



# DISCLAIMER

Il materiale contenuto nel drive è stato raccolto e richiesto tramite autorizzazione ai ragazzi frequentanti il corso di studi di Informatica dell'Università degli Studi di Salerno. Gli appunti e gli esercizi nascono da un uso e consumo degli autori che li hanno creati e risistemati per tanto non ci assumiamo la responsabilità di eventuali mancanze o difetti all'interno del materiale pubblicato.

Il materiale sarà modificato aggiungendo il logo dell'associazione, in tal caso questo possa recare problemi ad alcuni autori di materiale pubblicato, tale persona può contattarci in privato ed eliminaremo o modificheremo il materiale in base alle sue preferenze.

Ringraziamo eventuali segnalazioni di errori così da poter modificare e fornire il miglior materiale possibile a supporto degli studenti.



**CoScienze**  
Associazione

2024-2025

# IoT Security

Prof. Christian Carmine Esposito



Appunti fatti da:

- Adinolfi Francesco Pasquale,
- Arrovino Renato,
- Cerciello Vincenzo,
- Garofalo Francesco,
- Vitale Mirko

(ragazzi del curriculum di Sicurezza)

SPERIAMO CHE VI  
SIANO DI AIUTO!  
VI AUGURIAMO  
TANTA BUONA  
FORTUNA CHE VE NE  
SERVIRÀ TANTA

## 1.1. INTRODUZIONE ALLA INTERNET DELLE COSE E IL SUO HARDWARE

### INTRODUZIONE AL CONCETTO DI WSN E IOT:

Una **rete di sensori distribuiti** (Wireless Sensor Network - WSN) contiene una serie di piccoli dispositivi di rilevamento intercomunicanti e indipendenti che vengono utilizzati per monitorare le condizioni ambientali e fisiche e per comunicare tra loro grazie ad una rete adeguata.

Le informazioni dell'ambiente sono inviate ai nodi gateway, passando per una stazione base, per attivare un allarme o interfacciarsi con altri sistemi.

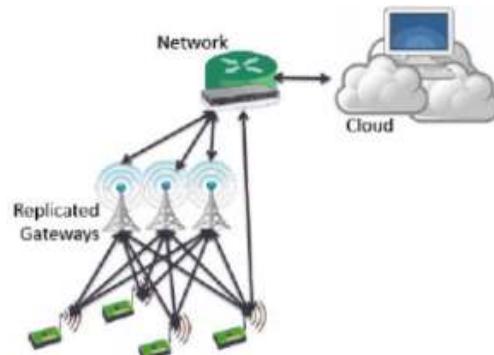
Quindi, possiamo definire l'IoT come una rete di oggetti fisici, che si connettono e scambiano dati tramite Internet.

### INTRODUZIONE AL CONCETTO DI WSN E IOT:

L'architettura di WSN è una rete di piccole/medie dimensioni.

Questa architettura presenta alcuni limiti quando aumenta il numero di nodi e il volume dei dati prodotti:

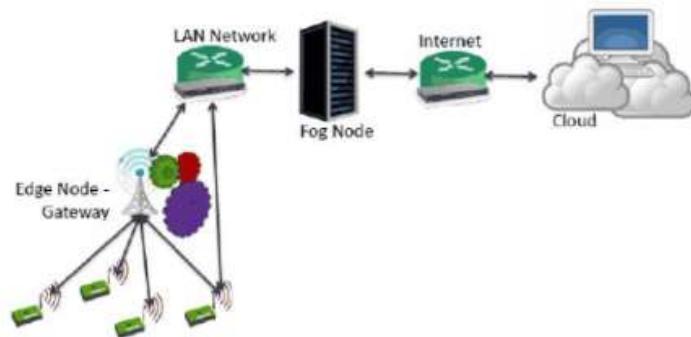
- Il **gateway** rappresenta un collo di bottiglia delle prestazioni, un singolo punto di errore e un punto di vulnerabilità per l'intero sistema; la soluzione è la **replica passiva o attiva**;
- Il **carico di lavoro** potrebbe sovraccaricare la Base Station: la soluzione è introdurre il Cloud; in questo modo abbiamo un **modello pay-per-use** con una risorsa di calcolo bilanciata;
- La **proliferazione delle modalità di accesso a Internet** implica che alcuni nodi possano avere una connettività diretta bypassando i gateway.



Quindi, l'attuale visione dell'Internet of Things è definita come WSN + Cloud:

Questa architettura presenta alcuni limiti quando aumenta il numero di nodi e il volume dei dati prodotti:

- un primo livello di elaborazione e archiviazione dei dati viene effettuato **ai margini dell'infrastruttura**;
- un livello intermedio di elaborazione/archiviazione viene effettuato presso il **fog node** (un dispositivo fisico che fornisce aiuto nella distribuzione del fog computing: un'architettura di calcolo in cui una serie di nodi ricevono dati da dispositivi IoT in tempo reale);
- il flusso di dati verso il cloud.



### IOT ABILITATO 5G:

La nuova generazione di infrastrutture di telecomunicazione, guidata dal 5G, sta abilitando lo sviluppo dell'IoT fondato su architetture Edge-Fog.

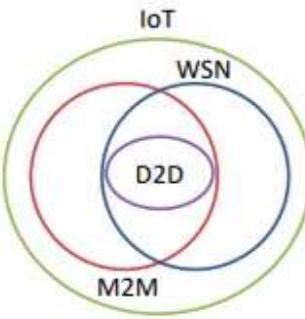
- la softwareizzazione e la virtualizzazione della rete (dovuta a SDN e NFV) sta apendo la strada all'integrazione nella rete dei nodi fog.
- il potenziamento delle antenne, che consentirà loro di fare molto di più della semplice comunicazione, consentirà l'edge computing;
- le onde millimetriche e la banda stretta le comunicazioni consentiranno comunicazioni a lungo raggio a bassa latenza e ad alta capacità.

### CONCETTI SOVRAPPOSTI:

**Macchina-macchina (M2M):** un insieme di metodi e protocolli che consentono ai dispositivi di comunicare e interagire su Internet (o su altre reti) senza intervento umano.

**Dispositivo-a-Dispositivo (D2D):** definito come comunicazione diretta tra due dispositivi mobili senza attraversare la stazione base (BS) o la rete core.

M2M	IoT
Comunicazione punto-a-punto solitamente integrata nell'hardware presso il sito del cliente	I dispositivi comunicano utilizzando reti IP, incorporando protocolli di comunicazione diversi
Molti dispositivi usano reti cellulari o cablate	La trasmissione dei dati avviene tramite un livello intermedio ospitato nel cloud
I dispositivi non richiedono necessariamente una connessione Internet	Nella maggior parte dei casi, i dispositivi richiedono una connessione Internet attiva
Opzioni di integrazione limitate, poiché i dispositivi devono avere standard di comunicazione corrispondenti	Opzioni di integrazione illimitate, ma è necessaria una soluzione in grado di gestire tutte le comunicazioni



## 1.2. SIMULAZIONE/EMULAZIONE

### Sviluppo del sistema:

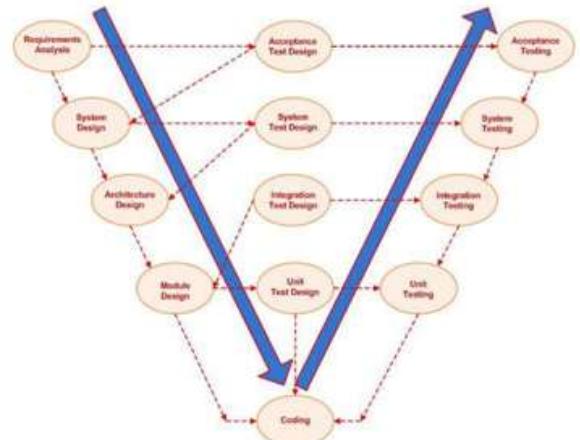
Il ciclo di vita dello sviluppo dei sistemi è un processo di pianificazione, creazione, test e distribuzione di un sistema informativo. Di solito, questo ciclo è composto da sei fasi:

- Analisi
- Progettazione
- Sviluppo e Test
- Implementazione
- Documentazione
- Valutazione

Il **modello V** è una rappresentazione grafica del ciclo di vita di sviluppo dei sistemi, utilizzato per produrre modelli rigorosi del ciclo di vita di sviluppo.

Il **lato sinistro** della "V" rappresenta la scomposizione dei requisiti e la creazione delle specifiche del sistema.

Il **lato destro** della "V" rappresenta l'integrazione delle parti e la loro convalida.



### TESTING:

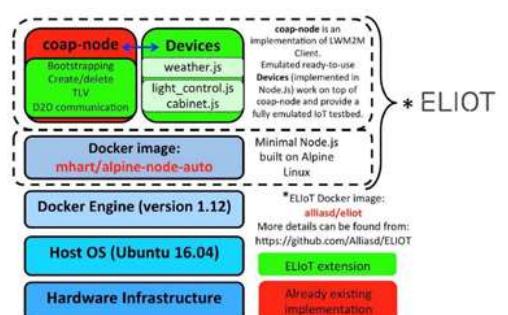
Parte del ciclo di vita dello sviluppo del sistema che certifica la qualità del prodotto. Testare l'IoT è molto difficile. Per farlo, possiamo utilizzare la simulazione o l'emulazione:

- **Simulazione:** Copia un sistema del mondo reale in un ambiente virtuale. Simula il comportamento di base ma non rispetta necessariamente tutte le regole dell'ambiente reale
- **Emulazione:** Duplica l'oggetto esattamente come esiste nella vita reale. Imitazione completa del sistema reale, ma in un ambiente virtuale.

Nel contesto IoT, l'approccio utilizzato è la simulazione della rete di sensori. Molti sistemi operativi dispongono di un emulatore ad hoc delle piattaforme hardware supportate.

### PIATTAFORMA IOT EMULATA:

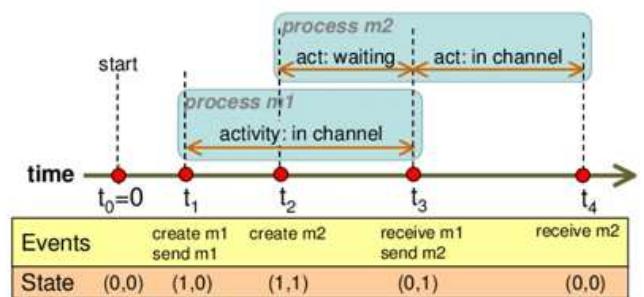
Emulated IoT Platform (ELIoT) ha la sua immagine Docker per archiviare le applicazioni del dispositivo e i file di configurazione necessari per eseguire i dispositivi emulati. È possibile lanciare sulla piattaforma un numero qualsiasi di container, ognuno con la propria interfaccia di rete e con una logica semplice che simula quella di un dispositivo IoT.



### SIMULATORI DI EVENTI DISCRETI:

La simulazione a eventi discreti (DES) modella il funzionamento di un sistema come una sequenza di eventi nel tempo:

- **Progressione temporale dell'evento successivo:** Il tempo di simulazione salta direttamente all'evento successivo.
- **Progressione temporale a incremento fisso:** Il tempo è suddiviso in sezioni e lo stato del sistema viene aggiornato in base agli eventi/attività di quel tempo.
- **Simulazione continua:** Lo stato del sistema cambia continuamente nel tempo.



## OMNET++:

Objective Modular Network Testbed in C++ è una libreria e un framework di simulazione C++ modulare e basato su componenti, utilizzato principalmente per la **creazione di simulatori di rete**.

- OMNeT++ utilizza moduli nidificati gerarchicamente e comunica tramite scambio di messaggi.
- Offre modelli per protocolli di rete e tecnologie di mobilità.

## PETRI NETS:

Un **grafo bipartito orientato** è una rappresentazione grafica in cui i **nodi** si suddividono in due insiemi distinti: **transizioni** e **luoghi**. Gli **archi orientati** (frecce) indicano le relazioni tra questi elementi e descrivono il comportamento del sistema.

In particolare:

- I **nodi** rappresentano:
  - **Le transizioni**: eventi che possono verificarsi all'interno del sistema
  - **I luoghi**: condizioni o stati del sistema.
- gli **archi diretti** specificano:
  - Quali **luoghi sono pre-condizioni** per una certa transizione (cioè devono contenere token affinché la transizione possa avvenire);
  - Quali **luoghi sono post-condizioni** della transizione (cioè riceveranno token una volta che la transizione è avvenuta).

Questo modello è tipico delle **reti di Petri** e viene utilizzato per descrivere il comportamento di sistemi distribuiti e concorrenti.

## QUEUEING THEORY:

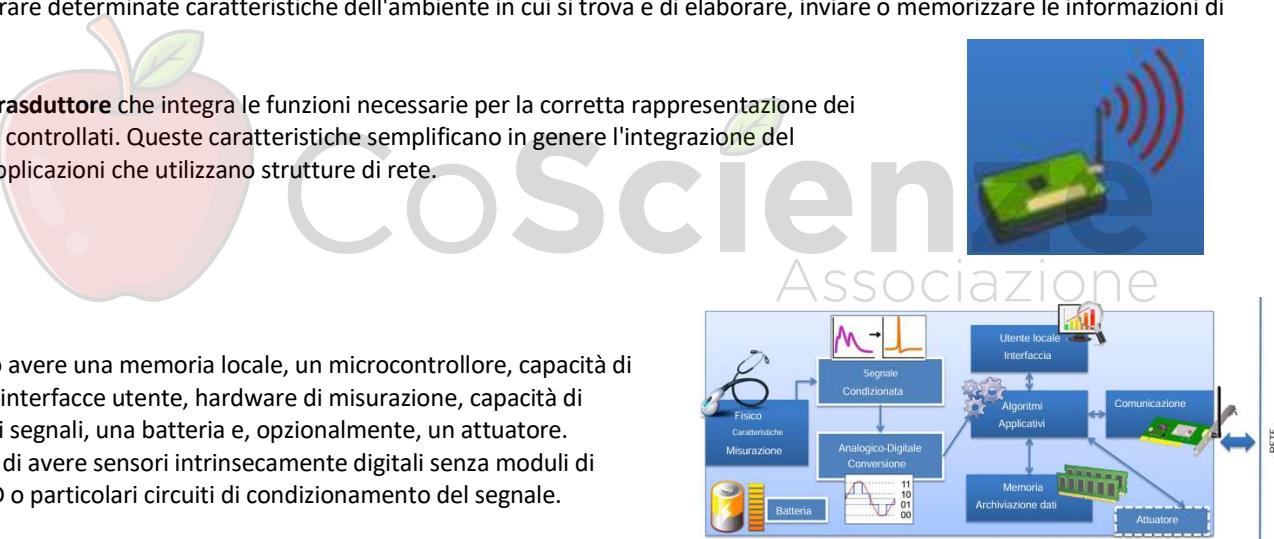
Una formulazione matematica che descrive le code prevedendone la lunghezza e il tempo di attesa. Una coda è composta da un buffer per i lavori in attesa e uno o più server che eseguono i lavori. Le code possono essere collegate tra loro per rappresentare l'interazione tra dispositivi IoT. Il sistema standard per descrivere e classificare le code è quello di Kendall, che utilizza tre fattori: A (tempo tra gli arrivi), S (distribuzione del tempo di servizio) e C (numero di canali di servizio aperti).

## 1.3. ANATOMIA DEL SENSORE E TASSONOMIA

### SENSORI:

Il componente di base di una rete di sensori distribuita è il **nodo sensore**, ovvero un piccolo dispositivo, dotato di una batteria limitata, in grado di monitorare determinate caratteristiche dell'ambiente in cui si trova e di elaborare, inviare o memorizzare le informazioni di monitoraggio.

Il sensore è un **trasduttore** che integra le funzioni necessarie per la corretta rappresentazione dei valori misurati o controllati. Queste caratteristiche semplificano in genere l'integrazione del trasduttore in applicazioni che utilizzano strutture di rete.



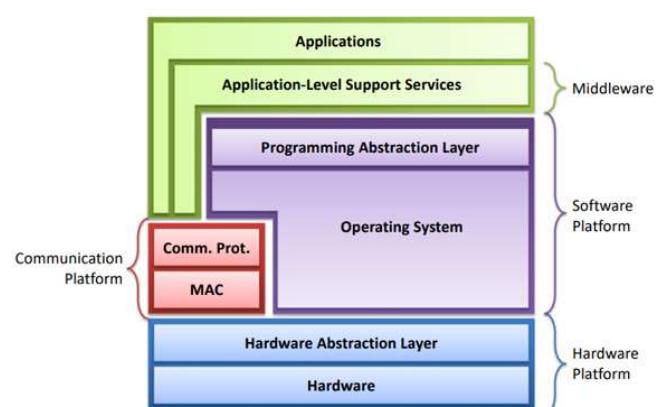
I sensori devono avere una memoria locale, un microcontrollore, capacità di comunicazione, interfacce utente, hardware di misurazione, capacità di elaborazione dei segnali, una batteria e, optionalmente, un attuatore.

Nulla impedisce di avere sensori intrinsecamente digitali senza moduli di conversione A/D o particolari circuiti di condizionamento del segnale.

### SCHEMA DEL NODO:

Il nodo è composto da 4 componenti:

- **Middleware**: È disponibile una serie di servizi di sistema per supportare le applicazioni nella realizzazione di una serie di aspetti orizzontali, come la sincronizzazione temporale, la localizzazione e l'instradamento delle applicazioni.  
Al di sopra di questi servizi si trova il codice applicativo, che gestisce sia gli aspetti orizzontali che quelli specifici del contesto applicativo per cui la rete è stata progettata e utilizzata.
- **Software Platform**: Sui nodi sensore, rappresenta una libreria collegata al codice applicativo per produrre un binario per l'esecuzione delle applicazioni sul nodo di rete. Di solito è supportato da un linguaggio di programmazione ad hoc.
- **Communication Platform**: Il supporto comunicativo è composto da una serie di algoritmi per accesso efficiente al mezzo di comunicazione, tenendo conto requisiti di risparmio energetico e una serie di reti e trasporti protocolli per la comunicazione wireless/cellulare.
- **Hardware Platform**: indica l'hardware su cui è costruito il nodo e l'interfaccia di programmazione a basso livello che permette di

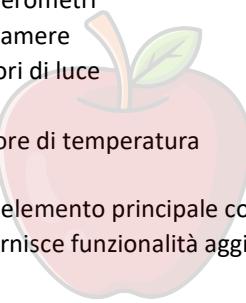


richiamarne le funzionalità.

## HARDWARE PLATFORM:

Grazie alla tecnologia **MEMS** (Micro-Electro-Mechanical Systems), ovvero elementi meccanici ed elettromeccanici miniaturizzati, realizzati mediante microfabbricazione, la piattaforma hardware sta diventando sempre più piccola. Possiamo dividere le piattaforme hardware in due classi principali:

- **Hardware generico:** piattaforme generiche e programmabili (come Arduino);
- **Hardware ad hoc:** specifico per attività di monitoraggio e controllo. Possiamo dividerlo in 4 categorie:
  - **Interfacce di comunicazione:** possiamo suddividerle in:
    - Interno:
      - **U(S)ART** (Universal (Synchronous) Asynchronous Receive Transmit): modulo per comunicare in modo asincrono e sincrono con un altro modulo sia trasmettendo che ricevendo dati secondo un protocollo. Bassa velocità.
      - **I2C** (Inter Integrated Circuit): protocollo che prevede lo “scambio” di dati tra uno o più Master e uno o più Slave, mediante un bus dati e uno di clock. Sono possibili tre velocità (100 KHz, 400 KHz, 1 MHz).
      - **SPI** (Serial Peripheral Interface): protocollo Master-Slave, con ognuno dotato di un registro a scorrimento contenente i dati e collegato con un minimo di quattro bus. Il trasferimento comporta lo scambio del contenuto del registro.
    - Esterno:
      - **Cablato**
      - **Senza fili**
        - **Bluetooth**
        - **Wifi**
  - **Apparecchiature di monitoraggio** che vengono classificate per:
    - Tecniche di elaborazione:
      - **Analogico:** la quantità elettrica prodotta in output varia in risposta alla variazione della quantità in input.
      - **Digitale:** i valori di uscita sono limitati o sono una rappresentazione discreta di una forma d'onda analogica, in genere i valori sono 0-1.
    - Obiettivo:
      - Accelerometri
      - Telecamere
      - Sensori di luce
      - GPS
      - Sensore di temperatura
  - **CPU/MCU**
    - CPU: è l'elemento principale con il compito di elaborare le operazioni
    - MCU: fornisce funzionalità aggiuntive come timer, oscillatore, memoria, I/O o I/O per uso generale
  - **Batterie**



Coscienze  
Associazione

## 2.1. ASTRAZIONE SOTWARE DEI SENSORI

### SOFTWARE PLATFORM:

Per aiutare i progettisti, sono stati proposti molti **sistemi operativi per reti di sensori**. Essi sono come librerie collegate al codice dell'applicazione. Il sistema operativo solitamente supporta uno specifico linguaggio di programmazione, che è tipicamente un dialetto di C o specifico per reti di sensori.

I sistemi operativi tradizionali sono utilizzati principalmente per gestire processi, memoria, tempo CPU, file system e dispositivi e solitamente realizzati in modo modulare e a livelli, tra cui un livello basso chiamato kernel e un livello alto di librerie di sistema.

