time-series features from the signals each sensor has detected.
- It then extends a fuzzy clustering algorithm to group independent and concurrent signals into different events.
- The clustered events are used to obtain the sequence of time intervals between consecutive events of a given type, serving as evidence of the device's context.
- Lastly, IoT devices use their inter-event timings to authenticate each other and establish a shared group key.
- IoTCupid encodes the inter-event timings into passwords and extends a partitioned group password-based authenticated key exchange scheme for a group key establishment protocol.



- We consider devices that sense the same event as a group.
- Each subset of devices that have the same inter-event timings establishes a group key.

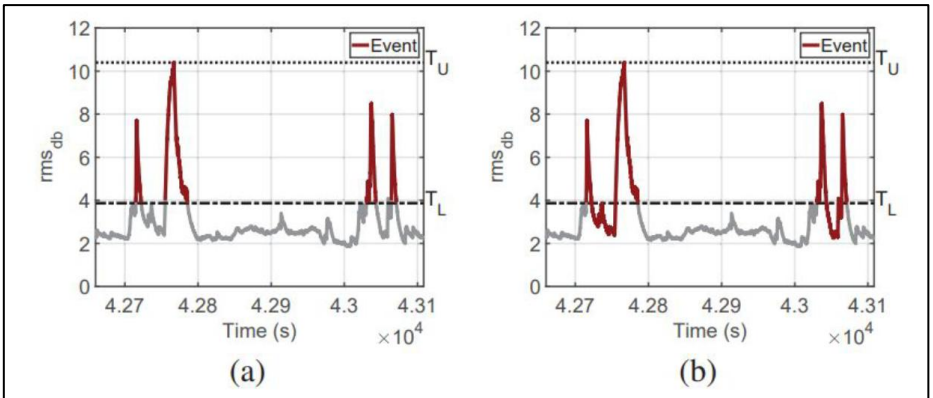
- IoTCupid does not require a central gateway or IoT hub but guarantees secure ad-hoc connectivity among heterogeneous IoT devices.
- To initiate association, devices broadcast their public keys encrypted with the extracted inter-event timing for establishing group keys.

Sensor Data Extraction

- To extract signals corresponding to events, we first segment sensor data into multiple samples with window size w_s.
- To address fluctuations during the day, we normalize the sensor readings to eliminate these fluctuations impact and capture transient changes caused by events.
- We then apply a smoothing filter by computing the exponentially weighted moving average $S_w = a*Y_w + (1-a)*S_{w-1}$, where a is the weight, Y_w is the sensor data and S_{w-1} is the EWMA of the preceding window.

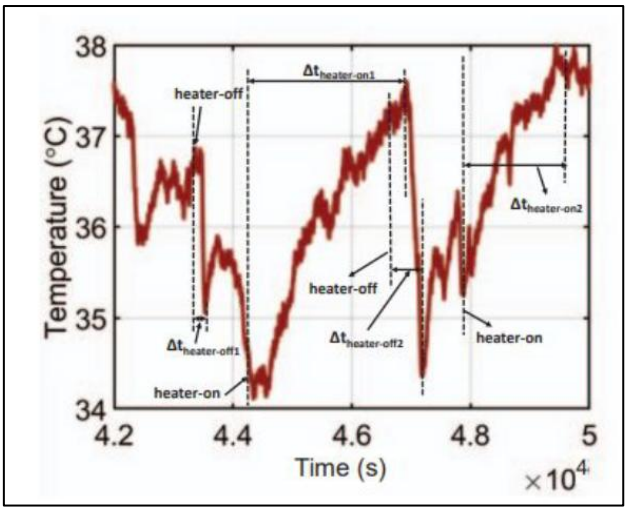
Event Detection

- We use a threshold-based approach to distinguish event's influence on sensor readings from background noise.
- We use a lower threshold T_L to identify peaks in sensor readings that distinguish events' impact from background noise.
- We use an upper threshold T_U to remove high amplitude noise signals.
- We consider the consecutive timestamps at which sensor values exceed T_L but are blow T_U as a single event.

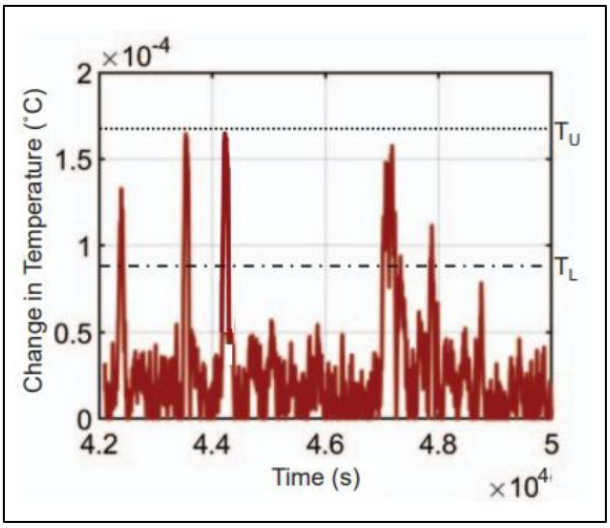


Discontinuities aggregated to avoid a single event being classified as multiple

- If sensors measure continuous quantities (e.g., temperature), the previous approach is not good.
- For instance, a heater-on event occurring at time t causes a gradual increase in temperature sensor values after a delay Δt.
- Same event at different timing may have different delays (figure below).



- To account for gradual changes and the varying delay, we leverage the rate of change in the sensor readings to detect signals corresponding to events for continuously influenced sensors.
- We first compute the derivative of the pre-processed sensor values in each window w as $S'_w = (S_{w\_ws} - S_{W0})/w_s$, where w_s is the window size and the terms in parentheses are recorded as the first and last sensor values in the window.
- We then apply lower and upper thresholds based on the average derivative of each sensor.
- We extract the timestamps where the absolute value of the sensor readings' derivatives lie within the predetermined TU and TU for the sensor.



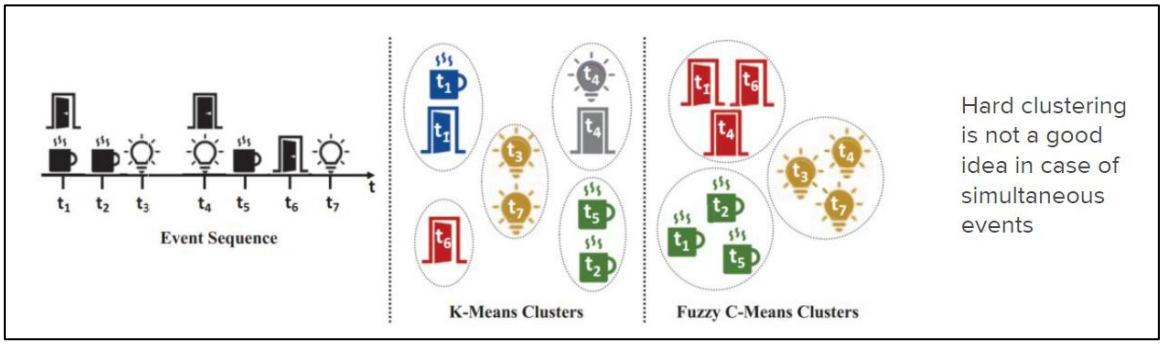
Content Extraction

- Context is about event clustering.
- The idea is to cluster the detected signals into different events to extract their inter-event timings.
- We do not leverage any prior information about the event types for clustering, as a single device might detect multiple events.