I sistemi operativi tradizionali **non sono ottimali** per le reti di sensori perché queste reti hanno risorse limitate e diverse applicazioni incentrate sui dati, oltre a una topologia variabile. A tal fine, le WSN necessitano di nuove soluzioni che tengano conto delle loro peculiarità e dei loro requisiti:

- **Gestione dei processi e pianificazione:** i sistemi operativi tradizionali allocano i processi in uno spazio di memoria separato, causando tempi di gestione elevati per il cambio di contesto tra processo e consumo elevato;
- **Gestione della memoria:** nei sistemi tradizionali la memoria è allocata esclusivamente a un processo o a un'attività, ciò non è possibile nei nodi sensore che hanno una scarsa capacità di archiviazione;
- **Modello del kernel:** Tradizionalmente, un kernel è composto da vari moduli interdipendenti e chiamate di sistema per servizi che vengono eseguiti in modalità supervisore. I sistemi operativi per i nodi sensori si basano su modelli di macchine a stati o di eventi.
- **Interfaccia di programmazione dell'applicazione (API):** i nodi sensore richiedono API modulari e generali per le loro applicazioni, ma consentono anche un facile accesso all'hardware sottostante;
- **Aggiornamento e riprogrammazione del codice;**
- **Gestione dei consumi energetici:** Prolungare il ciclo di vita del nodo.

### MODELLO CONCORRENTI:

I nodi sensore possono eseguire molte attività contemporaneamente, come raccolta e trasmissione dati, gestione degli eventi temporizzati.

- **Modello a eventi:** più efficiente nell'uso della memoria rispetto al modello multithread;
- **Modello multithread:** utilizzato nei sistemi operativi tradizionali, ma meno efficiente per i nodi sensore;
- **Modello ibrido:** Combina i vantaggi dei modelli a eventi e multithread, esempio il modello "protothread" che fornisce un flusso di controllo sequenziale, simile al modello multi thread, ma con un solo stack.

### ALLOCAZIONE DI MEMORIA:

La **memoria nei nodi sensore è limitata** e deve essere gestita attentamente per evitare la frammentazione. La memoria è divisa in:

- **Statica:** Contiene il programma e viene conservata in memoria ROM;
- **Dinamica:** Contiene variabili runtime, buffer, dati di input/output e lo stack, conservati in RAM;

L'approccio principale è l'allocazione statica per ridurre il rischio di frammentazione.

### CONSUMO ENERGETICO:

La gestione dell'energia è cruciale per prolungare il ciclo di vita dei nodi sensore. Strategie di gestione energetica includono lo spegnimento dei componenti non utilizzati e il monitoraggio dei consumi sia a livello hardware che software.

### TINY OS:

TinyOS è un sistema operativo open source, basato su componenti e specifico per applicazione, **sviluppato specificamente per reti di sensori**. Può supportare programmi concorrenti con basso consumo energetico requisiti di consumo e ha una bassa occupazione di memoria (circa 400 byte). Rappresenta un esempio di **architettura non monolitica**, che utilizza un modello di componenti che viene scelto e assemblato in base alle esigenze delle applicazioni.

Un componente è un'entità di calcolo indipendente che ha una o più interfacce. I componenti hanno tre astrazioni computazionali: comandi, eventi e task. I primi due sono meccanismi di comunicazione tra componenti (rispettivamente uno sincrono e uno asincrono), mentre il terzo è utilizzato per esprimere la competizione all'interno di un componente.

TinyOS fornisce un singolo stack condiviso, senza alcuna separazione tra spazio kernel e spazio utente. TinyOS adotta un modello di esecuzione e concorrenza con eventi.

Supporta la pianificazione non preventiva e ha introdotto un modello ibrido con supporto al multithreading dalla versione 2.1.

### NESC:

Linguaggio di programmazione basato su componenti ed eventi, derivato dal C, progettato per sviluppare TinyOS. Le applicazioni nesC sono costituite da componenti collegati tramite interfacce che gestiscono comandi ed eventi.

### SISTEMI EMBEDDED:

Un **sistema embedded** è un dispositivo progettato per svolgere una funzione specifica. Ha risorse limitate e include un motore di elaborazione (CPU). Si distinguono per dimensione:

- **Piccoli sistemi:** CPU a bassa potenza, almeno 2 MB di ROM e 4 MB di RAM;
- **Sistemi medi:** circa 32 MB di ROM e 64 MB di RAM;
- **Grandi sistemi embedded:** CPU potenti e memoria più ampia.

Con l'abbassamento dei costi delle memorie flash e la diffusione di **System on Chip (SoC)** compatti e a basso consumo, Linux è diventato la piattaforma ideale per questi sistemi.

## EMBEDDED LINUX:

Un **Embedded Linux** è semplicemente un sistema embedded che utilizza il kernel Linux. Non esiste una “versione embedded” del kernel: si utilizza una versione standard, spesso configurata per l’hardware specifico del dispositivo.

Linux supporta le architetture ARM, x86, MIPS e PowerPC (da 32 bit in su) e può essere adattato con facilità grazie alla sua modularità.

Un sistema Linux embedded è composto da:

- **Bootloader**: piccolo programma che avvia il kernel Linux e inizializza alcune periferiche;
- **Kernel**: gestisce le risorse hardware e fornisce API per le applicazioni utente;
- **Filesystem**: contiene i programmi e i dati in spazio utente, inclusi i servizi di avvio (init system).

Il **device tree** è caricato dal bootloader in RAM e descrive la configurazione hardware della scheda.

Per creare un sistema Linux embedded, è necessario:

- **Cross-compilare** il kernel, il bootloader e il root filesystem su una macchina di sviluppo;
- Usare strumenti come **Make**, con supporto da **autotools** o **CMake**, per gestire la compilazione;
- Gestire manualmente le dipendenze o affidarsi a **build system automatici**, che semplificano il processo.

## YOCOT PROJECT:

**Yocto** è un progetto open source che fornisce strumenti, template e metadati per creare sistemi Linux embedded personalizzati. Permette di:

- Costruire un sistema completo da sorgente;
- Supportare tutte le principali architetture embedded;
- Offrire strumenti per lo sviluppo di kernel e applicazioni;
- Personalizzare facilmente in base alle esigenze del progetto.

## 2.2. MIDDLEWARE

### INTRODUZIONE:

Il middleware colma il divario tra protocolli di rete e applicazioni, nascondendo i dettagli di basso livello per facilitare lo sviluppo, la distribuzione e la manutenzione delle applicazioni. Il middleware per le reti di sensori deve essere leggero, mentre quelli tradizionali hanno una sofisticata stratificazione software. Le sfide nella progettazione di middleware per sensori sono:

- **Controllo della topologia**: per organizzare i sensori in una rete;
- **Elaborazione basata sui dati**: con consumo energetico ottimizzato;
- **Integrazione specifica dell’applicazione**: per migliorare le prestazioni e risparmiare energia;
- **Uso efficiente delle risorse informatiche e di comunicazione**;
- **Supporto per applicazioni in tempo reale**.

### ARCHITETTURA:

Il **Programming Abstraction Layer** rappresenta l’interfaccia middleware per le applicazioni. Al suo interno vengono contenute le implementazioni degli abstract forniti nella sezione dei **Common Services** mentre invece, la zona del Run-Time Support serve come estensione del sistema operativo per ottenere le astrazioni sovrapposte. La zona del **Vertical Support** contiene meccanismi volti ad imporre restrizioni particolari alla qualità del servizio.

Il **middleware** agisce, quindi, come livello intermedio tra applicazioni e infrastruttura di rete, facilitando la comunicazione e l’interoperabilità.

Nelle reti di sensori, il middleware si adatta alle tre entità fondamentali (nodi, gateway, e backend), **differenziando i suoi compiti** a seconda del contesto.

I servizi di middleware si dividono in:

- **Servizi comuni (orizzontali)**: validi per qualsiasi contesto:
  - **Gestione del codice**: gestisce aggiornamenti e migrazione del codice in rete;
  - **Gestione dei dati**: raccoglie, archivia, sincronizza, analizza e interroga i dati condivisi all’interno della rete;
  - **Risorse di scoperta**: individua nuovi nodi o segnala quelli non più disponibili;
  - **Gestione delle risorse**: ottimizza risorse hardware (memoria, comunicazione, topologia);
  - **Integrazione**: collega reti differenti.

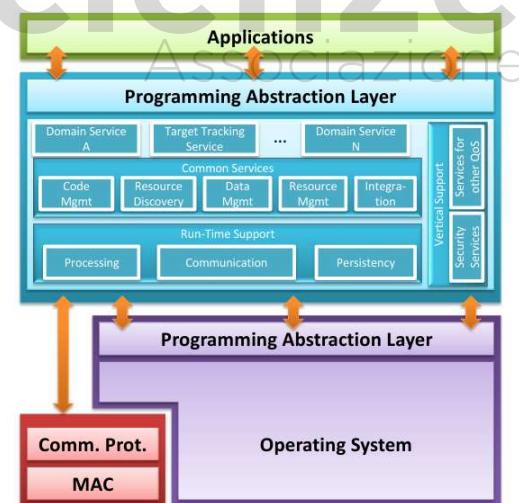
- **Supporto runtime**: è un’estensione del sistema operativo che fornisce:

- Pianificazione dei processi;
- Comunicazione tra processi (IPC);
- Gestione della memoria;
- Controllo dell’energia

Spesso è implementato come **macchina virtuale** per garantire compatibilità tra diverse piattaforme hardware. Un meccanismo di Qualità del Servizio (QoS) può essere integrato trasversalmente

- **QoS (Qualità del servizio)**:

- La **rete** esprime la QoS in termini di:



- Ritardo
  - Jitter
  - Perdita di pacchetti
  - Banda
  - Produttività
- Le **applicazioni** richiedono:
    - Accuratezza dei dati
    - Ritardo aggregato
    - Copertura
    - Durata operativa

Il middleware agisce come **intermediario**, confrontando le richieste dell'applicazione con le capacità offerte dalla rete. Se non c'è corrispondenza, il middleware **negozia nuovi parametri QoS** secondo un principio di **domanda e offerta**.

## CLASSIFICAZIONE:

Sono state proposte diverse classi di middleware per le WSN, molte delle quali sono state sviluppate nel contesto di progetti di ricerca accademica e non hanno superato la fase di mero prototipo, altre sono maturate come veri e propri prodotti commerciali o open source di ampio utilizzo. La **classificazione dei vari approcci middleware** è la seguente:

- **Approccio al database:** La rete di sensori è vista come un grande database; le query sono eseguite usando SQL. Esempio: TinyDB;
- **Approccio basato su eventi:** Modello publish/subscribe per definire, rilevare e distribuire eventi. Minimizza le trasmissioni. Esempio: TinyDDS;
- **Approccio basato sulle applicazioni:** Le applicazioni hanno privilegi fino allo stack del protocollo di rete in base ai loro requisiti;
- **Approccio con Macchine Virtuali (VM):** Creazione di macchine virtuali eseguite sui sensori. La modularità riduce la memoria e aumenta il consumo di energia. Esempio: LabVIEW;
- **Approccio basato sullo spazio delle tuple:** Paradigma della memoria condivisa con repository di tuple accessibile contemporaneamente da produttori e consumatori. Esempio: TinyLime
- **Approccio in base al servizio:** Middleware di servizio rende i servizi disponibili e facilmente accessibili secondo protocolli standardizzati. Esempio: TinySOA.

## TINYDB:

- Sistema di query per reti di sensori su TinyOS;
- Permette interrogazioni in stile SQL, senza scrivere codice nesC;
- Considera l'intera rete di sensori come un database;
- Composto da GUI, API client, motore di query, e interfaccia JDBC;
- Occupa circa 58 KB di codice compilato, con 3200 B in RAM;
- Ogni sensore ha attributi descritti in un file catalog.xml.

## TINYDSS:

- Implementa lo standard **OMG DDS** (Data Distribution Service) su TinyOS e SunSPOT;
- Supporta operazioni **publish/subscribe** per applicazioni distribuite;
- Funziona sia sui nodi sensore che sulla stazione base;
- Supporta routing personalizzabile tramite protocolli OERP (Overlay Event Routing Protocols).

## LABVIEW:

- Ambiente di sviluppo grafico usato per l'acquisizione, analisi dati e automazione;
- Permette di programmare i nodi sensori graficamente;
- Può inviare dati solo quando superano una soglia → risparmio energetico;
- Consente l'integrazione di codice C all'interno dei diagrammi grafici.

## TINYLIME:

- Adatta il middleware **Lime** (basato su tuple space) a TinyOS;
- Ogni client interagisce con i sensori attraverso tuple predefinite;
- Usa due tuple space: uno per i dati sensore, uno per le richieste;
- Se i dati non sono presenti o sono obsoleti, i sensori vengono interrogati;
- Composto da:
  - **MoteAgent**: riceve richieste;
  - **TOSMoteAccess**: le gestisce;
  - **Componente lato sensore**: legge ed elabora i dati.

## TINYSOA:

- Middleware orientato ai servizi (SOA) per reti di sensori;
- Fornisce un'**API semplice** per accedere alle reti wireless;
- Incapsula tutte le funzioni del nodo di monitoraggio;

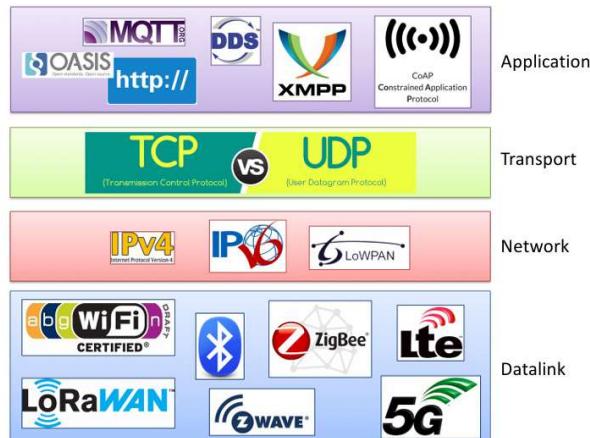
- Può essere installato su nodi specializzati o su un computer esterno;
- Si comporta come un **web service provider**, esponendo ogni rete come un servizio accessibile da remoto.

## FIWARE:

- Framework open source per costruire soluzioni Smart (Smart City, Smart Agriculture, ecc.);
- Il **FIWARE Orion Context Broker** gestisce e fornisce accesso ai dati da diverse fonti;
- Supporta vari protocolli (standard o proprietari), traducendoli nel formato **NGSI**;
- Gli oggetti del mondo reale (persone, stanze, sensori) sono rappresentati come **entità virtuali** con ID, tipo e attributi;
- Architettura cloud basata su **OpenStack** e container con **Docker GE**;
- Include “**Generic Enablers**”, componenti riutilizzabili per sviluppare, distribuire e gestire applicazioni IoT e cloud.

## 2.3. COMMUNICATION PLATFORM

### PIATTAFORMA DI COMUNICAZIONE (COMMUNICATION STACK):



### TECNOLOGIE WIRELESS:

Per rendere l'uso delle reti di sensori **più economico ed efficiente**, si preferisce utilizzare **infrastrutture wireless commerciali esistenti**, evitando di progettare soluzioni hardware completamente nuove.

- Un **nodo sensore** può supportare **una o più tecnologie wireless** contemporaneamente (es. con piattaforme come Waspmote);
- I progettisti **non necessitano competenze radio avanzate**, ma devono conoscere caratteristiche come: consumo energetico, copertura, banda, prestazioni e sicurezza;

Le bande di frequenza utilizzate sono:

- **Bande ISM:** 900 MHz, 2.4 GHz, 5.8 GHz (es. Bluetooth, IEEE 802.11b/g);
- **Bande UNII:** 5.15–5.825 GHz (es. IEEE 802.11a) → maggiore velocità (fino a 54 Mbps) ma anche maggior consumo;

Le interferenze possono derivare da:

- Fenomeni naturali (riflessione, rifrazione, diffrazione, multipath);
- Le reti WSN sono **sensibili alle interferenze radio**, anche se provengono da altre tecnologie IEEE (PAN, LAN, MAN)

Tra le principali tecnologie wireless troviamo:

- **Bluetooth:** ottimo per **comunicazioni a corto raggio, basso consumo e semplicità d'uso**, con velocità di 1–3 Mbps su banda 2.4 GHz. Ideale per collegamenti tra dispositivi personali o sensori vicini;
- **Wi-Fi:** offre **alte prestazioni e alta velocità di trasmissione**, ma ha un **consumo energetico elevato**;
- **ZigBee:** ottima per **basso consumo energetico e lunga durata**. Ideale per ambienti remoti o difficili da raggiungere
- **WiMAX:** adatto per **trasmissione a lunga distanza**, con buone velocità (30–40 Mbps), spesso usato per copertura estesa.
- **RFID/NFC:** utilizzato per **identificazione e scambio di dati a corto raggio**. NFC è limitato a pochi centimetri, ma consuma pochissimo

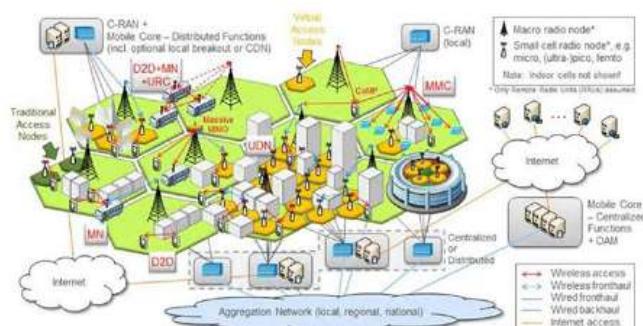
### 5G:

Il 5G è ancora in fase iniziale di implementazione e non è disponibile in tutti i Paesi. Se ne sente parlare spesso in relazione a:

- **Spettro 5G;**
- **Aste delle frequenze;**
- **Onde millimetriche (mmWave):** frequenze ad alta banda, trasmettono molti dati
  - **Vantaggi:** Ideale per ambienti urbani densamente popolati con molti dispositivi;
  - **Svantaggi:** Fortemente attenuate da ostacoli (alberi, edifici, umidità). Poco adatte a lunghe distanze.

Il 5G lavora su diverse frequenze:

- **600 MHz (bassa banda):** minore larghezza di banda, ma migliore penetrazione attraverso i muri, maggiore portata, utile per aree rurali e interni;
- **2-6 GHz (Banda C):** equilibrio tra copertura e capacità;
- **< 2 GHz:** ottima copertura indoor e portata più ampia;



- > 6 GHz (Super Data Layer): alta velocità e larghezza di banda ma copertura ridotta.

I fornitori di servizi combinano bande:

- **Basse frequenze:** per ampia copertura da una singola torre;
- **Alte frequenze:** per grande capacità in aree ad alta densità di traffico dati.

## RETE WAN A BASSA POTENZA:

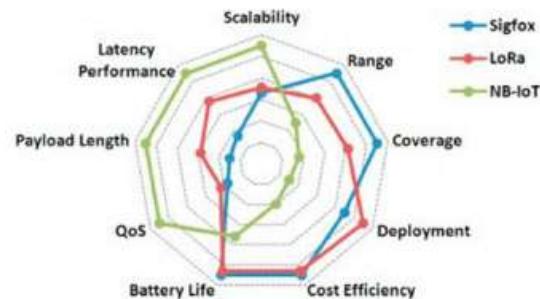
Sono progettate per trasmissioni a lungo raggio (5–40 km) con basso consumo energetico. Usano bande **sub-GHz** (es. 868 MHz in Europa), che garantiscono **elevata robustezza al rumore**, ma con **basse velocità di trasmissione** (qualche kbps).

Ideali per applicazioni IoT che richiedono **piccole quantità di dati** e **autonomia prolungata**.

**Costi contenuti:** un modulo radio < 2 €, costo operativo ~1 €/anno per dispositivo.

Tra le varie reti WAN a bassa potenza troviamo:

- **Sigfox** e **LoRaWAN**: ideali per **comunicazioni rare**, su **lunghe distanze**, con **autonomia batteria molto estesa**;
- **LoRaWAN**: più affidabile in condizioni dinamiche (es. dispositivi in movimento);
- **NB-IoT**: più adatto a casi in cui servono bassa latenza e qualità di servizio elevata (es. ambienti industriali o urbani complessi).



## IPV6 PER IOT:

L'**IPv6** è l'evoluzione di IPv4, con uno **spazio di indirizzamento molto più ampio**: fino a  $2^{128}$  **indirizzi unici**, ideale per miliardi di dispositivi IoT.

Migliora funzioni di rete come:

- **Routing**;
- **Mobilità**;
- **Sicurezza**, grazie a una gestione nativa e ottimizzata di **IPSec**.

Con IPv4 si usa il **NAT (Network Address Translation)** per superare il limite degli indirizzi disponibili. Problemi del NAT:

- Gli utenti non hanno un indirizzo IP pubblico → non sono raggiungibili direttamente;
- Si perde la **connessione end-to-end** e si **indebolisce l'autenticazione**.

Per adattare IPv6 ai dispositivi **vincolati** (con poca memoria e potenza), si è sviluppato:

- **6LoWPAN**: versione **compressa di IPv6**, che riduce la lunghezza degli indirizzi;
- I **router di confine** traducono gli indirizzi compressi in indirizzi IPv6 standard.