- As a first step, we extract time domain features (min, max,..) from the signals corresponding to events.
- To select a set of features F_{min} we perform dimensionality reduction via Principal Component Analysis (PCA).

Fuzzy C-Means Event Clustering

- In real deployments, multiple events may occur simultaneously and produce overlapping signals.
- This implies that the signal features of simultaneous events might significantly differ from those in single events.



- We extend fuzzy C-Means clustering to assign the detected signals into one or more appropriate event clusters based on the extracted features.
- We partition signals into c (input parameter) clusters.
- This method allow events occurring simultaneously to belong to their appropriate cluster.

Context Evidence Generation

- IoTCupid generates inter-event timings for each device after clustering the detected signals into different event types.
- Each device extract inter event times of event occurrences in each cluster, and uses it as evidence in the group key establishment.
- With this, although devices may be heterogeneous, two devices detecting the same event will measure the same inter-event timings.
- Although devices might be non-synchronized, inter-event timings are...

Challenges for Key Generation

- Recalling that the IoT deployment might be dynamic, deriving group keys might be challenging.
- First, groups must be generated dynamically based on the devices that sense the asme event.
- Second, the protocol should support device addition and removal.
- When a device is added, it should pair with other nodes.

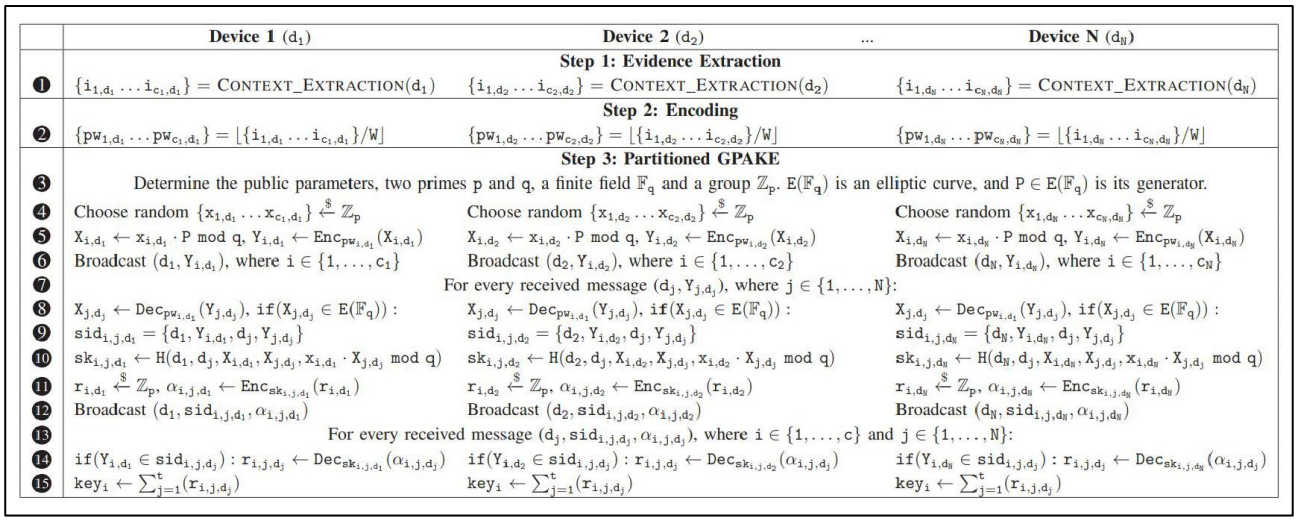
- When a device is removed its keys must be revoked to prevent an adversary from capturing them.
- Group keys can be generated using the secure communication channels from individual keys derived through a standard pairing protocol.
- Group Diffie-Hellman is one of them, but it requires multiple rounds.
- Furthermore, when a device is added to the IoT network, it must first individually pair with other devices to be authenticated and then participate in the group key establishment.
- NOT A GOOD OPTION.
- We could use fuzzy commitment schemes tp generate individual keys.
- The idea is to use error-correcting codes and enable verifying two evidences when they have small differences (Hamming distance < th.).
- However, these schemes are vulnerable to offline brute-force key guessing attacks.
- The adversary collects the network traffic and tries all evidences until they find the one that can decrypt the network traffic.
- To prevent offline brute-force key guessing attacks, we could use a large number of evidences (i.e., a large number of inter-arrival times).
- This is however highly inefficient, as it may take a long time to derive keys.
- Password-Authenticated Key Exchange (PAKE) protocols have been proposed to prevent offline brute-force attacks.
- We should however extend them into the group setting.

Group PAKE

- GPAKE enables multiple devices sharing the same evidence to derive group keys.
- The passwords of all devices that participate in the key agreement must be the same, as the scheme abort without establishing a shared key even if a single password is different.
- An adversary could leverage this limitation by joining the key agreement protocol with arbitrary evidences to deny legitimate key derivation → denial of key exchange.

New Key Establishment Protocol

- Objectives: dynamic group generation with computational efficiency, device addition/removal, resilience to offline brute-force and denial of key exchange attacks.
- We encode inter-event timings to be used as passwords.
- Devices then use the passwords to run a partitioned GPAKE scheme such that each subset of devices sensing the same events derives a group key.



	Device 1 (d_1)	Device 2 (d_2)	...	Device N (d_N)
	Step 1: Evidence Extraction			
❶	$\{i_{1,d_1} \dots i_{c_1,d_1}\} = \text{CONTEXT\_EXTRACTION}(d_1)$	$\{i_{1,d_2} \dots i_{c_2,d_2}\} = \text{CONTEXT\_EXTRACTION}(d_2)$		$\{i_{1,d_N} \dots i_{c_N,d_N}\} = \text{CONTEXT\_EXTRACTION}(d_N)$
	Step 2: Encoding			
❷	$\{pw_{1,d_1} \dots pw_{c_1,d_1}\} = \lfloor \{i_{1,d_1} \dots i_{c_1,d_1}\}/W \rfloor$	$\{pw_{1,d_2} \dots pw_{c_2,d_2}\} = \lfloor \{i_{1,d_2} \dots i_{c_2,d_2}\}/W \rfloor$		$\{pw_{1,d_N} \dots pw_{c_N,d_N}\} = \lfloor \{i_{1,d_N} \dots i_{c_N,d_N}\}/W \rfloor$
	Step 3: Partitioned GPAKE			
❸	Determine the public parameters, two primes p and q, a finite field $\mathbb{F}_q$ and a group $\mathbb{Z}_p$. $E(\mathbb{F}_q)$ is an elliptic curve, and $P \in E(\mathbb{F}_q)$ is its generator.			
❹	Choose random $\{x_{1,d_1} \dots x_{c_1,d_1}\} \xleftarrow{\$} \mathbb{Z}_p$	Choose random $\{x_{1,d_2} \dots x_{c_2,d_2}\} \xleftarrow{\$} \mathbb{Z}_p$		Choose random $\{x_{1,d_N} \dots x_{c_N,d_N}\} \xleftarrow{\$} \mathbb{Z}_p$
❺	$X_{i,d_1} \leftarrow x_{i,d_1} \cdot P \bmod q$, $Y_{i,d_1} \leftarrow \text{Enc}_{pw_{i,d_1}}(X_{i,d_1})$	$X_{i,d_2} \leftarrow x_{i,d_2} \cdot P \bmod q$, $Y_{i,d_2} \leftarrow \text{Enc}_{pw_{i,d_2}}(X_{i,d_2})$		$X_{i,d_N} \leftarrow x_{i,d_N} \cdot P \bmod q$, $Y_{i,d_N} \leftarrow \text{Enc}_{pw_{i,d_N}}(X_{i,d_N})$
❻	Broadcast (d_1, Y_{i,d_1}), where $i \in \{1, \dots, c_1\}$	Broadcast (d_2, Y_{i,d_2}), where $i \in \{1, \dots, c_2\}$		Broadcast (d_N, Y_{i,d_N}), where $i \in \{1, \dots, c_N\}$
❼	For every received message (d_j, Y_{j,d_j}), where $j \in \{1, \dots, N\}$:			
❽	$X_{j,d_j} \leftarrow \text{Dec}_{pw_{i,d_1}}(Y_{j,d_j})$, if$(X_{j,d_j} \in E(\mathbb{F}_q))$:	$X_{j,d_j} \leftarrow \text{Dec}_{pw_{i,d_1}}(Y_{j,d_j})$, if$(X_{j,d_j} \in E(\mathbb{F}_q))$:		$X_{j,d_j} \leftarrow \text{Dec}_{pw_{i,d_1}}(Y_{j,d_j})$, if$(X_{j,d_j} \in E(\mathbb{F}_q))$:
❾	$sid_{i,j,d_1} = \{d_1, Y_{i,d_1}, d_j, Y_{j,d_j}\}$	$sid_{i,j,d_2} = \{d_2, Y_{i,d_2}, d_j, Y_{j,d_j}\}$		$sid_{i,j,d_N} = \{d_N, Y_{i,d_N}, d_j, Y_{j,d_j}\}$
❿	$sk_{i,j,d_1} \leftarrow H(d_1, d_j, X_{i,d_1}, X_{j,d_j}, x_{i,d_1} \cdot X_{j,d_j} \bmod q)$	$sk_{i,j,d_2} \leftarrow H(d_2, d_j, X_{i,d_2}, X_{j,d_j}, x_{i,d_2} \cdot X_{j,d_j} \bmod q)$		$sk_{i,j,d_N} \leftarrow H(d_N, d_j, X_{i,d_N}, X_{j,d_j}, x_{i,d_N} \cdot X_{j,d_j} \bmod q)$
⓫	$r_{i,d_1} \xleftarrow{\$} \mathbb{Z}_p$, $\alpha_{i,j,d_1} \leftarrow \text{Enc}_{sk_{i,j,d_1}}(r_{i,d_1})$	$r_{i,d_2} \xleftarrow{\$} \mathbb{Z}_p$, $\alpha_{i,j,d_2} \leftarrow \text{Enc}_{sk_{i,j,d_2}}(r_{i,d_2})$		$r_{i,d_N} \xleftarrow{\$} \mathbb{Z}_p$, $\alpha_{i,j,d_N} \leftarrow \text{Enc}_{sk_{i,j,d_N}}(r_{i,d_N})$
⓬	Broadcast $(d_1, sid_{i,j,d_1}, \alpha_{i,j,d_1})$	Broadcast $(d_2, sid_{i,j,d_2}, \alpha_{i,j,d_2})$		Broadcast $(d_N, sid_{i,j,d_N}, \alpha_{i,j,d_N})$
⓭	For every received message $(d_j, sid_{i,j,d_j}, \alpha_{i,j,d_j})$, where $i \in \{1, \dots, c\}$ and $j \in \{1, \dots, N\}$:			
⓮	if$(Y_{i,d_1} \in sid_{i,j,d_j})$: $r_{i,j,d_j} \leftarrow \text{Dec}_{sk_{i,j,d_1}}(\alpha_{i,j,d_j})$	if$(Y_{i,d_2} \in sid_{i,j,d_j})$: $r_{i,j,d_j} \leftarrow \text{Dec}_{sk_{i,j,d_2}}(\alpha_{i,j,d_j})$		if$(Y_{i,d_N} \in sid_{i,j,d_j})$: $r_{i,j,d_j} \leftarrow \text{Dec}_{sk_{i,j,d_N}}(\alpha_{i,j,d_j})$
⓯	$key_i \leftarrow \sum_{j=1}^{t}(r_{i,j,d_j})$	$key_i \leftarrow \sum_{j=1}^{t}(r_{i,j,d_j})$		$key_i \leftarrow \sum_{j=1}^{t}(r_{i,j,d_j})$

(See slides for the detailed explanation)

7.7 Remote Attestation

IoT devices have their own software, and this can be compromised by malicious entities. It is not easy to detect attacks in on-field devices, as Stuxnet showed. We need solutions to **attest the legitimacy of (software and hardware) components** by distantly looking at them and their behaviour. We need to account for the constrained resources of these devices.

Remote Attestation

- Attestation is the activity of making a claim to an appraiser about the properties of a target by supplying evidence which supports that claim.
- An attester is a party performing the attestation activity.
- An attestation protocol is a cryptographic protocol involving a target an attester, an appraiser and possibly other principals serving as trust proxies.
- Target: supply evidence that will be considered authoritative by the appraiser while respecting privacy goals of its target.
- We denote as remote attestation a protocol whereby a challenge (Chal) verifies the internal state of a device called a prover (Prov). This protocol is performed remotely, i.e., over the Internet.

⇨ Goal: an honest Prov should create an authentication token that convinces Chal that the former is some well-defined expected state. If Prov has been compromised by an adversary, the authentication token must reflect this.

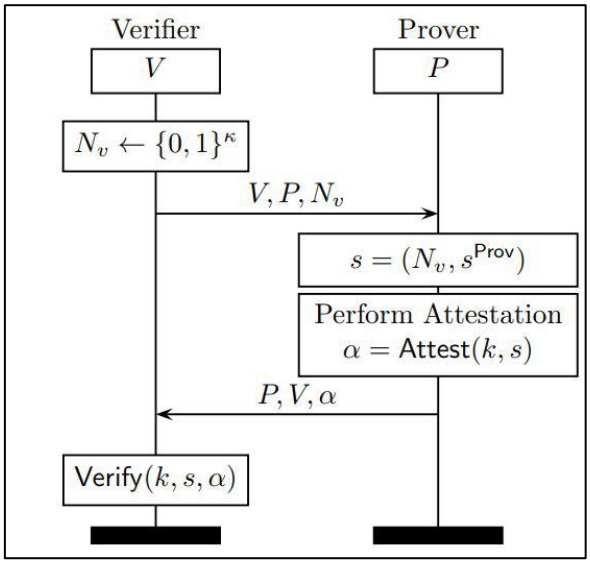
An **attestation protocol** P is comprised by the following components:

- **Steup()** - a probabilistic algorithm that, given a security parameter 1^k outputs a long-term key k;
- **Attest(k,.)** - a deterministic algorithm that, given a key k and a device state s, outputs an attestation token a.

- **Verify(k,.,.)** - a deterministic algorithm that, given a key k, a device state s, and an attestation token a, outputs 1 iff a corresponds to s, i.e., iff Attest(k,s) = a, and outputs 0 otherwise.

Attestation flow:

- The verifier challenges the prover with a fresh nonce (uniformly random and from a large pool).
- The prover attests its state with the key and creates a token.
- The verifier receives the token and decides whether to accept it.