Esempio: **Contiki OS**, uno stack IPv6 completo per dispositivi con soli 11,5 KB di memoria. IPv6 supporta la **configurazione automatica degli indirizzi (stateless)**: i nodi possono **auto-assegnarsi un indirizzo IP**, riducendo costi e complessità di gestione.

## ROUTING:

Le **reti di sensori** sono molto versatili e applicabili in contesti diversi. I **nodi sensori** operano spesso in modo ad hoc e con poca manutenzione. È fondamentale che trasmettano dati con protocolli di **routing efficienti dal punto di vista energetico**.

Le WSN operano in ambienti complessi con **risorse e potenza limitate**. Il routing deve essere progettato per **massimizzare la durata della rete**. La scelta dei percorsi è quindi cruciale ma complicata. Ci sono due tipologie di modelli di routing:

- **Single-hop**: Ogni nodo comunica **direttamente con la stazione base**, semplice da implementare ma ha un'alta richiesta energetica (inefficiente su grandi aree);
- **Multi-hop**: i dati passano attraverso **nodi intermedi**, usando distanze più corte, risparmio energetico, meno interferenze, possibilità di aggregare dati lungo il percorso, **utilizzato in soluzioni moderne** (IoT → cloud o nodi Edge/Fog).

I nodi possono organizzarsi in due modi:

- **Mesh network**: collegamenti diretti, dinamici e non gerarchici tra più nodi → Auto-organizzazione e auto-configurazione;
- **Cluster network**: struttura gerarchica per ottimizzare la comunicazione tra gruppi di nodi.

Il routing multi-hop è più efficiente ma **richiede cooperazione tra i nodi intermedi**. Determinare il **percorso ottimale** è complesso e centrale per il buon funzionamento della rete.

Protocolli di routing specifici per le reti di sensori IoT sono:

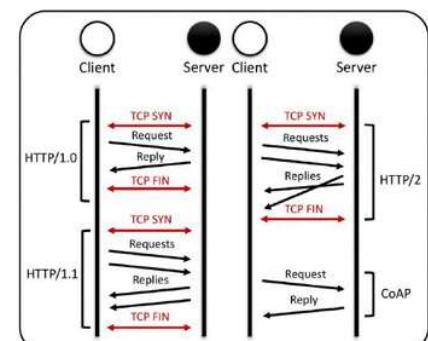
- **RPL (Routing Protocol for Low-Power and Lossy Networks)**: Protocollo di routing progettato per reti con risorse limitate e alta perdita di pacchetti;
- **Protocolli di routing ad hoc**: Ad esempio AODV (Ad hoc On-Demand Distance Vector) e DSR (Dynamic Source Routing), utilizzati per reti mobili e dinamiche.

## PROTOCOLLI DI COMUNICAZIONE:

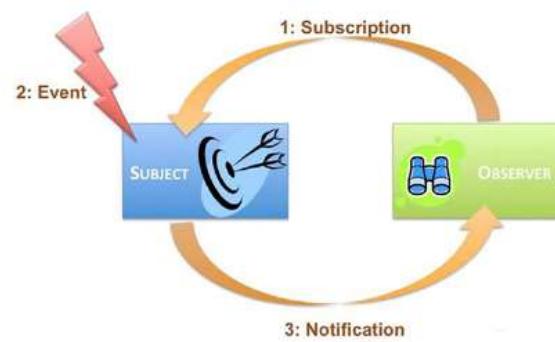
In ambito IoT, i protocolli si differenziano principalmente in base al **modello di interazione** tra i dispositivi: nei sistemi distribuiti troviamo diversi approcci fondamentali, ognuno con vantaggi distinti in termini di efficienza, scalabilità e aderenza al contesto applicativo:

- **Modello Request-Response**: è il paradigma più basilare, un client invia una richiesta, il server risponde. Viene utilizzato in ambienti centralizzati. I principali protocolli di questo modello sono:

- **REST/HTTP**
- **COAP**



- **Modello Publish/subscriber:** Usato per comunicazioni distribuite, asincrone e debolmente accoppiate. **Publisher** invia dati, **Subscriber** li riceve tramite un **broker**, senza che i due si conoscano. È più adatto per **sistemi IoT scalabili, distribuiti e dinamici**, dove l'interazione tra i dispositivi non è predeterminata.
- Vantaggi:**
- **Disaccoppiamento** tra produttore e consumatore di dati
  - Migliora **scalabilità e modularità** del sistema
  - **Non è richiesta la sincronizzazione** temporale tra publisher e subscriber.



Le architetture del servizio di notifica (broker) sono di due tipologie:

- **Centralizzata:** unico broker;
- **Distribuita:**
  - **Local broker:** vicini all'applicazione;
  - **Border broker:** collegano segmenti della rete;
  - **Inner broker:** instradano eventi tra nodi interni.

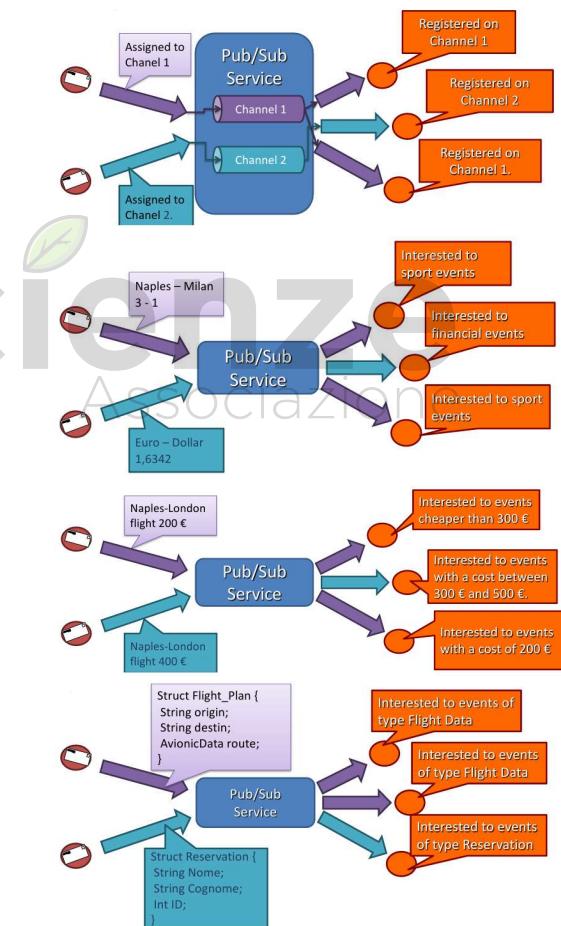
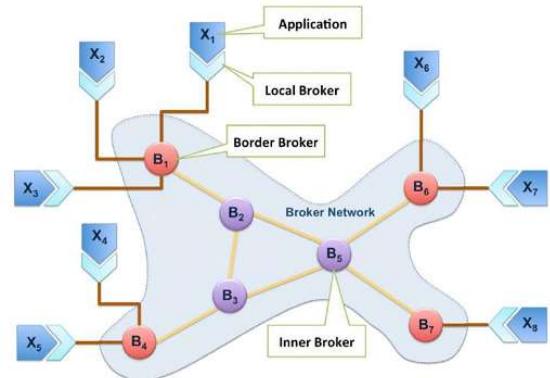
Soluzioni standard basate su publish/subscribe (es. Azure IoT Hub) sono ampiamente usate a livello industriale. In alcuni casi si preferisce un'infrastruttura "zero":

- Nessun broker fisso;
- I nodi si scoprono e comunicano autonomamente.

#### TIPI DI ABBONAMENTO NEI SISTEMI PUBLISH/SUBSCRIBE:

Le soluzioni middleware per eventi si distinguono in base alla strategia con cui gli abbonati ricevono le notifiche. Esistono quattro principali modalità:

- **Basato sul canale:**
  - Gli eventi vengono pubblicati su canali identificati;
  - Gli abbonati ricevono tutti gli eventi associati al canale su cui sono registrati;
  - Modello semplice, adatto a flussi separati di eventi.
- **Basato sull'argomento (topic-based):**
  - Gli eventi sono etichettati con una stringa di argomento (es. "sport", "finanza");
  - Gli abbonati dichiarano interesse per uno o più argomenti specifici;
  - Diffuso nei sistemi MQTT e notizie in tempo reale.
- **Basato sul contenuto (content-based):**
  - L'interesse è espresso come condizione logica sul contenuto dei dati;
  - Gli eventi vengono inviati solo se soddisfano un predicato specificato dal subscriber;
  - Maggiore flessibilità e selettività, ma più complesso da implementare.
- **Basato sul tipo:**
  - Gli eventi sono strutturati secondo un tipo di dato (schema);  
Gli abbonati si registrano a eventi di un determinato tipo strutturato (es. FlightData, Booking);
  - Ideale per sistemi fortemente tipizzati e per l'integrazione con linguaggi di programmazione.



#### MESSAGE QUEUEING (ACCODAMENTO DEI MESSAGGI):

Il servizio di notifica è realizzato tramite **code di messaggi**, ospitate su un server centralizzato o su una rete federata di server. Il **produttore (publisher)** invia ("push") il messaggio nella coda. I **consumatori (subscribers)** iscritti alla coda possono recuperare ("pop") i messaggi da essa.

È possibile associare più consumatori a una singola coda o argomento. Tuttavia, **ogni messaggio è ricevuto solo da un consumatore** per volta (non broadcast). Una volta elaborato, il messaggio viene rimosso dalla coda. Il destinatario del messaggio **dipende dalla logica dell'implementazione** (round robin, priorità, ecc.). Alcune architetture adottano modelli ibridi, combinando il Publish/Subscribe con il Message Queueing. Questo consente maggiore flessibilità nella gestione di carichi, priorità e strategie di distribuzione dei messaggi.

#### HTTP (HYPERTEXT TRANSFER PROTOCOL):

È il Protocollo client-server classico del Web. Basato su TCP e sul modello **richiesta-risposta**. REST associa operazioni CRUD a metodi HTTP (GET, POST, PUT, DELETE). **Svantaggi** per l'IoT:

- Overhead elevato (header lunghi, handshake TCP);
- Elevato consumo energetico;
- Poco adatto per notifiche push.

**HTTP/2.0** migliora l'efficienza:

- Multiplexing (più richieste in parallelo);
- Server push;
- Header compressi;
- Ma ancora poco testato in contesti IoT.

### COAP (CONSTRAINED APPLICATION PROTOCOL):

Simile a HTTP ma **ottimizzato per dispositivi a bassa potenza**. Utilizza **UDP** per ridurre overhead (ma può usare TCP se necessario). Header, metodi e codici sono **codificati in binario**. Supporta il paradigma REST e richieste osservabili (observe). Include un meccanismo di **acknowledgement** per QoS. Estendibile con un **broker publish/subscribe** (draft IETF).

### MQTT (MESSAGE QUEUE TELEMETRY TRANSPORT):

Protocollo **publish/subscribe**, leggero, basato su **TCP**. Richiede un **broker MQTT** (es. Mosquitto). Tre livelli di **QoS**:

- **QoS 0**: consegna con la massima cura, senza conferma di ricezione (best-effort);
- **QoS 1**: il messaggio arriverà tramite uno schema di ritrasmissione (almeno una volta);
- **QoS 2**: il messaggio verrà consegnato esattamente una volta senza duplicazioni (esattamente una volta).

Supporta il flag retain per memorizzare l'ultimo messaggio per nuovi subscriber. Variante **MQTT-SN** usa **UDP** e ID numerici per ridurre il payload, ma ha scarsa diffusione (utile per l'IoT).

### DDS (DATA DISTRIBUTION SERVICE):

Protocollo **publish/subscribe decentralizzato**, senza broker. Basato su **UDP** (ma supporta anche TCP). Architettura **peer-to-peer**: meno punti di fallimento. **Data-centric**, non message-centric. Ampio supporto QoS (oltre 20 livelli). Usa il protocollo **RTPS** per interoperabilità tra vendor. Adatto per sistemi **real-time distribuiti**.

### AMQP (ADVANCED MESSAGE QUEUING PROTOCOL):

Protocollo **open standard** per messaging. Versione 0.9.1: publish/subscribe con **broker e code**. Versione 1.0: **peer-to-peer, senza broker**. Usa **TCP** e supporta QoS simili a MQTT. Adatto a contesti aziendali complessi.

### XMP (EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL):

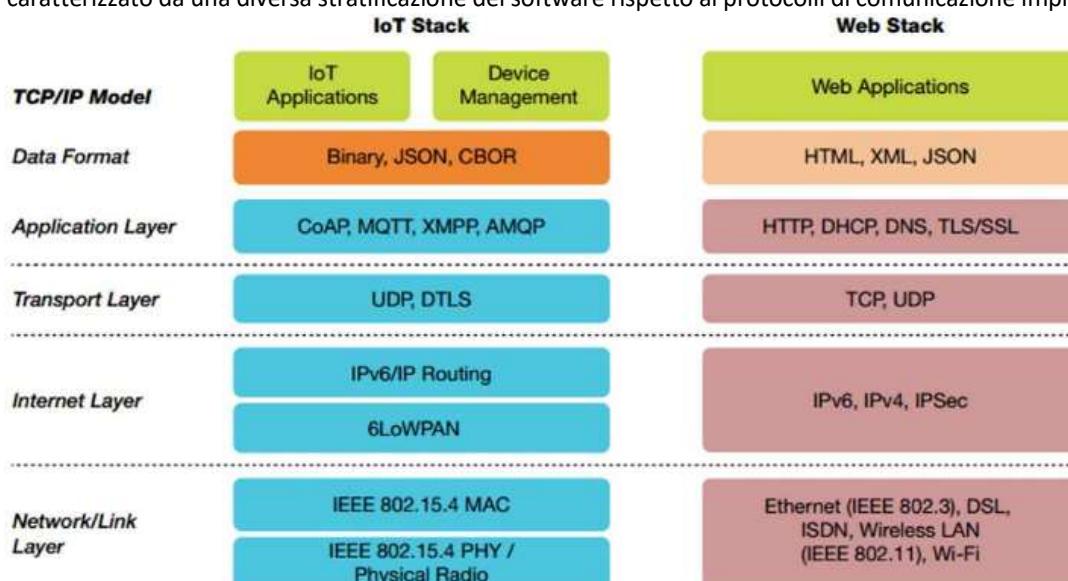
Basato su **XML** e **TCP**, nato per la messaggistica istantanea. Supporta sia il modello **client-server** che **publish/subscribe**. Ogni utente ha un JID simile a un'email (es. user@dominio/dispositivo). Offre **presence management** e **service discovery** (XEP-0030).

**Svantaggi:**

- XML poco efficiente: non adatto a reti a bassa larghezza di banda;
- Mancanza di garanzie QoS;
- Non ideale per dispositivi con risorse limitate, anche se ci sono evoluzioni in corso per adattarlo all'IoT.

### IOT VS WEB:

L'IoT è tipicamente caratterizzato da una diversa stratificazione del software rispetto ai protocolli di comunicazione implementati.



Differenze tra IoT e web:

- **IoT**: Utilizza protocolli specifici ottimizzati per risorse limitate e basse latenze, come MQTT e CoAP;
- **Web**: Si basa su protocolli tradizionali come HTTP, meno ottimizzati per dispositivi IoT.

## PROTOCOLLI DI SCOPERTA:

L'elevata **scalabilità** dell'IoT richiede meccanismi in grado di: registrare e scoprire dinamicamente risorse e servizi, operare in modalità autoconfigurata e senza intervento manuale, funzionare anche senza infrastruttura centralizzata.

### MDNS (MULTICAST DNS):

Permette di **usare nomi DNS in reti locali** senza configurazioni extra. Funziona senza bisogno di server DNS centrali.

#### Vantaggi:

- Nessuna amministrazione manuale;
- Funzionamento autonomo e senza infrastruttura;
- Resiliente in caso di guasti.

Funziona inviando richieste **multicast** per risolvere nomi in indirizzi IP tra dispositivi della rete. I dispositivi aggiornano le loro **cache locali** con nome e IP ricevuti.

### DNS-SD (DNS SERVICE DISCOVERY):

Estensione di mDNS per scoprire servizi specifici (es. stampanti, server). Permette a un client di trovare servizi senza sapere dove sono ospitati. Utilizza lo **spazio dei nomi DNS** per pubblicare e cercare servizi. Opera tramite pacchetti multicast su UDP. Esegue due passaggi:

- Ricerca del nome host del servizio;
- Associazione del nome all'indirizzo IP.

### LIMITI DI MDNS E DNS\_SD:

L'uso di mDNS/DNS-SD richiede **memorizzazione nella cache**, che può essere un problema per dispositivi con **risorse limitate**. Soluzione: gestire la cache con **scadenze temporali** (TTL) e svuotamento periodico.

## IMPLEMENTAZIONI PRATICHE:

- **Bonjour** (Apple);
- **Avahi** (Linux);

Entrambe supportano **mDNS + DNS-SD** e sono usate in molte applicazioni IoT e ambienti di rete locale.

## 2.4. SUPPORTO CLOUD

### INTRODUZIONE:

Il **cloud** indica risorse (dati, software, servizi) **accessibili da remoto** tramite Internet o reti private. Permette di usare **applicazioni online** (es. e-mail, CRM, videoconferenze) **senza installarle localmente**. Le caratteristiche di un sistema Cloud sono:

- Accesso da qualsiasi luogo tramite connessione Internet;
- Piattaforma indipendente: elimina i problemi di compatibilità;
- Favorisce mobilità, collaborazione e scalabilità;
- Con l'uso di thin client, il carico di elaborazione è sul server anziché sul dispositivo locale.

### MODELLO DI DISTRIBUZIONE:

- **Public Cloud**: accessibile a tutti; meno sicuro (es. servizi email pubblici);
- **Private Cloud**: accessibile solo all'interno di un'organizzazione; più sicuro;
- **Community Cloud**: condiviso tra più organizzazioni con interessi comuni;
- **Hybrid Cloud**: combina public e private cloud; le attività critiche restano nel cloud privato.

### MODELLO DI SERVIZIO:

Il Cloud Computing offre flessibilità, riduzione dei costi e accessibilità. Grazie a vari modelli di distribuzione e servizio, è diventato una tecnologia chiave per aziende, sviluppatori e utenti finali. Il cloud offre vari modelli di servizio:

- **IaaS (Infrastructure as a Service)**: fornisce **infrastruttura IT virtuale** (rete, storage, server)

#### Vantaggi:

- Risparmio su hardware e manutenzione;
- Scalabilità on-demand;
- Accesso remoto e gestione centralizzata;

- **PaaS (Platform as a Service)**: fornisce **ambiente di sviluppo e distribuzione software** (database, runtime, strumenti)

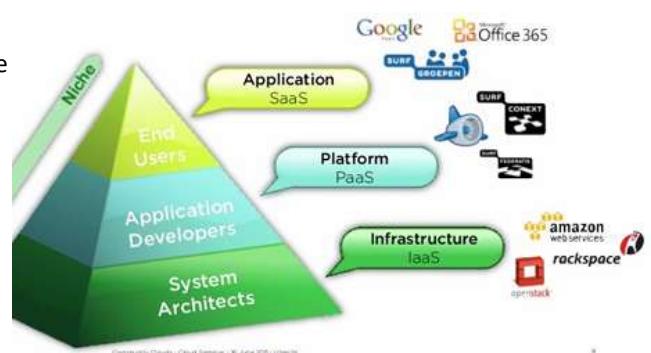
#### Vantaggi:

- Nessuna gestione dell'infrastruttura;
- Sicurezza dei dati (backup, protezione, controllo);
- Semplifica lo sviluppo e il test delle applicazioni;

- **SaaS (Software as a Service)**: software accessibile via web **senza installazione** (es. Google Workspace, Dropbox)

#### Vantaggi:

- Aggiornamenti automatici e sincronizzati per tutti gli utenti;



- Nessuna manutenzione o gestione tecnica;
- Accesso facile e test rapido di nuove soluzioni;

## CLOUD 4 IOT:

Il cloud consente di **processare dati in tempo reale** su grandi infrastrutture accessibili via Internet. Funziona come una **piattaforma di supporto** per:

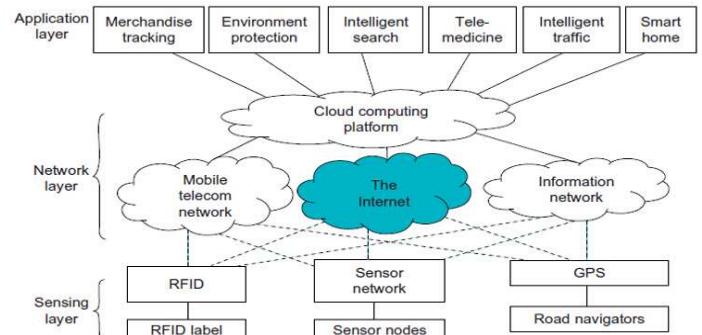
- L'invio dei dati dai dispositivi smart;
- L'elaborazione di big data;
- La visualizzazione e gestione dei risultati da parte degli utenti finali.

Il Cloud rende l'IoT **più scalabile, collaborativo e accessibile**. Le funzionalità offerte dalle piattaforme cloud per l'IoT sono:

- Supporto a **gateway** tra reti locali e Internet;
- Scoperta, configurazione e attivazione di servizi;
- Gestione proattiva e reattiva della piattaforma;
- Monitoraggio, contabilità e fatturazione;
- Supporto ai protocolli standard e interfacce utente.