Att-Forgery Security

- Let us define the following game: Game 1 ($\text{Att-Forgery}_{\text{Chal,Prov}}(k)$): Chall running P interacts with Prov as follows:
  - Chal runs k ← Setup(1^k) and outputs s^{Chal} to Prov
  - Prov is given oracles access to Attest, i.e., adaptively submit q device states and receive the corresponding token
  - Eventually Prov outputs a; the game outputs 1 iff Verify(k,s,a) = 1, i.e., iff a corresponds to s = (s^{Chal},s^{Prov})
- An honest node has no problem winning the previous game.
- If instead Prov has been compromised, its s^{Prov} has changed and must attempt to simulate the operation of Attest.
- We can define the following <u>security notion for an attestation protocol</u>.
  - *Definition*: Att-Forgery Security. An attestation protocol P = (Setup, Attest, Verify) is Att-Forgery-secure if there exists a negligible function negl such that for any probabilistic polynomial time prover Prov and sufficiently large k it holds Pr[$\text{Att-Forgery}_{\text{Chal Prov}}(k) = 1$] ≤ negl(k).

System model

- The central goal of attestation is to verify Prov's state.
- Successful execution however does not guarantee that the entire Prov's system can be trusted or that it cannot be compromised after attestation completion.
- We assume Prov to be a low-end embedded device with a single thread of execution, limited storage capacity, and general complexity.
- Although valid for any device, Att-Forgery-security is stronger if the cost of the device is smaller than that of a TPM.
- Prov has the following characteristics.
    - Single memory space (no separation from kernel and user memory).
    - Single thread of execution with exception of interrupts (no direct memory access).
    - Ability to disable interrupts and force a region of code to execute automatically.
    - Availability of Read Only Memory (ROM).
    - Ability to securely cleanup (erase) memory upon device reset.
    - Hardware-based control mechanism to prevent unauthorized access to certain memory location.
- We make no assumptions about Chal.
- A malicious Chal may perform a DoS by forcing Prov to take part in the RA protocol at will.
- Malicious Chal does not learn any new information about an honest Prov by performing RA, since Chal must already know the desired state of Prov in order to verify the attestation token.
- We assume that Chal is honest.
- **Note**: Challenger is the term from crypto, verifier is the term from RA. We can use them interchangeably.

Adversary Model

- We assume the adversary can compromise Prov at any time.
- Once it is compromised, the adversary has control over the prover device.
- There however needs to be a key that the adversary cannot access to although being in control of the prover.
- We assume that the adversary cannot modify the hardware components of the compromised device.
- We also assume that there is a way to protect Attest against side channel attacks.

Properties Required

- Attest needs to have specific security properties.
- We assure there exists a secure algorithm to compute a based on the prover's state s and a prover-specific key k (e.g., via HMAC).
- Attest must satisfy the following security properties

    - Only Attest can compute a valid token a.
    - a accurately captures s, i.e., Attest(k,s) = Attest(k,s') with negligible probability.
- Two ways to attacker remote attestation
    - Attack 1: The adversary simulates Attest and correctly computes a.
    - Attack 2: returned a does not reflect s, i.e., escape detection.
- The key k is the only secret held by Prov, and access to k allows the adversary to simulate Attest (i.e., type 1 attack).
- Exclusive access: attest must have exclusive access to k.
- No leaks: Attest leaks no function of k other than a, i.e., after Attest completes, the entire state of Prov is statistically independent from k.
- Immutability: Attest code is immutable. This means that it needs to be executed in-place from immutable memory.
- Uninterruptibility: the attacker has no means to interrupt the execution of attest.

Features from Properties

- We derive a set of features that are both necessary and sufficient for remote attestation that achieve the five security properties.
- **Exclusive access to k**: in our system model, the best solution is to add a small hardware-based check that monitors the address byt and the program counter and enforces that k is only accessible when PC is within attest.
- **No leaks**: we need a way to erase all intermediate values that depend on k, except the attestation token a, when they are no longer needed.
- **Immutability**: to ensure Attest to be immutable, we place it in ROM and execute it in place. We consider it as an inexpensive way to enforce immutability.
- **Uninterruptibility**: on a platform with a single thread of execution, the adversary can still regain control after invoking Attest by scheduling an interrupt. Both Attest and instructions to enable and disable interrupts should be atomic.
- **Invocation from start**: we must enforce exclusive invocation of Attest from its very first instruction. To this aim, custom hardware is needed to enforce the logic: if the program counter is an address within the Attest code, other than the first instruction address, then the previous instruction must also be within Attest.
- This prevents the adversary to jump in the middle of attest, there is no way to enforce this without OS support.
- We hence need to monitor the Attest region and reset the device if detecting illegal behaviour.

Types of Remote Attestation

- Remote attestation can be performed in several ways, with different requirements in terms of device capabilities, equipment, and security guarantees.
- At a high level, we can distinguish between software-based attestation and root of trust-based attestation.

Software-Based Attestation

- In software-based attestation, typically we use timing information to allow the verifier to assess the correctness of the firmware running on the prover.
- These approaches generally require strict timing requirements on the network, which might not always be feasible in generic IoT.
- They also generally consider a one-hop communication between verifier and prover, which makes it hard to be realized in large IoT networks.
- A Naive approach for verifying the prover's memory content would be to challenge the prover in computing a MAC of the memory content with a verifier-provided key.
- The verifier knows the memory content of the legitimate device.
- However, an attacker could easily cheat on this.
- Indeed, the attacker could save the original memory content and move it to an empty portion of the memory or to an external device that could be accessible when needing to compute the MAC-based proof.

Pseudo-Random Memory Traversal

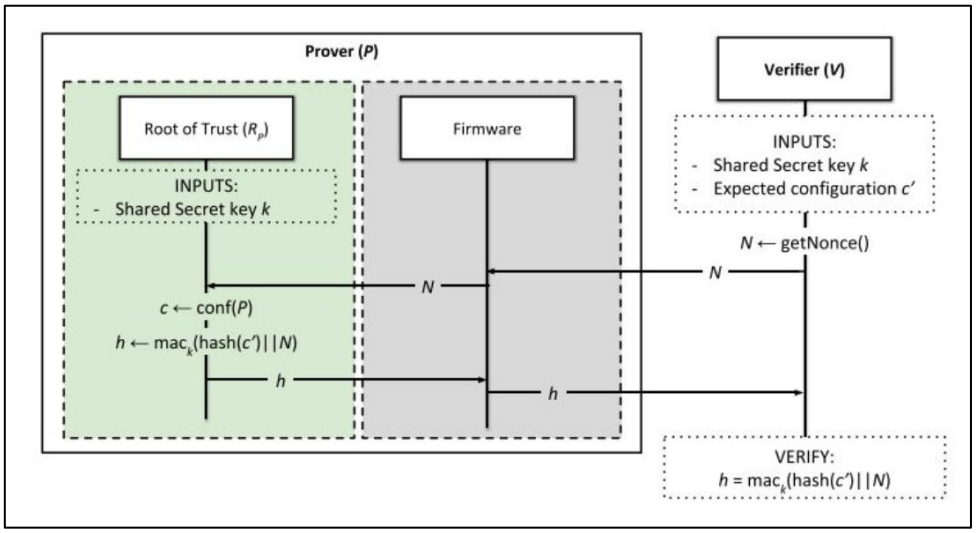
- The embedded device has a memory-content verification procedure that can be remotely activated by the verifier.
- The procedure uses a pseudorandom memory traversal.
- In particular, the challenge is used as seed for a pseudorandom number generator that generates a list of memory addresses to be checked.
- The adversary has no mean to know in advance the portion of the memory that will be checked.
- In case the verification procedure is heading towards a portion of the memory that has been altered, the attacker needs to divert it to the a memory location where the correct copy is stored.
- This however causes an increase in the time needed to compute the verification.
- Thus the verifier will either see a non valid authentication token or a suspiciously long time needed to compute the authentication token.

Attestation Based On Root of Trust

- These schemes leverage a root of trust residing inside the prover.
- This component is a assumed to be trusted and is the endpoint of the attestation protocol.
- It usually comprises a combination of hardware and software.
- The value to be attested is stored inside the root of trust.
- In IoT devices, the root of trust is realized using hardware with minimal security capabilities, such as code and memory isolation.
- Four strategies: interactive RA, Interactive Self-RA, Non-interactive RA, and non-interactive self-RA.

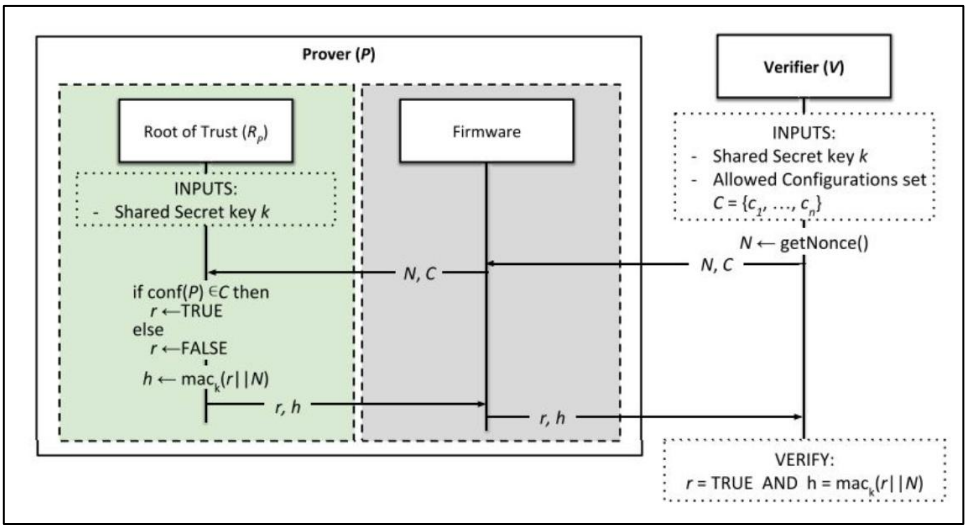
Interactive RA

- It consists of an interactive protocol between prover P and verifier V.
- V sends a challenge N to P's root of trust R_p, which responds with a proof of the device's configuration, e.g., c = hash(conf(P))||N.
- The proof h is either signal (public key crypto) or tagged via MAC (symmetric key).
- V verifies the integrity of P by verifying the authenticity of h.



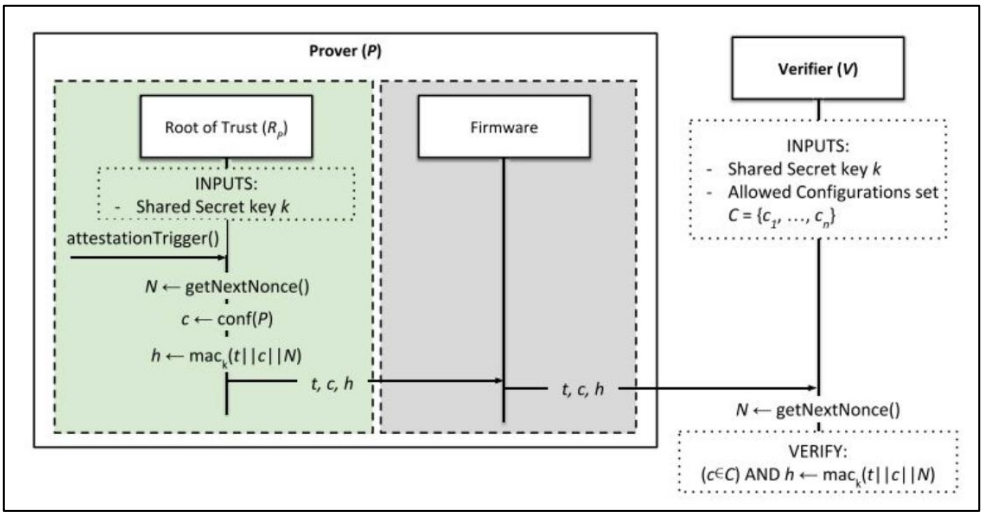
Interactive Self-RA

- Leveraging the capabilities of Trusted Execution Environments, it is possible to verify c at P's side given the list of potential allowed configurations securely stored and accessible by R_p.
- After receiving N from V anc computing c, R_p produces a signed/MACed token h authenticating a binary result r (true of false).
- This is then delivered to V as a customized token.



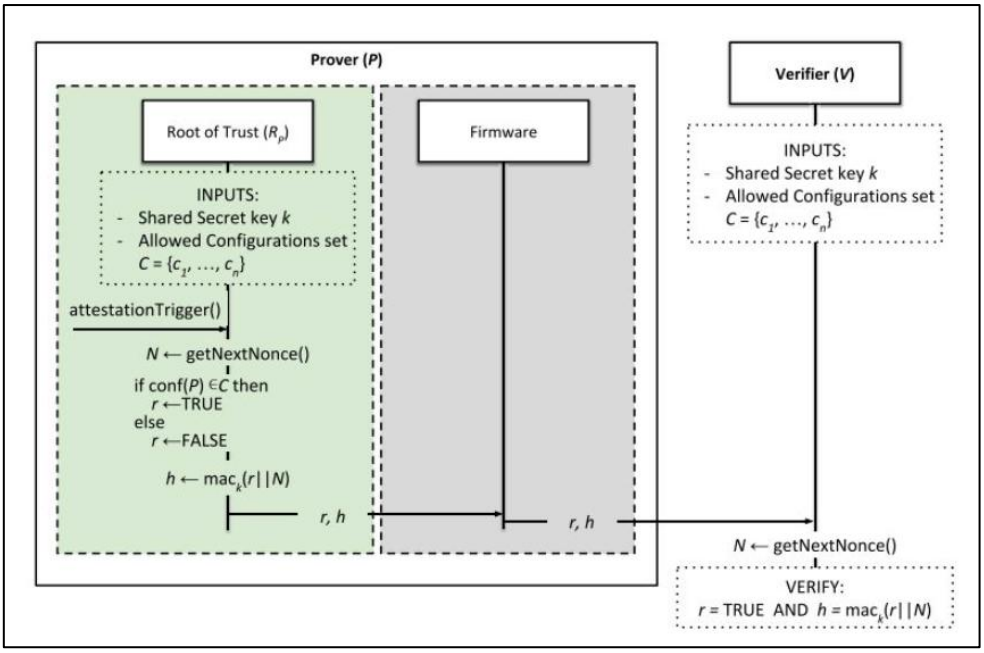
Non-Interactive RA

- The prover P autonomously decides the time at which attestation should happen and locally generates a pseudo-random nonce N.
- Remove the need for V to start the process, but require additional hardware, e.g., secure source of time.
- Examples of additional needs include Real Time Clocks, Attestation Trigger Circuit, or Reliable Read-Only Clocks.



Non-Interactive Self-RA

- Variation of the previous one, where P's TEE know the set of possible configurations and can therefore perform self attestation,



Collective RA

- In large networks, it might be convenient to assess the status of multiple nodes instead of performing single attestations.
- Collective remote attestation has been introduced to limit the time needed to perform attestation of multiple interconnected devices.
- We consider a large network of low-end devices which are heterogeneous in terms of software and hardware configurations.
- These are provers, and of course we have the verifier.
- We need an additional device, i.e., the aggregator.