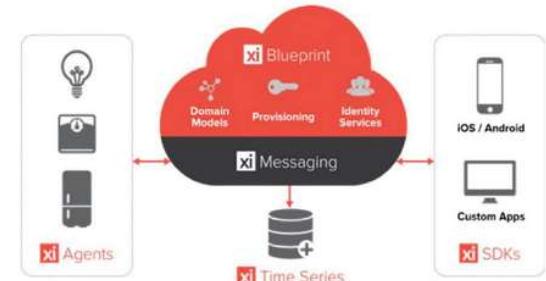
Il cloud presenta diverse sfide nell'integrazione con l'IoT:

- Difficoltà di **sincronizzazione tra diversi fornitori cloud**;
- Mancanza di **standardizzazione** tra servizi cloud;
- Differenze strutturali tra i requisiti dell'IoT e l'infrastruttura cloud;
- **Sicurezza e validazione dei servizi** non sempre uniformi;
- Gestione complessa di risorse e componenti eterogenei.



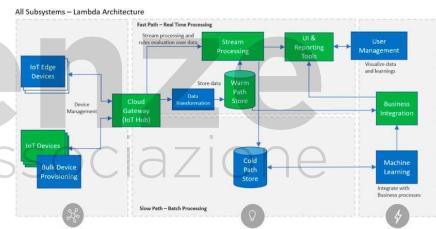
## XIVELY:

- Una delle prime piattaforme PaaS per l'IoT;.
- Espone API RESTful ed è compatibile con protocolli come HTTP, WebSockets, MQTT;
- Punti di forza:
  - Open source, semplice e interoperabile;
  - Visualizzazione dei dati in tempo reale;
  - Ampio supporto hardware (Arduino, mBed, ecc.).



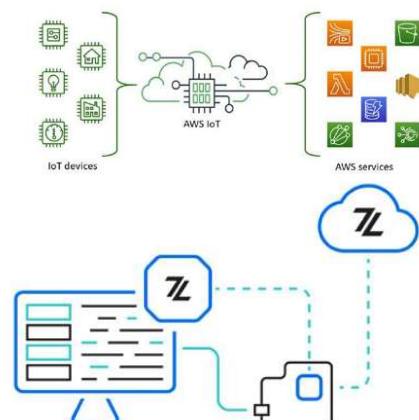
## MICROSOFT AZURE IOT SUITE:

- Include numerosi servizi per creare, gestire e analizzare soluzioni IoT:
  - **IoT Hub**: per connessioni e gestione dispositivi;
  - **IoT Central**: SaaS per configurazioni rapide;
  - **IoT Edge**: per analisi locali sui dispositivi;
  - **Digital Twins**: per modellare ambienti fisici;
  - **Time Series Insights**: per analisi dati temporali;
  - **Azure Maps**: per informazioni geografiche.



## AWS IOT (AMAZON WEB SERVICES):

- Facilita la **raccolta, analisi e gestione dei dati** da dispositivi IoT;
- Supporta MQTT e fornisce un SDK per connettere dispositivi in modo sicuro, anche **offline**;



## ZERYNTH:

- Piattaforma italiana per **sviluppo e gestione IoT industriale**;
- Componenti principali:
  - SDK multiplataforma per Python/C;
  - Sistema operativo real-time per microcontrollori;
  - Zerynth Device Manager (ZDM) per gestire e scalare soluzioni IoT.
- Si integra facilmente con i **principali provider cloud** (AWS, Azure, Google Cloud);

## 3.1. SECURITY & PRIVACY

### INTRODUZIONE:

I **sistemi IoT gestiscono grandi volumi di dati**, anche sensibili, utilizzati in ambiti critici come industria e sanità. Questa centralità li rende un obiettivo interessante per attacchi informatici, mirati a rubare dati compromettere i dispositivi e spiare utenti o condurre attacchi su larga scala. Con la diffusione dell'IoT in ogni ambito, **gli attacchi aumentano** nel tempo. I requisiti di sicurezza sono:

- **Confidenzialità**: impedire accessi non autorizzati ai dati;
- **Integrità**: assicurarsi che dati e comandi non siano stati alterati;
- **Disponibilità**: garantire che i servizi e dispositivi siano sempre operativi;
- **Responsabilità (Accountability)**: ritenere l'utente responsabile delle proprie azioni;
- **Auditabilità o Verificabilità**: monitoraggio delle azioni (monitoraggio continuo);
- **Affidabilità (Trustworthiness)**: verificare l'identità e la fiducia nella terza parte;
- **Non ripudio**: le azioni non possono essere negate;
- **Privacy**: rispetto delle politiche sui dati personali;

Un **dispositivo sicuro** (secure thing) è tale quando soddisfa tutti questi criteri.

### CONCETTI DI RISCHIO E CONTROMISURE:

- **Vulnerabilità**: debolezza del sistema (es. codice insicuro, configurazioni errate);
- **Attacco**: azione che sfrutta una vulnerabilità;
- **Exploit**: strumento o codice che realizza l'attacco;
- **Minaccia**: evento potenzialmente dannoso (anche non volontario);
- **Contromisura**: azione o tecnica che riduce la probabilità o l'impatto di un attacco.

### DIFFERENZA TRA PRIVACY E SICUREZZA:

- **Privacy**: diritto alla riservatezza (es. usare tende in una finestra);
- **Sicurezza**: protezione fisica o tecnica (es. mettere delle grate).

I due concetti sono correlati ma non equivalenti.

### GDPR (UE 2016/679) – REGOLAMENTO EUROPEO:

Il **Regolamento Generale sulla Protezione dei Dati (GDPR)**, in vigore nell'Unione Europea, stabilisce le regole per garantire la **tutela dei dati personali** degli individui.

Nel contesto dell'IoT, dove numerosi dispositivi raccolgono e trasmettono informazioni sensibili, il GDPR impone che i dati siano gestiti nel rispetto dei seguenti principi:

- **Legalità dei dati conservati**: dati trattati in modo lecito, corretto e trasparente;
- **Finalità limitate**: raccolta dei dati per scopi specifici;
- **Minimizzazione**: raccolta di dati solo strettamente necessari;
- **Accuratezza e aggiornamento dei dati**: dati accurati e aggiornati;
- **Limitazione della conservazione**: dati conservati solo per il tempo necessario;
- **Integrità e riservatezza**: protezione tecnica e organizzativa

### PRIVACY BY DESIGN VS PRIVACY BY DEFAULT:

- **Privacy by Design**: la protezione dei dati deve essere **integrata nella tecnologia fin dalla progettazione**;
- **Privacy by Default**: le impostazioni predefinite devono **limitare la raccolta e l'uso dei dati personali** al minimo necessario;

Esempi di strumenti:

- **Pseudonimizzazione**;
- **Minimizzazione**;
- **Salvaguardie tecniche e organizzative**;

## 3.2. VULNERABILITÀ E ATTACCHI NELL'IOT

I dispositivi IoT, spesso poco protetti e con risorse limitate, **sono vulnerabili a molteplici minacce** che possono compromettere sicurezza, integrità e disponibilità dei dati. Le principali vulnerabilità sono:

- **Sicurezza fisica debole**: accesso fisico non autorizzato che permette manomissione, clonazione o attacchi hardware (es. *Hardware Trojan*, tampering);
- **Autenticazione debole**: mancanza di meccanismi robusti a causa di limiti energetici/computazionali;
- **Crittografia inefficace**: algoritmi leggeri ma meno sicuri, facilmente aggirabili da attaccanti;
- **Porte aperte inutili e accessi predefiniti**: favoriscono intrusioni tramite credenziali deboli o di default;
- **Patch e aggiornamenti assenti**: firmware obsoleti e privi di protezione;
- **Cattive pratiche di programmazione**: backdoor, root user attivo, mancanza di TLS/SSL;
- **Assenza di audit**: senza logging è difficile individuare e tracciare comportamenti malevoli;
- **Raccolta di energia insufficiente**: drenare l'energia immagazzinata;
- **Controllo degli accessi insufficiente**: non richiedere la modifica delle credenziali utente.

## TIPOLOGIE DI ATTACCHI:

- **Fisici**: accesso diretto ai dispositivi per danneggiarli o modificarli;
- **Di rete**: intercettazioni, spoofing, attacchi man-in-the-middle;
- **Software**: sfruttamento di bug, malware, zero-day;
- **Sui dati**: furto, manipolazione o distruzione delle informazioni raccolte

## TASSONOMIA:

- **Livelli colpiti**: dispositivo, rete, software;
- **Obiettivi compromessi**: riservatezza, integrità, disponibilità, tracciabilità;
- **Contromisure**: autenticazione sicura, protocolli crittografici, patch management, IDS, honeypot, discovery tools.

## ATTACCHI CONTRO L'IOT:

I sistemi IoT sono vulnerabili a diversi tipi di attacchi che mirano a compromettere **riservatezza, integrità, disponibilità e autenticazione** dei dati e dei dispositivi:

- **Attacchi alla Riservatezza e Autenticazione**:
  - **Brute force & dictionary attack**: accesso non autorizzato tramite tentativi sistematici di credenziali (es. malware Mirai);
  - **Spoofing & eavesdropping**: intercettazione di dati o falsificazione dell'identità dei dispositivi;
  - **Side-channel attack**: recupero di chiavi crittografiche da misure fisiche (es. consumo di energia);
  - **Sybil attack**: creazione di identità false per manipolare reti P2P o sistemi di reputazione.
- **Attacchi all'Integrità dei Dati**:
  - **False Data Injection (FDI)**: iniezione di dati corrotti o falsi per alterare i risultati dei sensori;
  - **Modifica firmware**: alterazione del software interno dei dispositivi per comprometterne il funzionamento.
- **Attacchi alla Disponibilità**:
  - **Denial of Service (DoS)**: esaurimento delle risorse per bloccare l'accesso legittimo;
  - **Device capture**: cattura fisica del dispositivo per estrarre dati sensibili;
  - **Sinkhole & Wormhole**: deviazione del traffico verso nodi malevoli che possono alterare o perdere pacchetti.
- **Attacchi all'Energia**:
  - **Vampire attack**: invio di messaggi che consumano energia in modo anomalo:
    - **Carousel**: cicli infiniti sullo stesso percorso;
    - **Stretch**: percorsi volutamente allungati;
  - **Perdita**: Non vengono effettuate ottimizzazioni per non sprecare la batteria del dispositivo;
  - **Sleep deprivation**: i dispositivi sono forzati a rimanere attivi, riducendo la durata della batteria;
  - **Jamming**: interferenze radio che impediscono la comunicazione e causano collisioni di pacchetti.

### 3.3. PROBLEMI DI PRIVACY

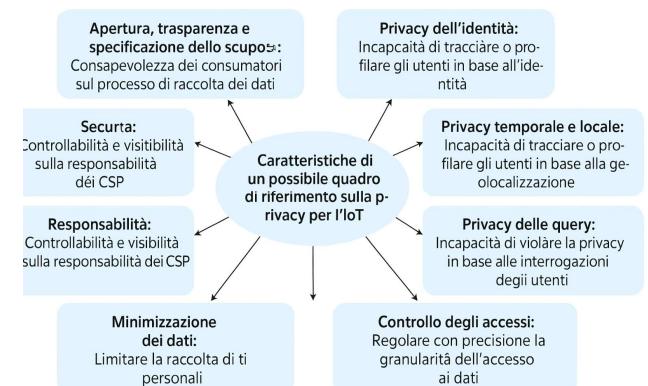
I dispositivi IoT raccolgono e trasmettono enormi quantità di dati, spesso sensibili (es. posizione, abitudini, identità), esponendo gli utenti a **rischi di tracciamento, sorveglianza e violazione della privacy**.

Le principali problematiche sono:

- **Profilazione e localizzazione**: i dispositivi connessi possono rivelare informazioni personali, creando profili dettagliati degli utenti senza consenso;
- **Dati sensibili archiviati a lungo termine**: la memorizzazione illimitata può causare violazioni, mancando il principio della "dimenticanza digitale";
- **Uso secondario dei dati**: le informazioni raccolte possono essere usate per scopi diversi da quelli dichiarati.
- **Anonimato difficile**: tecnologie come RFID/NFC e WSN sono vulnerabili al tracciamento se non adeguatamente protette.
- **Limitazioni tecniche**: dispositivi con risorse limitate non possono usare tecniche avanzate di cifratura, rendendo difficile applicare le protezioni tradizionali.
- **Cloud computing**: gli utenti perdono controllo su dati e identità quando affidati a fornitori terzi (CSP), con rischi di lock-in e poca trasparenza.

Altri aspetti rilevanti:

- Necessità di scalabilità, elaborazione distribuita e analisi real-time compatibili con la privacy.
- **Privacy by design & compliance**: è essenziale rispettare i principi di legalità, equità, proporzionalità, specificità degli scopi e responsabilità (conformità legale e sociale)



## 3.4. ANALISI FORENSE DELL'IOT

### INTRODUZIONE:

Con la diffusione massiccia dei dispositivi IoT, che raccolgono ed elaborano dati personali sensibili, emergono importanti sfide legate alla **privacy** e alla **sicurezza**. Se da un lato questi dispositivi possono migliorare i servizi quotidiani, dall'altro rappresentano una potenziale minaccia in caso di violazione o uso improprio dei dati.

La **Digital Forensics (DF)** è la disciplina che si occupa della raccolta, analisi e conservazione delle prove digitali. L'**IoT Forensics** ne è una sottobranca, applicata agli ambienti connessi, e ha l'obiettivo di estrarre informazioni rilevanti in modo **scientifico** e **legalmente valido** da dispositivi, reti e cloud.

Rispetto alla DF tradizionale, l'IoT Forensics affronta sfide specifiche:

- **Diversità delle fonti di prova**, che vanno dai dispositivi medici ai sistemi di bordo dei veicoli;
- **Difficoltà nella protezione dell'intera rete IoT**, data la sua complessità;
- Necessità di **determinare la causa di un incidente** (e non solo contenerlo), soprattutto in caso di minacce cyberfisiche.

Le **tracce digitali** raccolte possono contribuire a ricostruire eventi, identificare colpevoli e supportare procedimenti giudiziari (es. registrazione di un allarme o dati di un sensore ambientale).

Infine, il crescente ruolo del cloud ha introdotto il paradigma **Forensics-as-a-Service (FaaS)**, che permette di estendere le capacità investigative tramite piattaforme remote. Tuttavia, la varietà di protocolli e dispositivi rende complessa la standardizzazione della sicurezza nell'ecosistema IoT.

### SFIDE:

Il campo dell'IoT Forensics presenta numerose **sfide tecniche, legali ed etiche**, classificate in più ambiti:

#### ▪ **Identificazione e Raccolta delle Prove:**

- **Difficile localizzazione dei dati**, spesso distribuiti tra dispositivi, reti e cloud;
  - **Proliferazione dei dispositivi e grande quantità di dati** da analizzare rendono complessa la selezione delle prove;
  - **Dispositivi eterogenei** e non sempre rilevabili (es. elettrodomestici smart);
  - **Assenza di linee guida standardizzate** per la raccolta forense da dispositivi IoT.
- **Formazione, Privacy ed Etica:**
- **Scarsa preparazione delle forze dell'ordine** nel trattare i dispositivi IoT in modo forense;
  - **Uso crescente della crittografia** che limita l'accesso ai dati;
  - **Problemi di privacy** nella raccolta di dati sensibili non sempre pertinenti all'indagine.

#### ▪ **Analisi, Conservazione e Affidabilità delle Prove:**

- **Dati facilmente sovrascrivibili** o manipolabili;
- **Mancanza di una "catena di custodia" chiara**, soprattutto con dati distribuiti nel cloud;
- **Assenza di metadati** rende difficile la correlazione temporale e logica delle prove.

#### ▪ **Sfide Legate al Cloud:**

- **Modelli di servizio diversi** (IaaS, PaaS, SaaS) complicano l'ottenimento delle prove;
- **Mancanza di trasparenza** nei servizi cloud e gestione differente a seconda delle giurisdizioni;
- **Dati replicati e accessibili in più paesi**, ma soggetti a leggi locali spesso contrastanti.

#### ▪ **Attribuzione e Presentazione delle Prove:**

- **Difficoltà nell'identificare gli utenti reali** (anonimato, condivisione dei dispositivi);
- **Mancanza di trasparenza** nei servizi cloud e gestione differente a seconda delle giurisdizioni;
- **Sfide nel presentare le prove in tribunale**, con giurie spesso non esperte di tecnologia.

### APPROCCI FORENSI IOT:

Le tecniche tradizionali di Digital Forensics (DF) risultano **poco efficaci** in ambito IoT.

Per questo, sono stati proposti **nuovi modelli e approcci**, tra cui:

#### ▪ **Modello delle 3 Zone:**

- **Zona 1 (interna)**: dispositivi e reti locali (raccolta iniziale delle prove);
- **Zona 2 (intermedia)**: infrastrutture pubbliche (firewall, IDS, ecc.);
- **Zona 3 (esterna)**: cloud, social, ISP, dispositivi edge, reti esterne.

#### ▪ **Altri modelli teorici:**

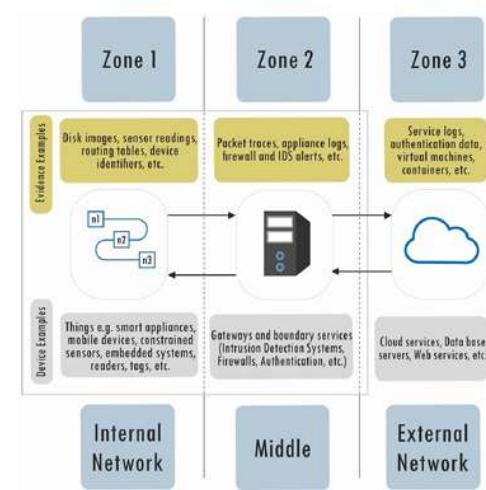
- **NBT Triage**: utile per dispositivi non più disponibili;
- **DFIF-IoT**: framework standard conforme alla ISO/IEC 27043;
- **Application-Specific Forensics**: combina processi forensi e di sicurezza;

#### ▪ **Digital Forensics Readiness (DFR)**: Promuove un approccio proattivo, raccogliendo e organizzando le prove prima che avvenga un incidente.

#### ▪ **Contrasto alle tecniche anti-forensi**: I criminali usano crittografia e offuscamento avanzati. È necessaria la **standardizzazione** e il miglioramento degli strumenti forensi IoT.

#### ▪ **Sfide giuridiche e organizzative**:

- Problemi di **giurisdizione** (dispositivo, utente o cloud?);
- **Scarsa collaborazione** tra forze dell'ordine, incident responders e laboratori;
- Si propone la **codifica e l'automazione** del processo investigativo per renderlo più accessibile.



- **Esempio: Hansken (NL):**

- Sistema olandese che fornisce **Forensics-as-a-Service.**;
- Digitalizza, centralizza e rende disponibili da remoto le tracce digitali;
- Offre interfacce diverse per investigatori tattici e tecnici.



CoScienze  
Associazione

## 4.1. STANDARD DI SICUREZZA E NORMATIVA SULL'IOT

### INTRODUZIONE:

Nel contesto della crescente diffusione dell'Internet of Things (IoT), le norme e i regolamenti svolgono un ruolo cruciale **per garantire la sicurezza e la privacy degli utenti e delle organizzazioni**. Implementare un approccio metodologico alla valutazione dei rischi è essenziale per adattarsi a contesti diversi in termini di dimensione, complessità e valore aziendale. Le valutazioni devono tener conto anche della presenza di **sistemi di governance IT**, controllo interno, e delle potenziali minacce informatiche.

È importante:

- Analizzare i rischi con metodi qualitativi e adottare controlli proporzionali;
- Utilizzare checklist di conformità in materia di sicurezza e privacy, utili per i DPO (Data Protection Officer);
- Scegliere dispositivi IoT affidabili che soddisfino criteri di sicurezza e protezione dei dati.

Inoltre, è essenziale acquisire consapevolezza dell'impatto del digitale sull'economia. La trasformazione digitale, infatti, richiede competenze adeguate e una cultura della sicurezza. Il Rapporto Clusit 2023 ha evidenziato un aumento del 169% degli attacchi informatici in Italia nel 2022, con impatti rilevanti nei settori manifatturiero, tecnico-scientifico e nei servizi professionali.

Per contrastare queste minacce, le organizzazioni devono:

- Mantenere i sistemi aggiornati;
- Usare canali e archiviazioni sicuri;
- Limitare le superfici di attacco;
- Predisporre efficaci piani di risposta agli incidenti

### ISO/IEC 27400:2022 – LINEE GUIDA SU SICUREZZA E PRIVACY NEI DISPOSITIVI IOT:

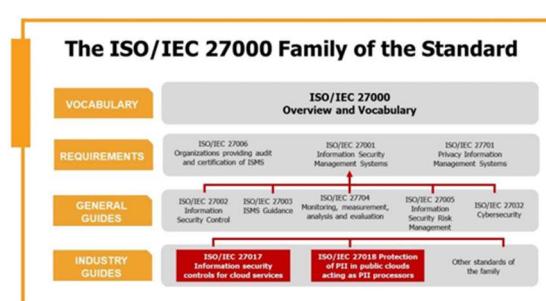
Pubblicata nel giugno 2022, la ISO/IEC 27400 rappresenta una tappa importante nell'evoluzione della normativa sulla sicurezza informatica. Si integra nella famiglia ISO/IEC 27000 ed è complementare alla norma ISO/IEC 30141:2018, che fornisce un'architettura di riferimento per i sistemi IoT.

L'IoT viene descritto come un'infrastruttura formata da entità fisiche e digitali, interconnesse e capaci di reagire agli stimoli provenienti sia dal mondo fisico che da quello virtuale, influenzando le attività reali.

Un dispositivo IoT è un'entità che comunica col mondo fisico tramite rilevamento o attuazione.

Caratteristiche principali dei dispositivi e sistemi IoT:

- Includono componenti hardware e software combinati o integrati con elementi fisici;
- Comunicano attraverso reti cablate o wireless;
- Possiedono capacità di rilevamento (sensoristica) e/o attuazione;
- Utilizzano applicazioni per elaborare dati, controllare dispositivi e integrarsi con altri sistemi;
- Hanno componenti operativi per configurazione e funzionamento;
- Possono essere utilizzati da utenti umani o digitali



### STAKEHOLDER E DOMINI FUNZIONALI DELL'IOT:

La ISO/IEC 27400 identifica diversi stakeholder con responsabilità specifiche:

- ISP (IoT Service Provider)**: gestisce i servizi che supportano il sistema IoT;
- ISD (IoT Service Developer)**: sviluppa, progetta e integra i servizi IoT;
- IU (IoT User)**: utilizzatore umano o digitale dei sistemi IoT;
- IDD (IoT Device Developer)**: figura specializzata nello sviluppo dei dispositivi.

La norma distingue sei domini funzionali dell'ecosistema IoT:

- User Domain (UD)**: comprende gli utenti e le attività di configurazione e utilizzo;
- Physical Entity Domain (PED)**: rappresenta le entità fisiche coinvolte;
- Sensing and Control Domain (SCD)**: include i sensori, attuatori, dispositivi e reti di comunicazione;
- Operations and Management Domain (OMD)**: copre le funzioni gestionali e operative, incluso il supporto tecnico;
- Resource Access and Interchange Domain (RAID)**: si occupa dell'accesso esterno e dell'infrastruttura di rete;
- Application and Service Domain (ASD)**: raggruppa le applicazioni e i servizi offerti agli utenti.

### CONTROLLI DI SICUREZZA E PRIVACY:

La norma ISO/IEC 27400 introduce 45 controlli specifici che coprono l'intero ciclo di vita dei sistemi IoT. Questi controlli non sono collegati rigidamente a singole fonti di rischio, ma sono associati a stakeholder e domini specifici, con l'obiettivo di rendere la loro implementazione più mirata ed efficace.

Fonti di rischio individuate:

- Attacchi intenzionali da parte di attori esterni o interni;
- Carenze nella governance e nella gestione;
- Errori umani e scarsa formazione;
- Vulnerabilità tecniche o di qualità del sistema;
- Fenomeni naturali (es. fulmini, alluvioni, terremoti);
- Uso improprio o compromissione di dispositivi e reti esterne.



## Tipologie di controlli:

- **Organizzativi**: definizione di policy, ruoli e responsabilità aziendali;
- **Generici IT**: monitoraggio, logging, gestione degli eventi di sicurezza;
- **Specifici IoT**: principi di progettazione sicura, gestione della sicurezza lungo tutto il ciclo di vita del dispositivo;

Alcune raccomandazioni operative includono:

- Utilizzo di tecnologie di rete adeguate alle esigenze di sicurezza e prestazione;
- Informazioni chiare agli utenti sui rischi d'uso;
- Adozione di principi di “*privacy by default*” e minimizzazione della raccolta dati;
- Disattivazione dei dispositivi non utilizzati per ridurre i punti di vulnerabilità.

## MODELLO DI MINACCIA PER SICUREZZA E PRIVACY:

Per supportare l'identificazione e la mitigazione delle minacce, la norma fa riferimento a due modelli fondamentali:

- **Stride** (per la sicurezza informatica): sviluppato da Microsoft,

Stride classifica le minacce in sei categorie:

- **Spoofing**;
- **Tampering** (Manomissione): modifica dei dati;
- **Repudiation** (Ripudio): impossibilità di provare un'azione compiuta;
- **Information Disclosure** (Divulgazione): accesso non autorizzato a informazioni;
- **Denial of Service** (Negazione del servizio): blocco dell'accesso legittimo alle risorse;
- **Elevation of Privilege**: ottenimento di accessi superiori non autorizzati.

È applicabile ai sistemi IoT in linea con l'architettura ISO/IEC 30141.

- **Linddun** (per la privacy): complementare a Stride, si concentra sulle minacce alla privacy suddivise in sette categorie:

- Linkability;
- Identifiability;
- Non-repudiation;
- Detectability;
- Disclosure of information;
- Unawareness;
- Non-compliance

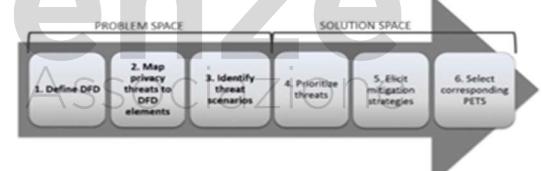
La metodologia Linddun si articola in due fasi:

- **Spazio del problema (passaggi 1-3)**: identificazione e descrizione delle minacce;
- **Spazio della soluzione (passaggi 4-6)**: definizione di misure di mitigazione e soluzioni specifiche.

Il sito ufficiale linddun.org offre un catalogo di tecniche e tecnologie di mitigazione (PETs – Privacy Enhancing Technologies), considerato come risorsa dinamica e in continuo aggiornamento. Inoltre, le strategie LINDDUN sono integrate nella norma ISO/IEC 27550, dedicata all'ingegneria della privacy nei processi di ciclo di vita dei sistemi.

	Threat	Property	Property description
IoT systems security objectives expressed as threats to counter	Spoofing	Authentication	The identity of IoT entities or IoT users is established (or you are willing to accept anonymous entities).
	Tampering	Integrity	IoT data and system resources are only changed in appropriate ways by appropriate people.
	Repudiation	Nonrepudiation	IoT users cannot perform an action and later deny performing it.
	Information disclosure	Confidentiality	Data is only available to the IoT users intended to access it.
	Denial of Service	Availability	IoT services are ready when needed and perform acceptably.
	Elevation of privilege	Authorization	IoT users are explicitly allowed or denied access to resources.

	Minaccia	Proprieta	Descrizione proprietà
Obiettivi della privacy dei sistemi IoT esprimere come da counter	Linkabilità	Non linkabilità	Nascondere il collegamento tra due o più azioni, identità e pezzi di informazioni associati a entità IoT o utenti IoT
	Identificabilità	Durezza	Nascondere il link tra un utente IoT o un'identità IoT e un'azione o un pezzo di informazione. La capacità di un utente IoT o un'entità IoT che persone nè conoscono ne confermare né contraddirlo
	Non ripudio	Negazione plausibile	Nascondere le attività di un utente IoT o un'entità IoT
	Rilevabilità	Irrilevabilità e inosservabilità	Nascondere le attività di un utente IoT o un'entità IoT
	Divulgazione di informazioni	Sicurezza Riservatezza	La capacità del sistema IoT di nascondere il contenuto dei dati o controllare il rilascio del contenuto dei dati
	Inconsapevolezza	Consapevolezza del contenuto	La consapevolezza dell'utente IoT sui propri dati
	Non conformità	Privacy Soft Politica e conformità ai consensi	IoT stakeholder IoT che è un controllore IP per informare l'utente IoT che è un principale IP sulla politica di privacy IoT, o per consentire all'utente IoT di specificare consensi in conformità con la



## LEGISLAZIONE EU:

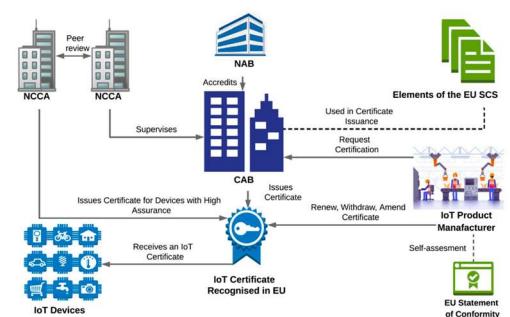
Entrato in vigore il 27 giugno 2019, il **CyberSecurity Act (CSA)** ha rafforzato il ruolo di ENISA, l'Agenzia dell'UE per la sicurezza informatica, e ha introdotto un quadro europeo di certificazione della sicurezza informatica per prodotti, servizi e processi digitali, compresi quelli legati all'IoT.

- La certificazione è attualmente volontaria, ma la Commissione Europea sta valutando la possibilità di renderla obbligatoria per alcune categorie ad alto rischio;
- Il sistema si basa su standard internazionali come ISO/IEC 15408 (criteri comuni) e ISO/IEC 18045 (metodologia di valutazione della sicurezza);
- I prodotti/servizi vengono classificati in tre livelli di garanzia (art. 52 CSA), in base a casi d'uso e capacità tecnologiche.

Gli attori coinvolti nel processo di certificazione sono:

- **CAB (Conformity Assessment Body)**: ente terzo e indipendente che conduce la valutazione;
- **NAB (National Accreditation Body)**: accredita i CAB in base ai requisiti richiesti;
- **Produttori/fornitori**: possono autovalutarsi per il livello base e devono emettere una dichiarazione di conformità da inviare a ENISA e alla NCCA;
- **NCCA (National Cybersecurity Certification Authority)**: supervisiona l'intero processo di certificazione, valida enti di valutazione, gestisce reclami e collabora con le autorità nazionali;

Livelli di garanzia	Descrizione delle attività di valutazione
Basic	Le attività di valutazione devono contenere almeno una revisione della documentazione tecnica.
Substantial	Le attività di valutazione devono contenere: (i) una revisione per confermare l'assenza di vulnerabilità note e (ii) test che validano la corretta implementazione delle funzionalità di sicurezza richieste.
Averanza	Le attività di valutazione devono contenere (i) una revisione per convalidare l'assenza di vulnerabilità note, (ii) test per garantire che le funzionalità di sicurezza richieste siano correttamente applicate e (iii) test di penetrazione per valutarne la resistenza.



## DIRETTIVA NIS E NIS2:

Con l'aumento delle minacce informatiche e la crescente digitalizzazione di servizi essenziali, ***l'Unione Europea ha adottato misure normative specifiche*** per rafforzare la resilienza dei sistemi informativi. In questo contesto, la Direttiva NIS e il suo aggiornamento NIS2 rappresentano i principali strumenti legislativi europei volti a migliorare la sicurezza informatica nei settori critici. Di seguito, vengono presentati i punti salienti di entrambe le direttive:

### ▪ **Direttiva NIS (2016/1148):**

- Primo quadro normativo europeo per migliorare la sicurezza informatica;
- Sebbene abbia aumentato le capacità degli Stati membri, ha mostrato limiti dovuti alla frammentazione delle implementazioni.

### ▪ **Direttiva NIS2 (UE 2022/2555):** In vigore dal 16 gennaio 2023, rafforza e aggiorna la precedente:

- Introduce la rete di gestione delle crisi CYCLONE;
- Aumenta l'armonizzazione dei requisiti di sicurezza e degli obblighi di notifica;
- Estende l'ambito di applicazione a più settori dell'economia e società;
- Richiede attenzione alla catena di fornitura, alla gestione delle vulnerabilità, all'igiene informatica e alla sicurezza di base di Internet.
- ENISA ha mappato l'intero ciclo di vita della supply chain IoT

Anche se non affronta direttamente l'IoT, NIS2 introduce elementi utili alla sicurezza di tali sistemi, come:

- Procedure organizzative per la gestione delle vulnerabilità;
- Requisiti di sicurezza nei rapporti tra fornitori;
- Incentivi a migliorare la qualità dei prodotti ICT.

## CYBER RESILIENCE ACT (CRA):

Previsto in vigore nel 2024, il CRA rappresenta un passo decisivo verso una normativa vincolante per tutti i prodotti con elementi digitali.

Principali requisiti:

- I prodotti potranno essere immessi sul mercato solo se soddisfano livelli adeguati di sicurezza rispetto ai rischi;
- Obbligo di notifica degli incidenti entro 24 ore;
- Valutazione di conformità obbligatoria per prodotti e processi, incluse le pratiche di gestione delle vulnerabilità;
- Gli Stati membri possono nominare autorità nazionali per supervisionare l'applicazione del CRA.

CRA opera in sinergia con NIS2 e CSA, creando un ecosistema normativo coerente e orientato alla resilienza informatica dei prodotti digitali.

## DIRETTIVA RED (2014/53/UE) E REGOLAMENTO DELEGATO 2022/30:

La Direttiva RED (e il relativo regolamento 2022/30) disciplina le apparecchiature radio, inclusi i dispositivi IoT che comunicano tramite tecnologie come Wi-Fi e Bluetooth.

Definisce requisiti essenziali su:

- Sicurezza e salute;
- Compatibilità elettromagnetica;
- Uso efficiente dello spettro radio;
- Protezione della privacy, dei dati personali e contro le frodi;
- Integra anche interoperabilità, accesso ai servizi d'emergenza e conformità di software combinato con dispositivi radio.

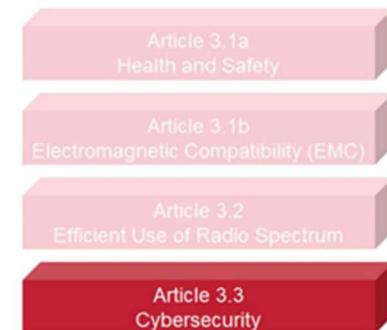
## OBBLIGHI SPECIFICI PER I PRODUTTORI (ARTICOLO 3.3 RED):

I produttori dovranno garantire che i dispositivi:

- Non danneggino le reti o interrompano servizi web (Art. 3.3d);
- Impediscano l'accesso non autorizzato o la trasmissione non voluta di dati personali (Art. 3.3e);
- Prevengano pagamenti elettronici fraudolenti e trasferimenti monetari sospetti (Art. 3.3f);

Queste misure devono considerare le nuove minacce nel settore digitale, come:

- Cryptojacking;
- Ransomware;
- Frodi tramite comunicazioni a corto raggio (NFC);
- Manomissione dei sistemi biometrici;



## 4.2. DESIGN SICURO E SVILUPPO

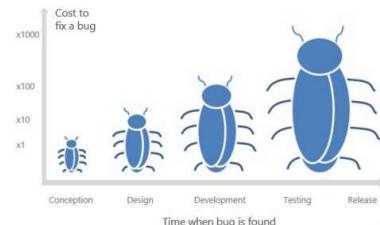
### INTRODUZIONE:

La sicurezza dei prodotti è un elemento centrale nello sviluppo e nella manutenzione dei sistemi IoT, sia nel mercato industriale sia in quello consumer. Per garantire un'infrastruttura IoT solida e affidabile, è fondamentale che i dispositivi siano intrinsecamente sicuri fin dalla fase di progettazione e produzione.

Il processo di sviluppo dovrebbe partire da una politica di sicurezza ben definita, che funga da base per rispettare le normative applicabili alla sicurezza informatica. Solo così si possono realizzare prodotti e servizi digitali conformi e protetti.

Il momento in cui viene scoperta una vulnerabilità ha un impatto diretto sui costi:

- Se il bug viene rilevato in fase di progettazione, è facile da correggere e può essere integrata una misura preventiva con costi contenuti;
- Se viene scoperto dopo il rilascio in produzione, la correzione diventa molto più onerosa, sia in termini economici (circa quattro volte superiore) sia in termini di danni reputazionali e perdita di fiducia da parte dei clienti;



## DIFESA IN PROFONDITÀ E SDLC (SECURE DEVELOPMENT LIFE CYCLE):

La garanzia della sicurezza per le applicazioni IoT può essere ottenuta al meglio adottando una strategia di difesa approfondita, che a sua volta richiede l'adozione di una pratica di Secure Development Life Cycle (SDLC).

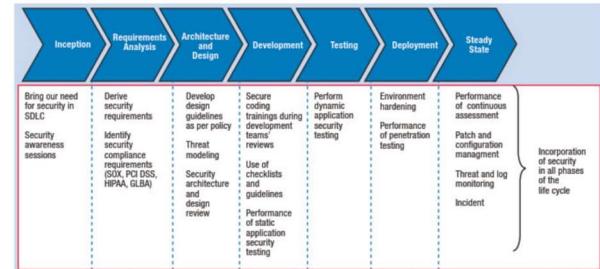
L'intento principale è quello di creare sicurezza all'interno del ciclo di vita di queste applicazioni partendo da zero, riducendo potenzialmente e gradualmente i difetti di sicurezza, progettazione, implementazione e distribuzione.

La corretta aderenza a tali best practice di garanzia si tradurrà in applicazioni prive di vulnerabilità che potrebbero essere state introdotte accidentalmente o intenzionalmente in qualsiasi momento del loro ciclo di vita.

### SICUREZZA NELL'SDLC:

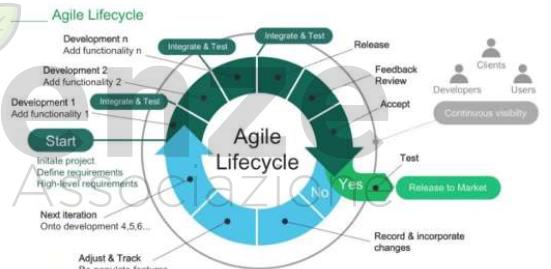
La sicurezza nelle soluzioni IoT deve essere considerata fin dall'inizio del ciclo di vita dello sviluppo software (SDLC).

- **Injection (fase iniziale):** è fondamentale riconoscere l'importanza dell'integrazione della sicurezza e sensibilizzare il team attraverso apposite sessioni formative;
- **Analisi dei requisiti:** durante questa fase vengono identificati i bisogni di sicurezza specifici e le normative di conformità da rispettare, come PCI DSS, HIPAA o SOX, così da costruire una base chiara di ciò che dovrà essere protetto;
- **Progettazione e architettura:** si applicano tecniche di threat modeling per analizzare i possibili scenari di attacco contro ciascun componente IoT. Questo consente di individuare i potenziali attori delle minacce e proporre le relative contromisure, che saranno incorporate nella progettazione stessa del sistema;
- **Sviluppo:** è essenziale scrivere codice conforme a standard sicuri ampiamente accettati, come quelli del CERT, per evitare vulnerabilità comuni. Si utilizzano checklist, revisioni tra pari e test statici per garantire che il codice prodotto sia robusto dal punto di vista della sicurezza;
- **Test:** prevede invece l'esecuzione di verifiche dinamiche sulla sicurezza delle applicazioni. Il tipo di valutazione da eseguire dipende dalla natura dell'applicazione IoT e delle API coinvolte. Una volta pronto il rilascio, la distribuzione del sistema avviene dopo aver effettuato test di penetrazione standard su tutti i componenti, con un'attenzione particolare all'infrastruttura e alla capacità di reggere carichi di traffico realistici;
- **Gestione operativa (steady state):** richiede il monitoraggio costante dell'ambiente IoT tramite una soluzione centralizzata. Questa permette di effettuare audit di sicurezza regolari, valutando il livello di conformità del sistema ai criteri stabiliti e garantendo l'integrazione della sicurezza lungo l'intero ciclo di vita della soluzione.



### LE SFIDE DELLA SICUREZZA NELLO SVILUPPO AGILE:

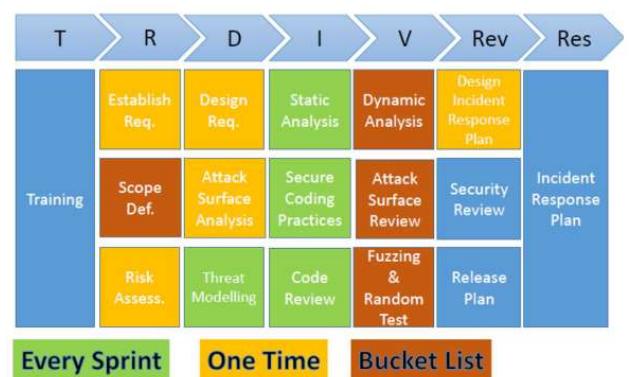
I progetti IoT spesso seguono **metodologie Agile** (Scrum, Kanban, XP), che favoriscono cicli brevi e iterativi. Tuttavia, l'integrazione della sicurezza è complessa, poiché i cicli di sviluppo sono brevi e i requisiti difficili da formalizzare nelle user story (l'unità base della progettazione Agile).



### REQUISITI DI SICUREZZA:

Per affrontare le criticità della metodologia Agile, è utile classificare i requisiti di sicurezza in tre categorie, in base alla loro frequenza d'esecuzione, e non alla fase del progetto:

- **Requisiti critici (Every Sprint):**
  - Obbligatori per ogni rilascio;
  - Se non implementati, il rilascio è considerato incompleto;
  - Riguardano aspetti fondamentali di protezione;
- **Requisiti una tantum (One Time):**
  - Da eseguire una sola volta durante il ciclo di vita del progetto;
  - Tipicamente completati nelle fasi iniziali, ma con possibilità di tolleranza temporale;
  - Ad esempio: definizione delle policy di sicurezza, scelta delle librerie sicure;
- **Requisiti ricorrenti (Bucket List):**
  - Esecuzione regolare nel tempo, senza ordine preciso;
  - Comprendono attività come test di vulnerabilità, aggiornamento continuo delle dipendenze, verifica delle configurazioni di sicurezza;
  - Sono parte delle pratiche di sicurezza di mantenimento nel lungo periodo;



### VALUTAZIONE DELLE CRITICITÀ:

Nel contesto della sicurezza informatica, un vettore di attacco rappresenta la sequenza di azioni che un avversario (attaccante) può intraprendere per raggiungere un obiettivo dannoso. Spesso, il **dispositivo IoT è il punto d'ingresso preferito** poiché rappresenta l'anello più debole della catena di sicurezza.