Formal Verification of RA

- Verifiable Remote Attestation for Simple Embedded Devices (VRASED).
- A hybrid hardware/software solution to provide formal verification.
- Security of hardware while minimizing cost thanks to software.
- First formally verified remote attestation scheme.

Overview of VRASED

- VRASED is composed of a HW module (HW-Mod) and a SW implementation (SW-Att) of Prv's behavior according to the RA protocol.
- HW-Mod enforces access control to K (Prv's unique secret key) in addition to secure and atomic execution of SW-Att.
- SW-Att is responsible for computing an attestation report.

Formal Verification, Model Checking, Temporal Logic

- Computer-aided formal verification involves three basic steps:
    - The system of interest must be described via a formal model (e.g., finite state machine)
    - Properties that the model should satisfy must be formally specified
    - The system model must be checked against formally specified properties to guarantee that the system retains such properties
- Checking can be achieved via either theorem proving or model checking.
- In model checking, properties are specified as formulae using temporal logic and system models are represented as FSMs.
- A system is represented by a triple (S, S_0, T) where S is a finite set of states $S_0 \subseteq S$, is the set of possible initial states, and $T \subseteq S \times S$ is the transition relation set (set of states that can be reached in a single step from each state).
- Thanks to temporal logic we can represent expected system behavior over time.
- We use NuSMV.

NuSMV

- In NuSMV, properties are specified in Linear Temporal Logic, particularly useful for sequential systems.
- Use of propositional connectives such as conjunction $(\wedge)$, disjunction $(\vee)$, negation $(\neg)$, and implication $(\rightarrow)$.
- LTL includes also temporal connectives enabling sequential reasoning.
- We define a list of useful temporal connectives.
- **X**ϕ – ne**X**t ϕ: holds if ϕ is true at the next system state.
- **F**ϕ – **F**uture ϕ: holds if there exists a future state where ϕ is true.
- **G**ϕ – **G**lobally ϕ: holds if for all future states ϕ is true.
- ϕ **U** ψ – ϕ **U**ntil ψ: holds if there is a future state where ψ holds and ϕ holds for all states prior to that.
- NuSMV works by checking LTL specifications against the system FSM for all reachable states in such FSM.

Adv Capability

- We consider an adversary A that can control the entire software state, code, and data of Prv.
- A can modify any writable memory and read any memory that is not explicitly protected by access control rules. A can hence read anything that is not protected by HW-Mod.
- A can also relocate malware from one memory segment to another to avoid being detected.
- A may also have full control over direct memory access controller on Prv.

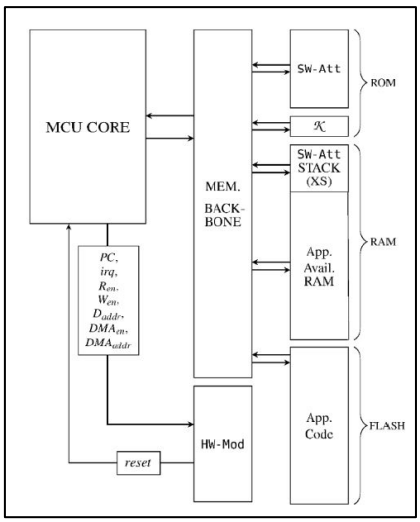
Verification Axioms

- We focus on attestation functionality of Prv.
- The verification approach relies on the following axioms.
- **A1 - Program counter:** the PC always contains the address of the instruction being executed in a given cycle.
- **A2 - Memory address:** whenever memory is read or written, a data-address signal (D_{addr}) contains the address of the corresponding memory location. R_{en} and W_{en} bits must be set for read and write access, respectively.
- **A3 - DMA**: whenever a DMA controller attempts to access main system memory, a DMA-address signal DMA_{addr} reflects the address of the memory location being accessed and a DMA_{en} bit must be set. DMA cannot access memory when DMA_{en} is off.
- **A4 - MCU reset**: at the end of a successful reset routine, all registers (including PC) are set to zero before resuming normal software execution flow. Since resets are handled by MCU hardware, no way yo modify them.
- **A5 - Interrupts**: when interrupt, the corresponding irq signal is set.
- The MSP430 design supports all the five axioms.
- Sw-Att uses HACL* HMAC-SHA256 function implemented and verified in F*.

- We assume that the standard compiler can be trusted to semantically preserve its expected behavior.
- **A6 - Callee-Saves-Register**: any register touched in a function is cleaned by default when the function returns.
- **A7 - Semantic preservation**: functional correctness of HMAC semantically preserved when converted in C.

VRASED system architecture

- Implemented by adding HW-Mod to the MCU architecture.
- Memory layout is extended to include ROM housing SW-Att code and K.
- Access control and SW-Att atomicity are enforced by HW-Mod.
- HW-Mod takes 7 input signals from the MCU core.



Notation

Notation	Description
PC	Current Program Counter value
R_{en}	Signal that indicates if the MCU is reading from memory (1-bit)
W_{en}	Signal that indicates if the MCU is writing to memory (1-bit)
D_{addr}	Address for an MCU memory access
DMA_{en}	Signal that indicates if DMA is currently enabled (1-bit)
DMA_{addr}	Memory address being accessed by DMA, if any
irq	Signal that indicates if and interrupt is occurring (1-bit)
CR	(Code ROM) Memory region where SW-Att is stored: $CR = [CR_{min}, CR_{max}]$
KR	($\mathcal{K}$ ROM) Memory region where $\mathcal{K}$ is stored: $KR = [K_{min}, K_{max}]$
XS	(eXclusive Stack) secure RAM region reserved for SW-Att computations: $XS = [XS_{min}, XS_{max}]$
MR	(MAC RAM) RAM region in which SW-Att computation result is written: $MR = [MAC_{addr}, MAC_{addr} + MAC_{size} - 1]$. The same region is also used to pass the attestation challenge as input to SW-Att
AR	(Attested Region) Memory region to be attested. Can be fixed/predefined or specified in an authenticated request from $\mathcal{V}$rf: $AR = [AR_{min}, AR_{max}]$
$reset$	A 1-bit signal that reboots the MCU when set to logic 1
A1, A2, ..., A7	Verification axioms (outlined in section 3.1)
P1, P2, ..., P7	Properties required for secure RA (outlined in section 3.2)

- We use the following notation
- AR_{min} and AR_{max}: first and last physical addresses of the memory region to be attested
- CR_{min} and CR_{max}: physical addresses of the first and last instructions of SW-Att in ROM
- K_{min} and K_{max}: first and last physical addresses of the ROM region where K is stored
- XS_{min} and XS_{max}: first and last physical addresses of the RAM region reserved for SW-Att computation

- We use the following notation
- MAC_{addr}: fixed address that stores the result of SW-Att computation (HMAC)
- MAC_{size}: size of HMAC result
- [A,B] denotes that contiguous memory region between A and B
- Therefore, $C \in [A,B] \Leftrightarrow (C \leq B \wedge C \geq A)$
- $PC \in CR$ Holds when PC is within CR_{min} and CR_{max}

FSM Representation

- Each FSM output changes in time as a function of both the current state and current input values
- Each FSM has as inputs a subset of the following signals and wires $\{PC, irq, R_{en}, W_{en}, D_{addr}, DMA_{en}, DMA_{addr}\}$
- Each FSM has only one output, reset, that indicates whether any security property was violated. Implicit representation:
  - reset is 1 whenever FSM transitions to reset state
  - reset remains 1 until a transition leaving the reset state is triggered
  - reset is 0 in all other states

RA Soundness

- RA soundness corresponds to computing an integrity ensuring function over memory at time t.
- We use an HMAC computed on memory AR with a one-time key derived from K and Chal.
- Since SW-Att is not instantaneous, RA soundness must endure that attested memory does not change during the computation of the HMAC (temporal consistency).
- In other words, the result of SW-Att call must reflect the entire state of the attested memory at the time when SW-Att is called.
- In other words, the result of SW-Att call must reflect the entire state of the attested memory at the time when SW-Att is called.