L'attaccante sfrutta vulnerabilità nei dispositivi IoT, come assenza di meccanismi di protezione o difetti nei protocolli di rete, per raggiungere

sistemi critici collegati, anche se tali connessioni non sono immediatamente visibili o dirette.

(Primo pezzo)

Le modalità di accesso al dispositivo sono (**Access to IoT**):

- **Accesso fisico:**

- **Insider:** ha accesso diretto o fisico al dispositivo (es. tecnico interno)
- **Outsider:** non ha accesso diretto, ma può manipolare un dispositivo simile;

- **Accesso logico:**

- **Privilegiato:** accesso autorizzato tramite interfaccia nota;
- **Non privilegiato:** nessun accesso a priori; richiede bypass di controlli;

Gli attacchi senza bisogno di accesso privilegiato sono più probabili, perché più facili da eseguire.

Competenze, risorse e motivazione dell'avversario (**Capabilities**):

- Gli attacchi che richiedono competenze avanzate o risorse significative (strumenti costosi, infrastrutture complesse) sono meno probabili;
- Attacchi realizzabili con strumenti economici o da avversari poco esperti sono più comuni;

Motivazione (**Motivation**):

La motivazione può essere vista come un **modo alternativo per descrivere il potenziale guadagno** che un avversario trarrebbe da un attacco riuscito, in combinazione con la penalità prevista per un avversario che viene rintracciato. Gli attacchi che possono attrarre avversari con una motivazione anche debole hanno più probabilità di verificarsi. Al contrario, gli attacchi che verrebbero innescati solo da avversari fortemente motivati sono meno possibili.

(Secondo pezzo)

Poiché il dispositivo IoT è l'abilitatore/amplificatore dell'attacco, un avversario dovrà scoprire e sfruttare le vulnerabilità esistenti, raggruppate in due categorie principali:

- **Incorporate:** problemi di progettazione o implementazione a livello hardware (HW) e software (SW) (**Embedded features**);
- **Di rete:** debolezze nei protocolli di comunicazione o nei meccanismi di supporto (**Network features**);

Un avversario con sufficienti competenze cercherà di scoprire e sfruttare tali vulnerabilità per penetrare nei sistemi.

(Terzo pezzo)

Per le **connessioni dirette (Direct Connectivity)** i dispositivi IoT possono essere:

- **Fisicamente connessi** a un sistema critico (es. all'interno dello stesso edificio protetto);
- **Logicamente connessi**, anche se non visibili immediatamente nella topologia di rete;

La **prossimità fisica** o la **connettività logica indiretta** (es. tramite sistemi ausiliari o di bassa priorità) possono creare percorsi di attacco nascosti, spesso trascurati durante la valutazione dei rischi. Questi attacchi sono particolarmente pericolosi perché non identificati in fase preventiva. (**Indirect Connectivity**)

Un esempio noto è l'uso di dispositivi IoT domestici compromessi per costruire botnet (reti di dispositivi controllati) e lanciare attacchi contro infrastrutture critiche. (**No connectivity**)

Per valutare la gravità di un attacco IoT, vengono considerate tre dimensioni principali:

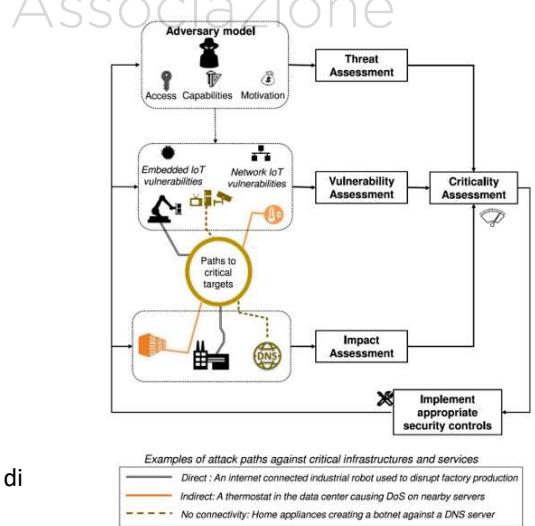
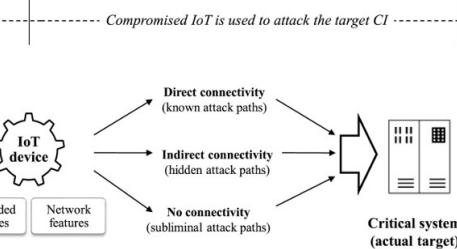
- **Livello di minaccia:** misura quanto il sistema è esposto a un determinato attacco;
- **Livello di vulnerabilità:** misura la facilità con cui un avversario può sfruttare le debolezze del sistema;
- **Livello di impatto:** rappresenta l'entità dei danni (materiali, operativi, reputazionali) che potrebbero verificarsi

La combinazione di questi tre fattori consente di stimare il livello di criticità complessivo di un attacco.

Ogni fattore di rischio (minaccia, vulnerabilità, impatto) viene misurato su una scala qualitativa a tre livelli:

- **Basso (valore 0);**
- **Medio (valore 1);**
- **Alto (valore 2);**

Questa classificazione permette di costruire matrici di rischio e prioritizzare gli interventi di mitigazione.



## 4.3. PROCESSO DI SVILUPPO DEI PRODOTTI IOT

### INTRODUZIONE:

Le agenzie governative riconoscono i rischi derivanti da dispositivi IoT non sicuri. Per questo, sono state sviluppate linee guida e standard per aiutare le aziende a integrare la sicurezza nella progettazione, implementazione e manutenzione dei prodotti IoT connessi in rete. Le best practice coinvolgono scelte riguardanti protocolli, crittografia, API, piattaforme e altro.

### REQUISITI DI SICUREZZA:

Nel ciclo di vita di un dispositivo IoT, **la fase iniziale dei requisiti rappresenta un punto critico**. È in questo momento che si definiscono:

- l'uso previsto del dispositivo;
- le sue funzionalità specifiche;

- l'ambiente operativo in cui verrà impiegato (industriale, domestico o misto).

Progettare **la sicurezza sin dall'inizio** è essenziale: trattarla come un'aggiunta successiva sarebbe un errore strategico. In questa fase si delineano:

- **requisiti legati al contesto**, influenzati da fattori come il tipo di rete e la connettività;
- **requisiti basati sulle funzionalità**, che variano in base ai casi d'uso e agli scenari aziendali.

In un dispositivo IoT, software e hardware sono strettamente interconnessi. Le decisioni prese a livello software influiscono direttamente sulla scelta dei componenti fisici.

Per assicurare coerenza e qualità lungo tutta la catena produttiva, i gateway di controllo ricevono e verificano i requisiti come input e li verificano. A supporto di queste attività, vengono usati strumenti come:

- l'analisi del rischio;
- le checklist basate su standard noti (ad esempio l'OWASP IoT Top 10);
- la documentazione sulle minacce più comuni.

che offrono una guida utile agli sviluppatori.

Un classico esempio è il termostato intelligente: connesso a Internet, controllabile da remoto, ma anche legato alla rete privata domestica e al sistema di riscaldamento. In scenari come questo, la sicurezza del dispositivo non è solo una questione di protezione digitale, ma anche di tutela della sicurezza fisica.

Per garantire la sicurezza di dispositivi come il termostato, è fondamentale che ogni unità disponga di un'identità digitale unica, che consenta:

- autenticazione sicura;
- comunicazione con altri sistemi in modo sicuro.

Poiché molti dispositivi IoT restano attivi per anni, è necessario prevedere la possibilità di aggiornamenti sicuri che tengano conto delle vulnerabilità future. A tal fine, è indispensabile:

- implementare una **Root of Trust (RoT)**, ossia un meccanismo che consente di verificare l'autenticità degli aggiornamenti ricevuti;
- adottare una politica di aggiornamento robusta che impedisca attacchi di rollback, sfruttando versioni precedenti del software ancora vulnerabili.

## ARCHITETTURA DEL PRODOTTO IOT:

**L'architettura di un dispositivo IoT** dipende principalmente dalle funzionalità locali da implementare e dal tipo di connettività prevista.

Le soluzioni più comuni adottano **microcontrollori ad alte prestazioni (MCU)** dotati di strumenti software integrati per comunicare facilmente con i servizi cloud.

Questi sistemi possono includere sottosistemi crittografici dedicati per la protezione dei dati e delle comunicazioni.

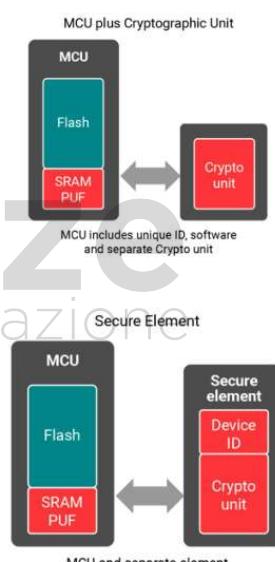
Una componente fondamentale è l'integrazione di **unità crittografiche (CU)**, che comunicano con l'MCU tramite interfacce sicure come:

- SPI;
- I<sup>2</sup>C.

utilizzando chiavi pubbliche per garantire la sicurezza delle operazioni.

Dispositivi ancora più avanzati utilizzano **Elementi Sicuri (SE)**, capaci di:

- rilevare manomissioni fisiche;
- proteggere la memoria con crittografia;
- garantire l'integrità anche nella fase di produzione.



## CONNELLIVITÀ IOT:

Anche la connettività ha un impatto sull'architettura. Alcuni componenti hardware, come gli Elementi Sicuri (SE), supportano protocolli avanzati (es. DTLS), abilitando funzionalità come l'aggiornamento sicuro del software.

## ANALISI DELLE MINACCE ALLA SICUREZZA:

Una volta definita l'architettura hardware e di connettività, è **essenziale analizzare le minacce** al prodotto IoT. Questa analisi fa parte del processo di sviluppo e include:

- **vettori di attacco** utilizzati da attori malintenzionati per sottrarre proprietà intellettuale;
- scelta di componenti hardware appropriati in base alle minacce identificate;
- valutazione del supporto fornito dagli strumenti e dalle infrastrutture di produzione sicura.

Aspetti fondamentali dell'analisi includono:

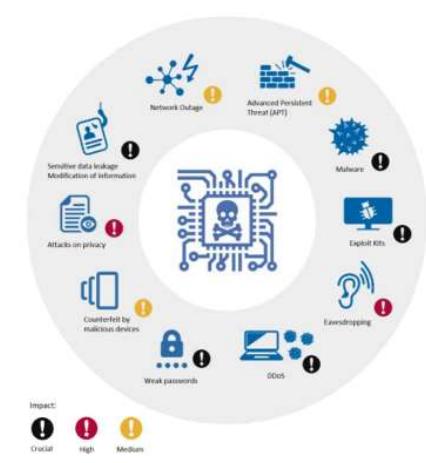
- **posizione** del dispositivo (è facilmente accessibile?);
- **accesso fisico** (la scheda è esposta?);
- **accesso elettrico** (ci sono più porte di comunicazione?).

È necessario identificare e scomporre gli **asset software** in ambienti complessi, focalizzandosi sui rischi specifici per lo sviluppo.

Le minacce possono avere **livelli di impatto variabile** e provocare **effetti a cascata** nell'infrastruttura. Gli esperti intervistati hanno confermato la differenza di impatto in base a gli scenari d'uso.

Microsoft propone il modello STRIDE, che classifica le minacce in sei categorie (come visto sopra).

Infine, l'analisi deve includere tre ambiti:



- **Personne:** sviluppatori, utenti finali e altri stakeholder;
- **Processi:** sicurezza in tutte le fasi del ciclo di sviluppo;
- **Tecnologie:** strumenti e misure tecniche per ridurre vulnerabilità.

Ogni componente del sistema IoT deve essere protetto in funzione del suo **grado di criticità**.

## STRATEGIA DI SVILUPPO:

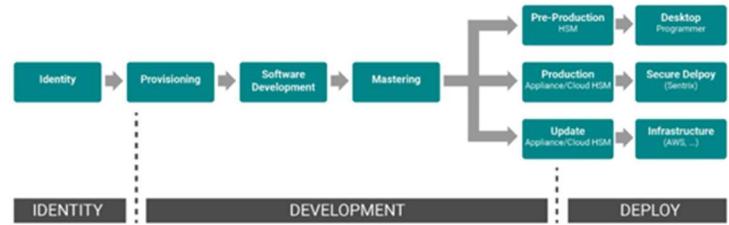
Dopo aver definito i requisiti di sicurezza e identificato le minacce, lo sviluppatore IoT deve creare un piano di implementazione della sicurezza che includa:

- Le **credenziali critiche** del prodotto;
- Il **processo di sviluppo, produzione e ciclo di vita**;

Le tre aree critiche da considerare sono:

- **Punti di fiducia nello sviluppo:** definizione dell'identità del prodotto, Root of Trust (RoT), certificati;
- **Produzione sicura:** prevenzione di compromissioni nel provisioning (es. chiavi trappolate, identità clonate);
- **Sicurezza post-produzione:** aggiornamenti software sicuri, protezione dei dati utente, mantenimento della fiducia

Le fasi chiave del flusso di sviluppo del prodotto sono:



### Identità:

- Ogni dispositivo IoT deve avere un'identità unica e forte (es. chiave privata, certificato del dispositivo, ID silicio) per garantire l'autenticazione sicura e la comunicazione protetta tra dispositivi, servizi e utenti;
- Questo elemento è cruciale per implementare una **Public Key Infrastructure (PKI)** che consenta l'autenticazione continua e sicura;



### Provisioning:

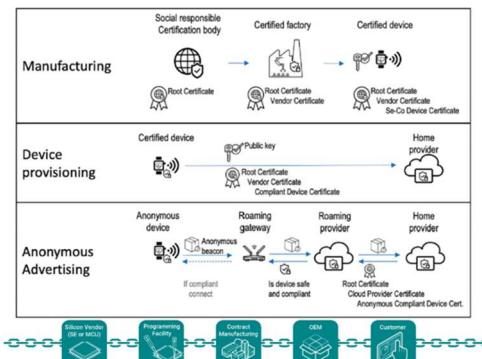
- Durante questa fase, il dispositivo IoT viene programmato con la sua identità e le credenziali di fiducia (chiavi e certificati);
- Il dispositivo è quindi pronto per essere consegnato all'utente finale o a un'applicazione, con un firmware sicuro e memorizzato in aree protette. È fondamentale che tutte le credenziali siano immesse in modo sicuro per evitare attacchi futuri;

### Mastering:

- Il software finale viene rilasciato, firmato e crittografato;
- Il pacchetto masterizzato comprende il software crittografato e firmato e un pacchetto di aggiornamento protetto, firmato con la **chiave privata** del produttore (OEM);
- Questo processo avviene in un ambiente sicuro di produzione o durante aggiornamenti post-produzione;

### Distribuzione sicura:

- Il pacchetto software masterizzato viene consegnato al dispositivo IoT in modo sicuro;
- Durante la produzione, il pacchetto viene iniettato nel dispositivo, mentre gli aggiornamenti sicuri avvengono tramite metodi come gli aggiornamenti **over-the-air (OTA)**, che assicurano che il dispositivo possa ricevere e applicare modifiche senza compromettere la sicurezza;



## IDENTITÀ, RADICE DELLA FIDUCIA, CATENA DELLA FIDUCIA:

L'identità del dispositivo (es. chiavi, certificati, ID silicio) è essenziale per l'autenticazione e la comunicazione sicura.

Una **catena di fiducia** efficace parte dal **fornitore di silicio**, passa per il **fornitore di programmazione** e i **produttori OEM**, fino al cliente finale.

Serve per abilitare una **PKI** e garantire aggiornamenti sicuri sul campo.

## PROVISIONING:

Il **provisioning** è il processo in cui una MCU sicura o un SE viene programmato con:

- **Chiavi, certificati, configurazioni** che definiscono il dispositivo IoT;
- **Firmware del cliente/OEM** (quando necessario);

Questo processo:

- Blocca il dispositivo;
- Protegge i segreti contenuti al suo interno;
- Avviene spesso **fuori dal controllo diretto dell'OEM**, quindi è necessario applicare una **strategia "zero trust"** per assicurare la catena di fornitura.

## MASTERING:

La fase di masterizzazione (mastering) assicura che il firmware:

- Sia **firmato** e **crittografato**, per prevenire furto di IP e contraffazione;
- Sia compatibile con i **bootloader sicuri** già presenti nei dispositivi.

Il risultato è un pacchetto di **file crittografati**, che:

- Possono essere decriptati e verificati **solo** da bootloader sicuri;
- Vengono distribuiti in ambienti protetti per essere caricati nel dispositivo durante la produzione o per aggiornamenti in campo.

## DISTRIBUZIONE SICURA:

Dopo lo sviluppo e il test, l'applicazione IoT deve essere distribuita:

- A una **struttura di programmazione** per la produzione in serie;
- Al **prodotto sul campo** per aggiornamenti software sicuri.

Poiché non si può presumere che il canale di distribuzione sia sicuro, si adotta una filosofia "**zero-trust**" e si impiegano strumenti di sviluppo che sfruttano l'hardware sicuro integrato nei microcontrollori moderni.

## 4.4. CODIFICA SICURA

---

### INTRODUZIONE:

Gran parte delle minacce alla sicurezza nei sistemi IoT deriva da vulnerabilità presenti nel codice. Per questo motivo, è essenziale evitare comandi non sicuri fin dalla fase di sviluppo:

- **Linguaggi come C/C++** sono comunemente usati, ma presentano rischi tipici come:
    - buffer overflow;
    - vulnerabilità degli interi;
    - vulnerabilità nelle stringhe.
  - È importante **eliminare o sostituire** funzioni pericolose con alternative sicure già nella fase di scrittura del codice;
  - Due tecniche fondamentali per proteggere il software sono:
    - **Convalida**: verifica che l'input soddisfi requisiti specifici;
    - **Sanificazione**: modifica l'input per renderlo valido e sicuro
- Combinare convalida e sanificazione consente una **difesa approfondita**, migliorando la sicurezza e la qualità del software.

### STRATEGIE DI CONVALIDA:

- **Whitelist (positiva)**: accettare solo valori esplicitamente ammessi;
- **Blacklist (negativa)**: rifiutare pattern o caratteri noti come pericolosi (approccio meno sicuro e incompleto).

### STRATEGIE DI SANIFICAZIONE:

- **Whitelist (positiva)**: rimuovere o modificare caratteri non presenti in un elenco approvato (da validare comunque dopo);
- **Blacklist (negativa)**: tentare di ripulire i caratteri dannosi, ma è più fragile e difficile da mantenere.

### SOVRACCARICO DEL BUFFER:

Il buffer overflow è una vulnerabilità che si verifica quando più dati del previsto vengono scritti in un buffer (una zona di memoria), causando la sovrascrittura di memoria adiacente.

Un esempio classico è l'uso della funzione `gets()` in C, che non controlla la lunghezza dell'input: se l'utente inserisce più di 12 caratteri, si sovrascrive memoria oltre il buffer `pwd[12]`.

Questa sovrascrittura può compromettere il corretto funzionamento del programma, fino a modificare l'indirizzo di ritorno di una funzione, con conseguente esecuzione di codice dannoso.

Esistono vari tipi di overflow:

- **Heap overflow**: che coinvolge l'area di memoria dinamica;
- **Stack overflow**: più pericoloso, perché può alterare direttamente il flusso di esecuzione del programma.

Un attacco comune è l'**arc injection**, in cui l'attaccante modifica il flusso del programma iniettando un indirizzo di salto dannoso. Se conosce l'indirizzo esatto dove ha iniettato il proprio malware, può eseguire una vera e propria **code injection**.

In scenari reali, l'indirizzo esatto dello stack non è sempre prevedibile. Per questo motivo, l'attaccante può utilizzare una sequenza di istruzioni "NOP" (no operation), nota come **NOP slide**, per aumentare la probabilità che il programma esegua il codice iniettato.

I principali sistemi operativi e compilatori hanno adottato una serie di protezioni da stack overflow. Non rendono l'attacco impossibile, ma solo più impegnativo dal punto di vista tecnico:

- **Data Execution Prevention (DEP)**: rende non eseguibile lo stack. Se il programma tenta di eseguire codice iniettato, genera un errore;
- **Return-Oriented Programming (ROP)**: tecnica che aggira il DEP usando "gadget" (piccoli pezzi di codice legittimo terminanti con un'istruzione di ritorno) per eseguire codice arbitrario;
- **Address Space Layout Randomization (ASLR)**: randomizza la posizione delle aree di memoria del programma, rendendo più difficile per l'attaccante prevedere indirizzi utili. Tuttavia, non è infallibile: l'attaccante può usare leak di puntatori o sfruttare librerie condivise non randomizzate;
- **Stack Canary**: è un valore casuale inserito nello stack tra le variabili locali e l'indirizzo di ritorno. Se il canary viene modificato (segno di overflow), il programma termina. È una delle difese più efficaci, anche se
  - può essere aggirato da overflow che non sovrascrivono l'indirizzo di ritorno;
  - non protegge dalle buffer over-read, ovvero letture oltre il limite del buffer che causano leak di informazioni;
  - ha un costo in termini di performance, motivo per cui alcuni compilatori lo disattivano per funzioni ritenute "sicure".

```
int check_pwd() {
    char pwd[12];
    gets(pwd);
    return (strcmp(pwd, "p4ssw0rd") == 0);
}

int main() {
    int pwd_ok;
    puts("Enter password:");
    pwd_ok = check_pwd();
    if (!pwd_ok) {
        puts("Access denied");
        exit(-1);
    }
    puts("Access granted");
    // ...
}
```

### CODIFICA SICURA:

La codifica sicura **analizza gli errori che in passato hanno causato gravi vulnerabilità software**. Linguaggi come **C** e **C++**, pur essendo efficienti, sono particolarmente esposti perché non effettuano controlli impliciti (es. su limiti di buffer).

- **Comportamenti indefiniti** (es. accesso fuori limite, puntatori nulli);
- **Comportamenti non specificati** (es. ordine di valutazione);
- **Comportamenti inaspettati** (ipotesi sbagliate del programmatore)

I **dati contaminati** (non sanificati) possono causare problemi se usati in funzioni con input ristretto (sink).

Funzioni **pericolose**:

- `gets()`: sempre vulnerabile: non controlla la lunghezza. Ex:

```
char username[8];
get(username); //input "malicious" provoca overflow
```

- `strcpy`, `strcat`, `strcmp`: rischiano di scrivere fuori dal buffer;
- `sprintf` → preferire `sprintf`;
- Vulnerabilità di formattazione: `printf`, `fprintf`, `sprintf`, `snprintf` se l'input non è validato;

Alternative più sicure sono:

- `fgets` (con allocazione dinamica);
- `strncpy`, `strncat`;
- libreria `std::string` in C++ con operatori `>>` e `<<`;
- Java limita molti attacchi grazie all'architettura più sicura;

Infine, il **CWE (Common Weakness Enumeration)** è un elenco pubblico delle debolezze software più comuni, utile come riferimento per evitare errori noti.

## 4.5. GDPR E PRIVACY IOT

### INTRODUZIONE:

L'**Internet of Things (IoT)** si basa sull'identificazione continua degli utenti per fornire servizi personalizzati. Tuttavia, ciò può mettere a rischio la **privacy**, specialmente in assenza di anonimizzazione dei dati o di misure di sicurezza robuste.

Il **GDPR** introduce principi chiave per la protezione dei dati personali, tra cui:

- consenso informato;
- privacy by design/default;
- valutazione d'impatto;
- trasparenza algoritmica;
- limitazione del trattamento.

Secondo il GDPR e il WP29:

- **Produttori di dispositivi, sviluppatori di app e piattaforme IoT** possono essere considerati **Titolari del trattamento**;
- Devono adottare misure come **pseudonimizzazione** e **minimizzazione dei dati**.

Oltre al GDPR, il **futuro Regolamento ePrivacy** coprirà anche comunicazioni **macchina-macchina**, incluse quelle con dati non personali.

Tuttavia, il livello applicativo (es. app) rimane soggetto al GDPR.

La privacy deve essere integrata **fin dall'inizio** della progettazione IoT. Pratiche consigliate includono:

- **crittografia e autenticazione** per dati in transito e a riposo;
- **controlli di accesso** basati sul principio del minimo privilegio;
- **audit e logging** delle operazioni;
- **verifica dell'integrità dei dati**;
- **modifica delle impostazioni predefinite di privacy**;
- **formazione del personale** su sicurezza e protezione dati;
- **containerizzazione** delle applicazioni.

Approcci pratici alla PbD (Privacy by Design):

- **isolamento del backend**;
- **separazione e redazione dei dati**;
- **tecniche di trasformazione** che eliminano dati identificabili.

### IDENTITÀ UTENTE:

L'identificazione è cruciale per garantire comunicazioni sicure nell'IoT, permettendo che solo dispositivi e utenti autorizzati accedano ai dati. Tuttavia, le tecnologie di identificazione possono minacciare la privacy, soprattutto con l'uso di dati biometrici o comportamentali.

L'**identità digitale** di un utente viene spesso costruita esternamente, tramite profilazioni e inferenze automatiche, talvolta all'insaputa dell'interessato. Questo limita il controllo dell'utente su come viene percepito o trattato dai sistemi.

Per proteggere la privacy si possono usare **identità virtuali** e **ombre digitali**, che condividono solo dati minimi e contestuali, evitando la rivelazione dell'identità completa. Tuttavia, la personalizzazione dei servizi spesso richiede l'unione di dati da diverse fonti, aumentando i rischi.

Principali criticità:

- profilazione e discriminazioni tramite collegamento tra identità e dati IoT;
- divulgazione involontaria di informazioni sensibili;
- mancanza di trasparenza e controllo sull'identità digitale.

Serve quindi un equilibrio tra personalizzazione dei servizi e protezione della privacy.

## PROFILAZIONE:

Nel contesto IoT, la profilazione può derivare da diversi metodi di monitoraggio, inclusi la raccolta di dati che porta a inferenze sulla persona, la profilazione tramite collegamento di set di dati IoT, e la profilazione tramite condivisione con terze parti. La condivisione di dati tra dispositivi può comportare inferenze invasive. Ad esempio, dispositivi come Fitbit possono essere utilizzati per dedurre comportamenti potenzialmente discriminatori, come l'impulsività o la scarsa capacità di ritardare la gratificazione, che potrebbero influenzare negativamente l'immagine di un individuo.

Anche se alcune inferenze non sono dannose, la profilazione può portare a discriminazioni ingiuste, anche quando si utilizzano dati non sensibili, ma da cui si possono comunque trarre informazioni sensibili. L'uso di machine learning nell'IoT può complicare ulteriormente la previsione e la comprensione delle inferenze, creando rischi imprevisti per la privacy.

Problemi Legali e Diritti:

- **Articolo 21 del GDPR:** consente agli utenti di opporsi al trattamento dei dati, inclusa la profilazione;
- **Articolo 22 del GDPR:** fornisce garanzie contro il processo decisionale automatizzato, concedendo agli utenti il diritto di intervenire e contestare decisioni automatizzate;

## CONDIVISIONE DEI DATI/IDENTITÀ:

Gli utenti spesso non hanno controllo sulla divulgazione dei propri dati e identità nell'IoT, il che può portare a profilazioni discriminatorie. I meccanismi di condivisione dei dati devono essere trasparenti, ma dare il controllo agli utenti potrebbe non essere sempre utile se non sono pienamente consapevoli dei rischi.

I modelli **multi-utente** e **multi-controller** dei dispositivi complicano ulteriormente il controllo sull'identità. Gli utenti, infatti, si trovano a dover gestire informazioni personali in sistemi complessi dove la privacy è spesso compromessa.

Diritti degli Utenti secondo il GDPR:

- **Diritto di accesso (Articolo 15):** gli utenti hanno il diritto di chiedere informazioni sui dati trattati e ottenere una copia;
- **Diritto di rettifica (Articolo 16):** gli utenti possono correggere dati inesatti;
- **Diritto all'oblio (Articolo 17):** gli utenti possono chiedere la cancellazione dei dati non necessari per gli scopi per cui sono stati raccolti;

Questi diritti promuovono la trasparenza, ma c'è ancora incertezza su come bilanciare gli interessi degli utenti e dei titolari del trattamento, specialmente in caso di dati imprecisi o incompleti.

## CONSENSO E INCERTEZZA:

I sistemi di gestione dell'identità e di controllo degli accessi nell'IoT, che migliorano privacy e fiducia, si basano sulla supervisione e scelta dell'utente su come i suoi dati vengano condivisi. Tuttavia, la protezione effettiva dipende dalla qualità delle scelte fatte dagli utenti, che devono essere informati sui rischi associati alla condivisione dei dati. Il GDPR, attraverso l'articolo 25, impone misure di "privacy by design" e "privacy by default" per risolvere l'incertezza legata alla privacy e garantire un consenso informato.

L'articolo 25 stabilisce che i titolari del trattamento dei dati devono implementare misure come la minimizzazione dei dati, la limitazione della conservazione e della finalità, sebbene queste misure siano ostacolate dalla necessità di raccolta e condivisione di grandi volumi di dati per offrire servizi personalizzati. Il GDPR stabilisce standard più elevati per il consenso informato (articolo 7) e obblighi di notifica (articoli 13 e 14), rendendo gli utenti consapevoli dei rischi della raccolta dei dati, senza aspettarsi che leggano informazioni complesse sulla privacy.

## ONESTÀ, FIDUCIA E TRASPARENZA:

Le relazioni di fiducia nell'IoT sono essenziali per la condivisione dei dati tra dispositivi, utenti e controller, e sono rafforzate da autorizzazioni associate a identità specifiche. La fiducia tra oggetti dipende dall'autenticazione prima della comunicazione, mentre quella tra utenti e dispositivi riguarda la percezione di controllo degli utenti.

Il GDPR promuove la fiducia e la trasparenza, richiedendo la valutazione d'impatto sulla protezione dei dati (DPIA), che può aumentare la fiducia degli utenti, specialmente se pubblicata. Le normative stabiliscono che gli utenti siano informati sui rischi legati al trattamento automatizzato dei dati, inclusa la profilazione, e sui metodi algoritmici usati. Inoltre, il GDPR enfatizza l'importanza di un sistema sicuro che soddisfi gli standard minimi per aumentare la fiducia. In caso di violazioni, i titolari devono informare le autorità competenti entro 72 ore e, se necessario, avvisare gli utenti.

Queste disposizioni sono essenziali per garantire trasparenza e rafforzare il rapporto di fiducia tra utenti e fornitori di dispositivi IoT, pur con la sfida di definire chiaramente cosa costituisca un "alto rischio" nelle violazioni dei dati.

## METODI DI TUTELA DELLA PRIVACY:

I problemi di privacy nell'IoT non riguardano solo gli utenti direttamente connessi, ma anche coloro che si trovano nell'ambiente. A causa della mancanza di confini di controllo ben definiti, catturare le violazioni della privacy nell'IoT è complesso.

L'**anonimizzazione** dei dati rimuove le informazioni identificabili per proteggere la privacy degli utenti. La **denaturazione** offusca o altera parti specifiche delle immagini per garantire la privacy. L'**oblio digitale** riguarda l'eliminazione dimostrabile dei dati, mentre la **sintesi dei dati** fornisce un'astrazione per nascondere dettagli specifici e ridurre la granularità.

La **privacy differenziale** è una definizione matematica che assicura che non sia possibile determinare se i dati di un individuo siano stati inclusi in un set di dati.

Aggiungendo "rumore" ai dati, il comportamento dell'algoritmo non cambia quando un singolo individuo entra o esce dal database.

Questo garantisce che le informazioni individuali non vengano divulgate, ma non assicura che i segreti rimangano tali.

L'**oblio digitale** e la **sintesi dei dati** sono essenziali per affrontare le preoccupazioni degli utenti sulla raccolta dei dati. Con il calo dei costi

Solution	Summary
Authentication and Authorization	(i) Lightweight authentication and key establishment mechanisms (ii) Frameworks based on device fingerprinting techniques (iii) Context-aware access control models and enforcing mechanisms
Edge Computing and plug in architecture	(i) Software modules on the edge to overcome privacy concerns (ii) Privacy aware systems to allow user control over data (iii) Decentralized architectures based on Personal-Cloud Butlers
Data Anonymizing and denaturing	(i) Data brokers and separation algorithms to offer flexibility to service providers, yet respect user-predefined access rules (ii) Generalization to mask personal data (iii) Frameworks that provide emotion analytics lifecycle to allow denaturing
Digital Forgetting and Data Summarization	(i) Delete encrypted data when decryption key is deleted (ii) Acquire only the strictly needed data rather than all data (iii) Apply knowledge discovery in databases and data mining technologies

di archiviazione, cresce la capacità di memorizzare enormi quantità di dati, creando la necessità di una cancellazione periodica delle informazioni.

La **sintesi dei dati** può essere classificata in due modalità: **temporale**, dove i dati dipendono dal tempo, e **spaziale**, dove i dati sono correlati alla posizione. Diverse tecniche di oblio dei dati utilizzano la crittografia, particolarmente efficace nei nodi di archiviazione distribuiti come nell'IoT. La maggior parte di queste tecniche prevede che i dati crittografati vengano eliminati quando la chiave di decrittazione non è più disponibile.

### ALGORITMI DI SANIFICAZIONE:

Per bilanciare privacy e utilità dei dati, esistono algoritmi di sanificazione che utilizzano dati individuali o risultati analitici come input, applicando definizioni di privacy e metriche per l'utilità. Questi algoritmi riducono la precisione dei dati, mantenendo gli output utili senza che un avversario possa incrementare la propria conoscenza osservando i dati pubblici.

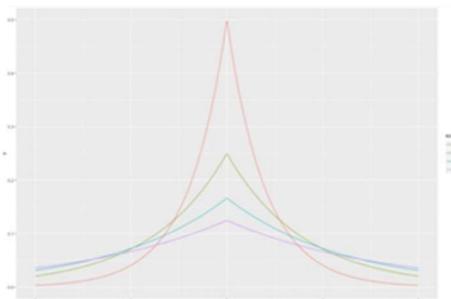
Un esempio concreto di una soluzione IoT che raccoglie valutazioni degli utenti

per un servizio aziendale può essere usato per illustrare il funzionamento

dell'algoritmo. Se un avversario vuole sapere quante persone hanno una valutazione negativa (ad esempio 3), l'algoritmo restituisce il numero (3) più un valore di "rumore" casuale estratto da una distribuzione di Laplace centrata su zero con deviazione standard pari a 2, ottenendo così un risultato  $N+L$  (dove  $L$  è il rumore casuale).

Questo processo garantisce un equilibrio tra utilità e privacy. L'avversario riceve una risposta vicina, ma non uguale, alla verità, proteggendo la privacy dell'utente. Tuttavia, se l'avversario ripete abbastanza query al database, sarà in grado di stimare la verità di base. Ogni tentativo fornisce una media che si avvicina alla verità, ma l'intervallo di confidenza può essere ampio. Con più query, gli intervalli si restringono, e l'avversario può fare stime più accurate. In generale, sono necessarie circa 50 query per ottenere una stima decente.

L'algoritmo, pur proteggendo la privacy con l'aggiunta di rumore, ha come limitazione la possibilità che un avversario, tramite ripetute query, possa alla fine dedurre i dati sensibili.



## 5.1. MANOMISSIONE DEI DISPOSITIVI IOT

### NODE CAPTURE:

Uno dei possibili **attacchi** agli **IoT devices** è il **Node Capture**, che consiste nell'ottenere il controllo totale di un dispositivo IoT tramite accesso fisico diretto. Questo tipo di attacco è diverso dall'attacco remoto tramite **bug software**, in cui un errore software permette a un attaccante di compromettere tutti i dispositivi che eseguono lo stesso software vulnerabile.

L'accesso fisico consente di manomettere direttamente l'hardware, con impatti che possono variare da **minimi** (se vengono rubate solo piccole quantità di dati) a **gravi** (se vengono compromessi **routing**, **dati sensibili** o altre funzioni critiche)

### INFO ACQUISITION:

Per eseguire un attacco di **node capture**, è necessario un **conoscimento preciso** del dispositivo target, in modo da individuare le interfacce fisiche accessibili.

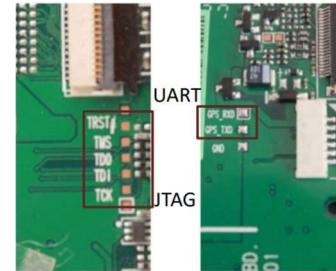
- Inizialmente si esegue una **ispezione visiva** dell'esterno del dispositivo per capire come funziona l'I/O e quali **interfacce** sono esposte;
- Poi si procede con un'**ispezione interna**, smontando il dispositivo per accedere ai componenti interni, ma con attenzione a non danneggiarlo;

Durante l'ispezione interna, si possono trovare dettagli utili, come il **processore Samsung ARM** con identificativo **S3C2440AL**. Consultando i **datasheet** di questi componenti, l'attaccante può ottenere informazioni vitali per l'attacco.



Un aspetto cruciale da determinare è la presenza di **porte di debug** e **interfacce di comunicazione**:

Le **interfacce di comunicazione** esposte (ad esempio **UART**, **JTAG**) possono essere sfruttate per ottenere accesso non autorizzato, **log** o addirittura un **root shell non autenticato** sul dispositivo target.



Se le informazioni tecniche online sono difficili da trovare, un altro metodo utile è l'ID della Federal Communications Commission (FCC). Questo ID può essere usato per cercare informazioni nel database della FCC o su siti web terzi come fccid.io o fcc.io.



## 5.2. UART, I2C, SPI

### COMUNICAZIONE A BASSO LIVELLO:

Per ogni dispositivo embedded, i vari componenti devono interagire tra loro e scambiarsi dati. Esistono due principali modalità di comunicazione:

- **Comunicazione seriale**: trasferisce un bit alla volta su un determinato canale;
- **Comunicazione parallela**: trasferisce un blocco di dati contemporaneamente, con ogni bit che richiede un canale separato;

**Comunicazione seriale** è più comune nei dispositivi embedded, in quanto richiede solo una linea per il trasferimento dei dati, mentre la **comunicazione parallela** è più veloce, ma occupa più spazio sulla scheda.

Alcuni dei **canali di comunicazione seriale** più comuni sono:

- **RS232** (usato in vecchi sistemi di computer);
- **USB** (Universal Serial Bus);
- **PCI** (Peripheral Component Interconnect);
- **HDMI** (High-Definition Multimedia Interface);
- **Ethernet**;
- **SPI** (Serial Peripheral Interface);
- **I2C** (Inter-Integrated Circuit);
- **CAN** (Controller Area Network)

Grazie ai recenti sviluppi tecnologici, la comunicazione seriale sta diventando **più economica, più veloce e più affidabile**.

### COMUNICAZIONI SERIALI PRINCIPALI:

- **UART** (Universal Asynchronous Receiver/Transmitter): è semplice, non richiede un clock e supporta solo una comunicazione unidirezionale tra due dispositivi;
- **I2C** (Inter-Integrated Circuit): più veloce di UART, ma non così veloce come SPI. Consente di collegare più dispositivi (fino a 127) usando solo due linee di comunicazione;
- **SPI** (Serial Peripheral Interface): è il protocollo più veloce dei tre, ma richiede più pin. Supporta una comunicazione full-duplex, cioè dati possono essere inviati e ricevuti simultaneamente;

### UART:

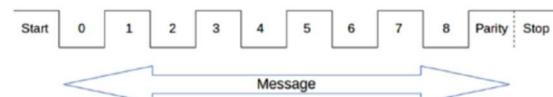
**UART** è una forma di comunicazione seriale che non richiede un clock esterno. Questo significa che non c'è bisogno di una linea

aggiuntiva per sincronizzare i due dispositivi. **Asincrono** significa che non esiste un segnale di sincronizzazione (come nel caso di SPI), ma ogni dispositivo deve essere configurato per la stessa **velocità di trasmissione**.

La struttura dei pacchetti UART è la seguente:

The UART packets structure:

- **Bit di partenza**: segnala l'inizio della trasmissione;
- **Messaggio**: il dato reale che viene trasferito;
- **Bit di parità**: utilizzato per il controllo degli errori;
- **Bit di stop**: segnala la fine della trasmissione



La porta UART può essere **hardware** o **software**. Se il dispositivo ha una sola porta UART, ma sono necessari più dispositivi da connettere, si possono emulare porte **UART software**.

Il **baud rate** è la velocità con cui i dati vengono trasferiti tra dispositivi durante una comunicazione UART. Inoltre, il **baud rate** (velocità di trasmissione dei dati) deve essere concordato tra i dispositivi. Poiché **UART** non ha un segnale di clock, entrambe le estremità della comunicazione devono essere configurate con la stessa velocità di trasmissione. Alcuni **baud rate comuni** sono:

- 9600, 19200, 38400, 57600, 115200 bits al secondo;

Durante un **attacco UART**, il primo passo sarà sempre **identificare il baud rate** corretto del dispositivo target. Questo può essere fatto in vari modi, come provare diverse velocità di baud e vedere quale restituisce dati leggibili. Uno strumento utile in questo processo è lo script **baudrate.py**, che aiuta a cambiare il baud rate durante la connessione seriale per identificare quello corretto.

L'output di questo script mostrerà quando viene identificato il baud rate corretto, poiché i dati diventeranno leggibili.

## I2C & SPI:

**I2C** è un protocollo di comunicazione **multi-master** che usa solo due fili: uno per i dati (SDA) e uno per il clock (SCL). È **half-duplex**, cioè può inviare o ricevere dati in un dato momento, ma non entrambi. **I2C** è limitato a comunicare su una **singola scheda** o circuito. Per permettere a più dispositivi di inviare, sono necessarie linee separate per ogni **dispositivo slave**.

**SPI** è un protocollo di comunicazione **full-duplex** che utilizza 4 fili:

- **SCK** (Clock);
- **MOSI** (Master Out Slave In);
- **MISO** (Master In Slave Out);
- **CS** (Chip Select).