Definition 1. *End-to-end definition for soundness of* RA *computation*

$$G: \{ PC = CR_{min} \wedge AR = M \wedge MR = Chal \wedge [(\neg reset)\ U\ (PC = CR_{max})] \rightarrow$$
$$F: [PC = CR_{max} \wedge MR = HMAC(KDF(\mathcal{K}, Chal), M)] \}$$

where M is any AR value and KDF is a secure key derivation function.

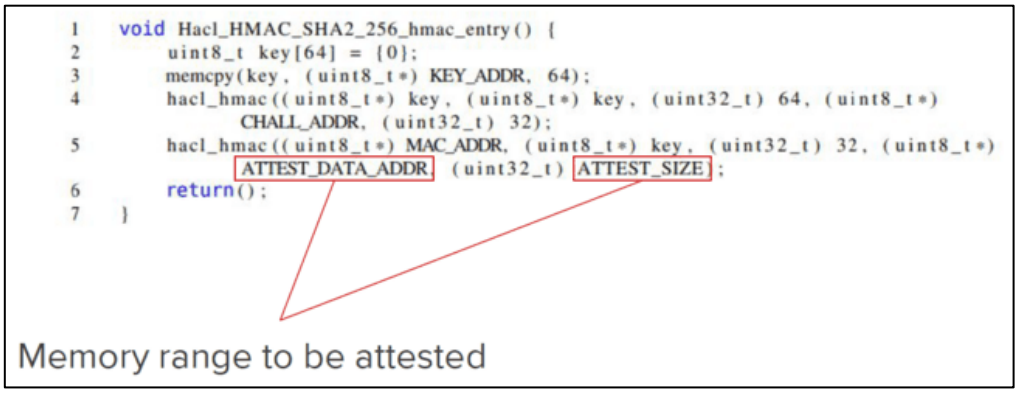
Definition 2.
2.1 RA ***Security Game*** *(*RA***-game):***
`Assumptions:`
- `SW-Att` *is immutable, and* $\mathcal{K}$ *is not known to* $\mathcal{A}$
- l *is the security parameter and* $|\mathcal{K}| = |Chal| = |MR| = l$
- $AR(t)$ *denotes the content in AR at time* t
- $\mathcal{A}$ *can modify AR and MR at will; however, it loses its ability to modify them while* `SW-Att` *is running*

RA-`game:`
1. `Setup:` $\mathcal{A}$ *is given oracle access to* `SW-Att`.
2. `Challenge:` *A random challenge* $Chal \leftarrow \$\{0,1\}^l$ *is generated and given to* $\mathcal{A}$. $\mathcal{A}$ *continues to have oracle access to* `SW-Att`.
3. `Response:` *Eventually,* $\mathcal{A}$ *responds with a pair* (M, σ), *where* σ *is either forged by* $\mathcal{A}$, *or the result of calling* `SW-Att` *at some arbitrary time* t.
4. $\mathcal{A}$ *wins if and only if* $\sigma = HMAC(KDF(\mathcal{K}, Chal), M)$ *and* $M \neq AR(t)$.

2.2 RA ***Security Definition:***
An RA *protocol is considered secure if there is no ppt* $\mathcal{A}$, *polynomial in* l, *capable of winning the game defined in 2.1 with* $Pr[\mathcal{A}, \text{RA-}game] > negl(l)$

VRASED SW-Att

- To minimize required hardware feature, hybrid RA approaches implement integrity ensuring functions (e.g., HMAC) in software.
- Derive a new unique context-specific key (key) from the master key K via HMAC-based key derivation function on Chal.
- Call HACL*'s HMAC using key as the HMAC key.
- Sw-Att resides in ROM. guaranteeing software immutability.
- Moreover, HW-Mod enforces that no other software running on Prv can access memory allocated by SW-Att.



Definition 3. *SW-Att functional correctness*

$$\mathbf{G}: \{ PC = CR_{min} \wedge MR = Chal \wedge [(\neg reset \wedge \neg irq \wedge CR = \texttt{SW-Att} \wedge KR = \mathcal{K} \wedge AR = M)\ \mathbf{U}\ PC = CR_{max}]$$
$$\rightarrow \mathbf{F}: [PC = CR_{max} \wedge MR = HMAC(KDF(\mathcal{K}, Chal), M)] \}$$

where M is any arbitrary value for AR.

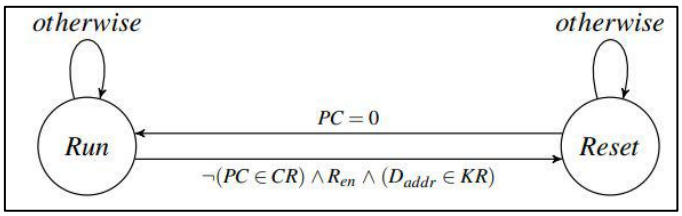
Natural language: if the memory counter is always preserved until the PC gets to CM_{max}, finally MR will contain a valid response

Key Access Control (HW-Mod)

- If malware manages to read K from ROM, it can reply to Vrf with a forged result.
- MW-Mod access control (AC) sub-module enforces that K can only be accessed bt SW-Att.

$$\mathbf{G}: \{\neg(PC \in CR) \wedge R_{en} \wedge (D_{addr} \in KR) \rightarrow reset \}$$

- The system must transition to the Reset state whenever code from outside CR tries to read from D_{addr} within the key space.
- We design a FSM from the LTL specification with two states: run and reset.
- Outputs sìreset =1 when the AC submodule transitions to reset, implying a hard reset on the MCU.



- We define two LTL specifications for atomic execution and controlled invocation.

$$\mathbf{G}: \{[\neg reset \wedge (PC \in CR) \wedge \neg(\mathbf{X}(PC) \in CR)] \rightarrow [PC = CR_{max} \vee \mathbf{X}(reset)] \}$$

$$\mathbf{G}: \{[\neg reset \wedge \neg(PC \in CR) \wedge (\mathbf{X}(PC) \in CR)] \rightarrow [\mathbf{X}(PC) = CR_{min} \vee \mathbf{X}(reset)] \}$$

$$\mathbf{G}: \{irq \wedge (PC \in CR) \rightarrow reset \}$$

- Enforce that the only way for SW-Att execution to terminate is through its last instruction PC=CR_{max}.
- Specified by checking current and next PC values using ne**X**t operator.
- If current PC value is within SW-Att region and next PC value is out of SW-Att region, then either current PC values is the address of the last instruction in SW-Att(CR_{max}), or reset is triggered in the next cycle.

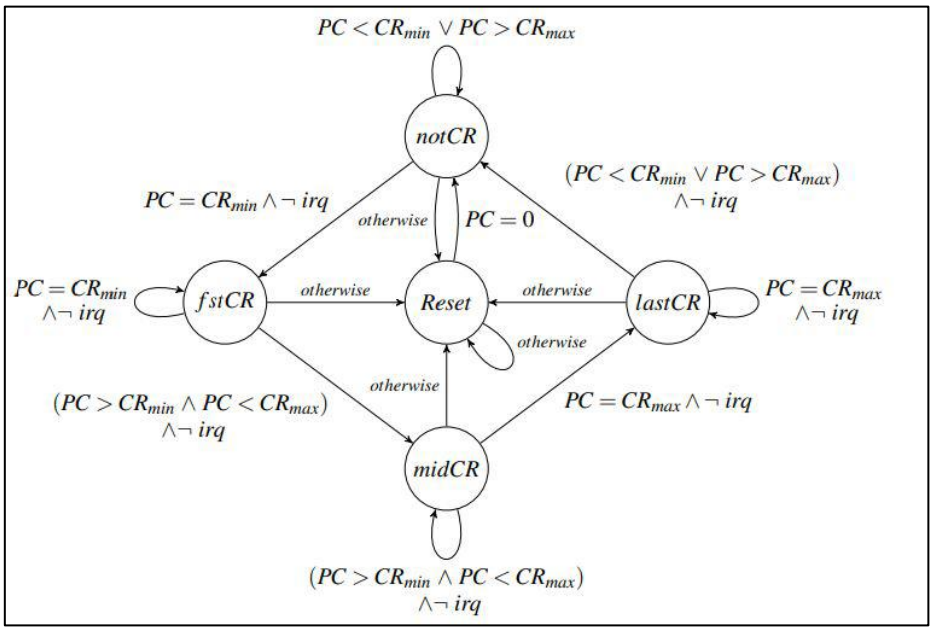
$$\mathbf{G}: \{[\neg reset \wedge (PC \in CR) \wedge \neg(\mathbf{X}(PC) \in CR)] \rightarrow [PC = CR_{max} \vee \mathbf{X}(reset)] \}$$

- Enforce that the only way for PC to enter SW-Att region is through the very first instruction CR_{min}.
- This and the previous invariant (LTL specification) imply that it is impossible to jump into the middle of SW-Att or leave SW-Att before reaching the last instruction.
- Atomicy is verified by the following LTL specification.
- Although atomicity could be violated by interrupts, this LTL specification prevents an interrupt to happen while SW-Att is executing.
- Therefore, if interrupts are not disabled by software running on Prv before calling SW-Att, any interrupt that could violate SW-Att atomicity will necessarily cause an MCU reset.

$$\mathbf{G}: \{irq \wedge (PC \in CR) \rightarrow reset\}$$

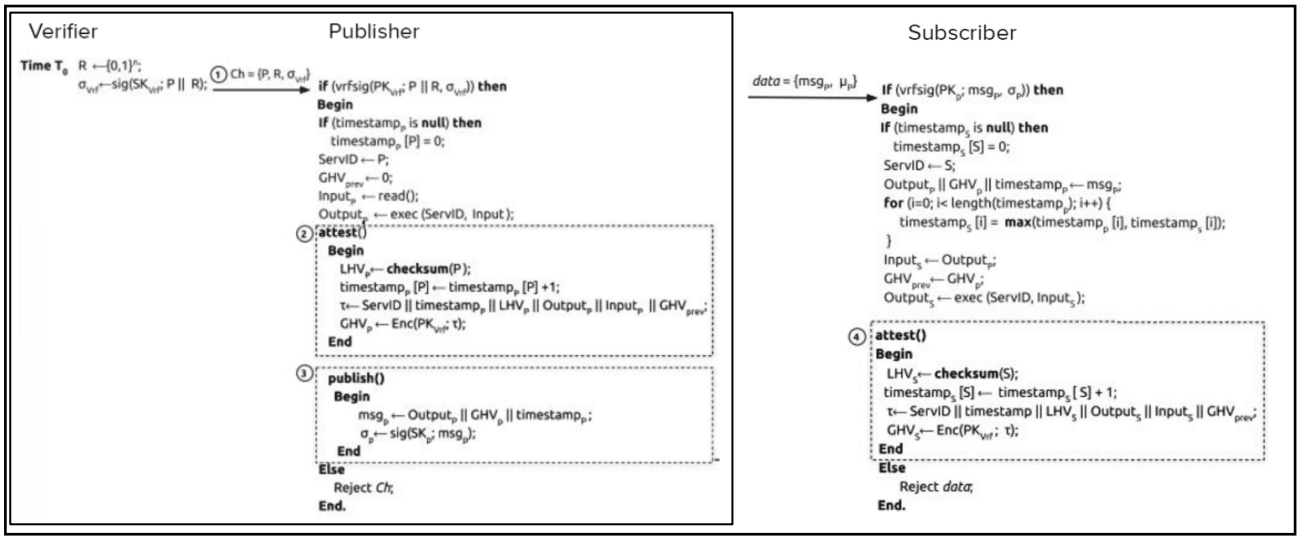
Verified Model

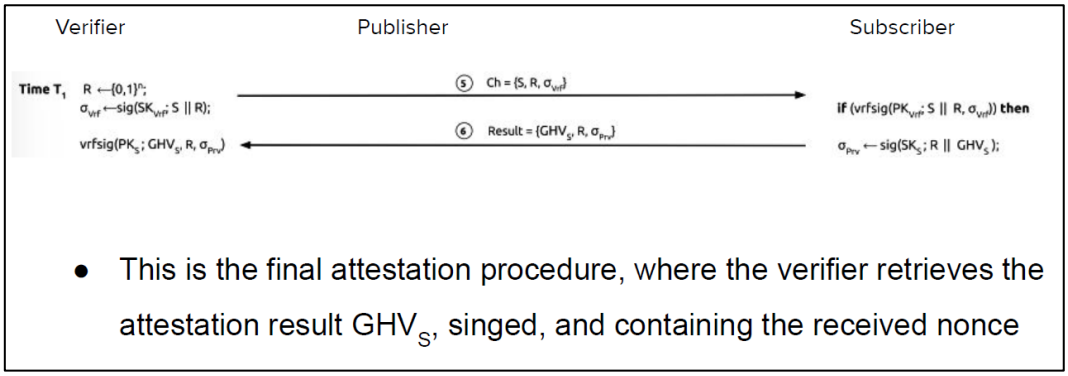
- The FSM has 5 states
  - Two basic states notCR and midCR represent movements when PC points to an address 1)outside CR, and 2) within CR, respectively
  - Two fstCR and lstCR represent states when the PC points to the first and last instructions of SW-Att, respectively
  - A reset state
- Transition to reset state whenever 1) any sequence of values for PC does not obey the aforementioned conditions, 2) irq is logical 1 while executing SW-Att.



An Example of Collective RA

- Let us consider an IoT distributed system where nodes communicate in an asynchronous manner via a publish/subscribe pattern.
- The verifier performs attestation in two steps: initialization at time T0 and attestation at time T1.
- During initialization time, the verifier initiates the attestation procedure with one or more services (publishers).
- The publisher then performs the local attestation and publishes the result together with the data it produced.
- Every subscriber service receives the publish the data and also perform the attestation
- At attestation time, verifier sends an attestation request to one or more subscribers, which act as prover for the whole network.
- Subscribers report an attestation result that includes the result of all the previous services that were directly or indirectly involved in triggering a given event to which the subscriber was registered.





- This is the final attestation procedure, where the verifier retrieves the attestation result GHV_S, singed, and containing the received nonce