Rispetto a **I2C**, **SPI** è più veloce, ma richiede più pin sulla scheda. Entrambi i protocolli vengono usati per la **memoria EEPROM**, dove è possibile leggere o scrivere dati da e su dispositivi di memoria.

Le **EEPROM seriali** tipicamente hanno 8 pin e possono essere interfacciate sia con **I2C** che con **SPI**.

In **SPI**, il master seleziona uno slave e comincia a trasmettere dati. Le linee **SCK** (clock), **MISO** (data out) e **MOSI** (data in) sono utilizzate per il trasferimento dei dati.

In **I2C**, il master può controllare il clock, ma ogni slave ha la possibilità di rallentarlo tramite una tecnica chiamata **clock stretching**.

In **I2C**, il master seleziona e comunica con gli slave attraverso la linea **SDA** (dati) e **SCL** (clock). I dati possono essere letti o scritti sulla memoria della EEPROM, utilizzando uno script Python per la lettura dei dati.

Entrambi i protocolli consentono di comunicare con vari dispositivi attraverso una rete, ma **SPI** è più veloce rispetto a **I2C**, che è ideale per dispositivi a bassa velocità.

Pin name	Function
#CS	Chip select
SCK	Serial data clock
MISO	Serial data input
MOSI	Serial data output
GND	Ground
VCC	Power supply
#WP	Write protect
#HOLD	Suspends serial input

## 5.3. SFRUTTAMENTO DELLA COMUNICAZIONE A BASSO LIVELLO

### UART EXPLOITATION:

Per eseguire un'**exploitation** basata su **UART**, è necessario un dispositivo che possa emulare una connessione seriale per accedere al dispositivo target, come l'**Attify Badge**. Altri strumenti alternativi includono **USB-TTL** o **BusPirate**.

Per stabilire la connessione, bisogna individuare le **porte UART** del dispositivo target, che sono solitamente visibili tramite un'ispezione visiva dell'hardware. Si cerca una serie di **3 o 4 pin** o **pad** vicini tra loro.

La connessione **UART** richiede quattro pin principali:

- **Trasmissione (Tx)**: trasmette i dati;
- **Ricezione (Rx)**: riceve i dati;
- **Massa (GND)**: pin di riferimento per la terra;
- **Tensione (Vcc)**: solitamente 3.3V o 5V.



Un **multimetro** può essere utilizzato per identificare questi pin tramite il test di **continuità** (per GND) o misurando la **differenza di tensione** per gli altri pin.

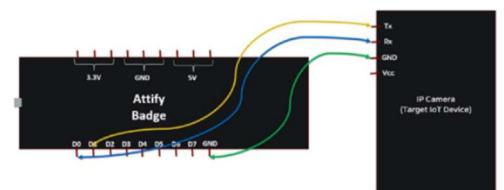
L'**Attify Badge** ha un totale di **18 pin**, di cui 10 sono utilizzati per **tensione (3.3V/5V)** e **terra (GND)**. I pin **D0-D3** sono utilizzati per interagire con i dispositivi embedded tramite vari protocolli di comunicazione, tra cui **UART**, **SPI** e **I2C**.

Pin	UART	SPI	I2C	JTAG
D0	TX	SCK	SCK	TCK
D1	RX	MISO	SDA*	TDI
D2		MOSI	SDA*	TDO
D3		CS		TMS

La connessione tra il **target** e l'**Attify Badge** avviene nel seguente modo:

- **Tx del target** va al **Rx dell'Attify Badge (D1)**;
- **Rx del target** va al **Tx dell'Attify Badge (D0)**;
- **GND del target** va al **GND dell'Attify Badge**.

**Importante:** Non collegare **Vcc**, poiché ciò potrebbe danneggiare irreparabilmente il dispositivo target. Una volta completata la connessione, il dispositivo **Attify Badge** può essere collegato a un computer tramite una **micro USB**.



Quando l'**Attify Badge** è connesso, il sistema operativo del computer rileva il nuovo dispositivo. Ad esempio, su **Linux/Mac OS**, il dispositivo apparirà come una voce in **/dev/**, come **/dev/ttysUB0**. Questo è anche il COM port predefinito per **baudrate.py**.

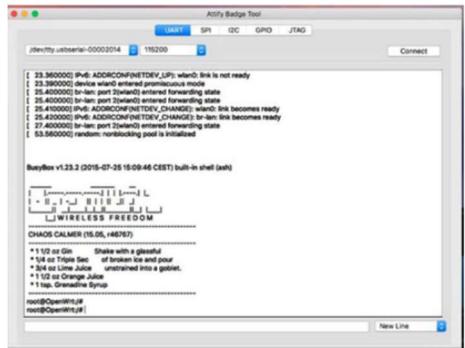
	loop-control	sda	tty25	tty57	tty53
agpport	mapper	sda1	tty26	tty58	tty50
autofs	mem	sda2	tty27	tty59	tty51
block	mcelog	sda5	tty28	tty60	tty54
bsg	memory_bandwidth	serial	tty29	tty61	tty55
btrfs-control	net	sg0	tty3	tty62	tty56
bus	network_latency	sg1	tty30	tty63	tty57
cdrom	network_throughput	sg2	tty31	tty64	tty58
char	null	shm	tty32	tty7	tty59
console	port	snapshot	tty33	tty8	tty50
core	ppp	snd	tty34	tty9	uhid
cpu	psaux	sr0	tty35	ttyprintk	uinput
cpu_dma_latency		srl	tty36	tty50	urandom

Utilizzando lo script **baudrate.py**, è possibile identificare il baud rate del dispositivo target provando diverse velocità fino a quando non vengono visualizzati dati leggibili.

Una volta identificato il baud rate corretto, si può interagire con il dispositivo target tramite **UART**. Questo può essere fatto utilizzando strumenti come **screen** o **minicom**, che permettono di stabilire una connessione seriale.

Ad esempio, dopo aver riavviato il dispositivo target, i **log di debug** possono essere visualizzati e, in alcuni casi, è possibile ottenere un **root shell non autenticato** sul dispositivo.

Esiste uno strumento grafico chiamato **Attify Badge GUI**, disponibile su GitHub, che può semplificare il processo di exploitazione tramite un'interfaccia grafica.



## I2C EXPLOITATION:

Exploiting I2C significa leggere o scrivere i dati di un dispositivo utilizzando una EEPROM I2C in un dispositivo del mondo reale. Per farlo, è necessario un dispositivo che utilizzi un chip di memoria flash che funzioni con il protocollo di comunicazione I2C.

Un esempio di EEPROM I2C è il MicroChip 24LC256, che è una memoria 256K I2C organizzata come memoria seriale 32K × 8. Questa memoria può essere letta o scritta tramite il protocollo I2C.

Per sfruttare l'attacco, è possibile trovare il nome del componente nel datasheet e cercarlo online, dove si troveranno anche le informazioni sulle pinnature (pinout) dell'EEPROM.

Per eseguire l'attacco, bisogna connettere l'EEPROM all'**Attify Badge**. Ciò può essere fatto in due modi:

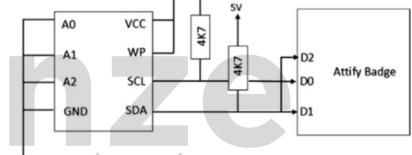
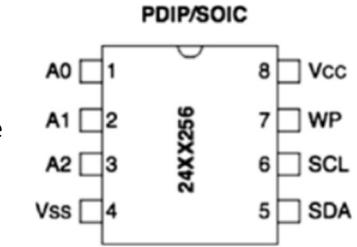
- Collegando direttamente l'EEPROM con una **clip SOIC**;
- Rimuovendo l'EEPROM dal dispositivo e saldandola su un **adattatore EEPROM**.

Le connessioni per il **MicroChip 24LC256 EEPROM** sono le seguenti:

- A0, A1, A2 e GND**: collegati a **GND**;
- Vcc e WP**: collegati a **5V** (dove **WP** è il pin di protezione scrittura, attivo basso);
- D1 e D2** dell'Attify Badge: collegati alla linea **SDA**;
- D0** dell'Attify Badge: collegato alla linea **SCL (Clock)**.

Per interagire con l'EEPROM I2C, è possibile usare uno script Python chiamato **i2ceeprom.py**. Questo script consente di leggere e scrivere dati sulla memoria EEPROM tramite il protocollo I2C.

Lo script è disponibile su GitHub e permette di configurare la dimensione dell'EEPROM e la velocità del clock (400 KHz, in questo caso). Dopo aver letto o scritto i dati, la connessione **I2C** viene chiusa e i dati letti vengono salvati nel file **EEPROM.bin**.



## SPI EXPLOITATION:

Per leggere e scrivere dati su un **SPI EEPROM**, si utilizza uno script chiamato **spiflash.py**, disponibile su GitHub. Lo script è progettato per interagire con dispositivi SPI e può essere scaricato dalla cartella **src/examples/** di **libmpsse**.

- Definizione dei comandi**: lo script imposta i comandi di lettura e scrittura predefiniti, che sono compatibili con la maggior parte dei chip SPI;
- Impostazione della velocità**: La velocità di comunicazione predefinita con il chip target è di **15 MHz**, ma può essere modificata tramite l'opzione **-f**;
- Impostazione dei pin**: I pin **WP** (Write Protect) e **HOLD** vengono impostati come **alti**.
- Connessione e operazioni**: Lo script si connette al chip target tramite la libreria **mpsse** e permette di eseguire operazioni di lettura, scrittura e cancellazione usando i flag specificati (WCMD, RCMD, WECMD, RECMD).

Per eseguire l'**SPI exploitation**, il chip flash target può essere **rimosso** dalla scheda (PCB) tramite desaldatura. Una volta rimosso, può essere saldato a un **adattatore EEPROM** (o lettore) per la lettura.

Alternativamente, è possibile **leggere il chip direttamente** senza rimuoverlo dalla scheda, utilizzando miniprobes o una clip SOIC su un dispositivo IoT reale.

Il passo successivo è fare le connessioni necessarie utilizzando l'**Attify Badge** o qualsiasi hardware compatibile con **FTDI**. Per determinare correttamente i pin del chip, confronta il chip reale con il **datasheet** e usa la **tacca** nell'angolo superiore sinistro del chip per identificare i numeri dei pin.

Per **leggere o scrivere** nella memoria flash tramite SPI, il dispositivo **Attify Badge** o un altro hardware compatibile con **FTDI** deve essere utilizzato per fare le connessioni. I pin SPI principali che devono essere collegati sono:

- SCK** (Clock);
- MOSI** (Master Out Slave In);
- MISO** (Master In Slave Out);

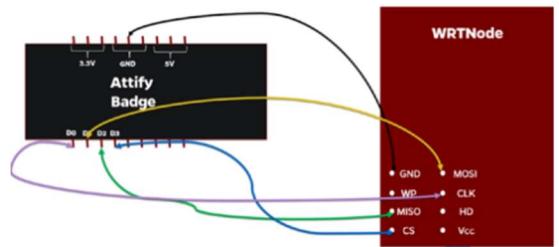
- CS (Chip Select).

Una volta che il dispositivo è correttamente connesso, è possibile eseguire il dump del contenuto della memoria flash utilizzando lo script ***spiflash.py***.

Con qualsiasi dispositivo, è possibile **estrarre** il contenuto memorizzato nel **chip EEPROM** del dispositivo. Inoltre, è possibile **scrivere** nuovi dati nel chip. Se è possibile interagire con il chip **EEPROM** tramite **SPI**, è anche possibile **scrivere una versione modificata** del firmware del dispositivo.

Si considera un dispositivo chiamato **WRTNode**, che ha un firmware completo (in questo caso **OpenWRT**), con l'obiettivo di **estrarre** il firmware utilizzando lo script **spiflash.py** e l'**Attify Badge**. Il **WRTNode** dispone di vari **pin e pad** per consentire le connessioni. Il **datasheet** del **WRTNode** è disponibile online, poiché si tratta di una **scheda di sviluppo** molto conosciuta.

I pin del **WRTNode** e dell'**Attify Badge** devono essere correttamente connessi per supportare la comunicazione tramite il protocollo **SPI**. Il file **wrtnode-dump.bin** contiene tutti i dati letti e può essere passato a strumenti di **analisi del firmware** per ottenere l'intero **file system**. Dopo aver effettuato la connessione, il passo successivo è eseguire **spiflash.py**, specificando una dimensione abbastanza grande per **estrarre l'intero contenuto** del chip flash.



#### **5.4. JTAG E IL SUO SFRUTTAMENTO**

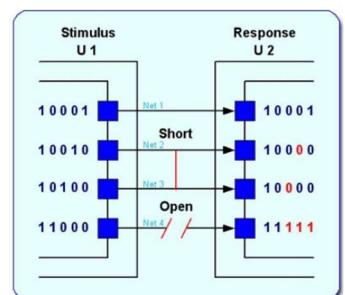
---

**JTAG:**

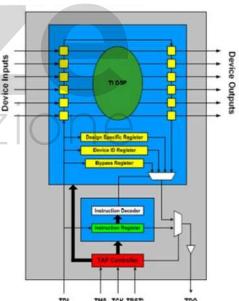
Il **JTAG** (Joint Test Action Group) è uno standard creato negli anni '80 da un gruppo di aziende per risolvere il problema di **debugging** e **test** dei chip, che diventavano sempre più complessi. Prima di JTAG, i test manuali dei chip erano difficili e lenti, soprattutto con l'aumento dei pin e della produzione di chip. Per risolvere questo problema, è stato creato lo standard **IEEE 1149.1**, che consente di incorporare un hardware direttamente nel chip per semplificare i test.

JTAG è un metodo per testare i chip presenti in un dispositivo e fare **debugging** tramite una tecnica chiamata **boundary scan**. Questo viene fatto aggiungendo **cellule di scansione** vicino ai pin del chip, creando una catena di test seriale. La catena può essere accessibile tramite la **Test Access Port (TAP)**. I dati vengono inviati nei chip e l'output viene confrontato con l'input per verificare che tutti i componenti funzionino correttamente.

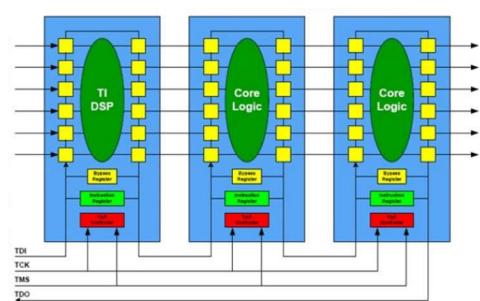
I fornitori di strumenti di **boundary scan** offrono algoritmi avanzati per non solo rilevare i **fallimenti** dei circuiti, ma anche per isolare i guasti a specifici **pin**, **dispositivi** e **reti**. Un file esterno definisce le capacità di test per ciascun dispositivo con **boundary scan**. Le celle di scansione possono essere accessibili per verificare i valori dei pin ad esse associati.



I circuiti integrati (IC) sono costituiti da celle di ***boundary scan*** che si trovano tra la logica del sistema e i pin di segnale che collegano l'IC alla scheda PCB. Ogni cella fornisce capacità di test specifiche. Le celle di scansione all'interno di un dispositivo sono collegate in serie per formare un ***shift register***, che può essere accessibile tramite le interfacce ***TDI*** (Test Data Input) e ***TDO*** (Test Data Output). Un controller standard è utilizzato per eseguire funzioni come ***scansionare i dati*** in ingresso e in uscita.



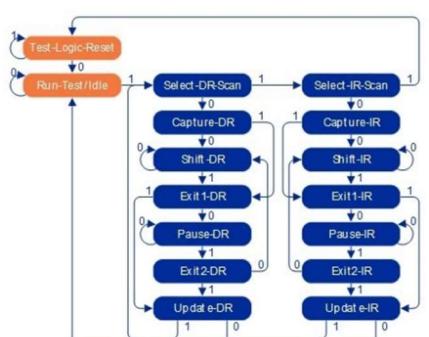
Il meccanismo di ***boundary scan*** può essere utilizzato per testare i componenti interni di un dispositivo o la connessione tra dispositivi diversi. Il software di ***boundary scan*** può utilizzare un componente per inviare segnali che vengono rilevati su un altro componente, verificando così la ***continuità*** tra i pin. I dispositivi possono essere messi in modalità ***BYPASS*** per ridurre la lunghezza complessiva della catena e velocizzare il test.



Il termine **TAP** si riferisce alle interfacce ITAG presenti su un dispositivo. Ci sono cinque segnali utilizzati dal TAP:

- **Test Clock (TCK)**: per sincronizzare le operazioni interne e inviare dati seriali alle celle di scansione;
  - **Test Data In (TDI)**: pin di ingresso seriale per i dati nelle celle di scansione;
  - **Test Data Out (TDO)**: invia i dati dalle celle di scansione;
  - **Test Mode Select (TMS)**: per controllare lo stato del controller;
  - **Test Reset (TRST)** (opzionale): pin di reset attivo basso per azzerare la macchina a stati interni.

Il **TAP controller** è una macchina a stato finito (FSM) a 16 stadi che si sposta da uno stato all'altro in base ai segnali **TMS** e **TCK**. Il controller gestisce il **test data register** e l'**instruction register** con i segnali di controllo. Quando un'istruzione deve essere inviata,



viene attivato il **TCK** e il **reset** è impostato su attivo basso per il ciclo del clock. Una volta completata l'operazione, il reset viene disattivato e **TMS** viene alternato per far avanzare la macchina a stati.

Esistono diversi **comandi** definiti dallo standard **IEEE 1149.1** che devono essere resi disponibili per un dispositivo in caso di **boundary scan**:

- **BYPASS**: inserisce il registro di bypass nella catena **DR**, così che il percorso da **TDI** a **TDO** coinvolga solo un flip-flop, permettendo di testare un singolo chip senza interferenze da altri chip;
- **SAMPLE/PRELOAD**: posiziona il registro di scansione nel **DR chain**, per pre-caricare i dati di test nel registro di scansione e copiare il valore I/O del chip nel registro dati;
- **EXTEST**: permette di testare la circuiteria esterna al chip, facendo uscire il valore dal registro dati verso i pad di uscita;

Il processo complessivo di **boundary scan** funziona come segue:

- Il **TAP controller** applica i dati di test sui pin **TDI**;
- Il **Boundary Scan Register (BSR)** monitora l'ingresso del dispositivo e cattura i dati tramite le celle di scansione;
- I dati vanno nel dispositivo tramite i pin **TDI** e ne escono tramite **TDO**;
- Il tester può verificare i dati sui pin di uscita del dispositivo e confermare il corretto funzionamento.

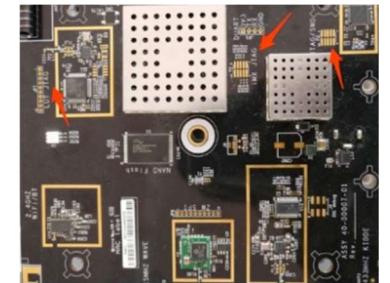
Questi test possono rilevare **difetti di produzione**, **componenti mancanti**, **pin non connessi** o **dispositivi posizionati erroneamente**.

**JTAG** è utilizzato per diversi **test** e **debugging**:

- Poiché **JTAG** è disponibile sin dal momento in cui il sistema si avvia, è estremamente utile per ingegneri e tester per esaminare i vari componenti del dispositivo embedded;
- Per i **penetration tester** e **ricercatori di sicurezza**, **JTAG** è utile per fare il **debug** del sistema target e dei suoi componenti. Se il dispositivo ha **accesso JTAG** e una **memoria flash onboard**, è possibile **estrarre** i contenuti della memoria tramite JTAG;

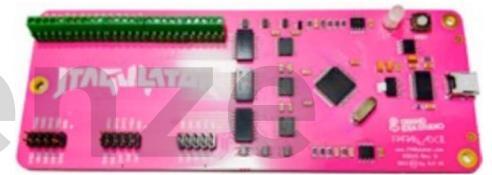
## JTAG EXPLOITATION:

L'identificazione dei **pin JTAG** può essere più complicata rispetto ad altri metodi. In molti dispositivi, i **pad JTAG** non sono pin veri e propri, ma **pad senza fori**. Per identificare correttamente i pin JTAG, è necessario utilizzare strumenti come **JTAGulator**, che è utile per determinare i pin di test in un dispositivo target. È anche utile avere esperienza di saldatura per lavorare con dispositivi reali.



Esistono due metodi per identificare i **pin JTAG**:

- **JTAGulator**: un dispositivo hardware open-source che aiuta a identificare i pin JTAG. Ha 24 canali I/O e può rilevare anche i pin **UART**. Si collega al computer tramite una porta seriale e può essere utilizzato tramite **screen** o **minicom**. Per usare JTAGulator, bisogna connettere i vari pin del dispositivo target ai canali di JTAGulator, collegando anche **GND**. Successivamente, è necessario impostare la **tensione del sistema target** tramite JTAGulator, quindi eseguire una scansione **BYPASS** per trovare i pin JTAG. JTAGulator identifierà i vari pin del dispositivo;
- **JTAGEnum con Arduino**: un'alternativa più economica a JTAGulator, ma più lenta e senza la capacità di rilevare i pin UART. Per usarlo, bisogna caricare il codice **JTAGEnum** sull'**Arduino IDE** e caricarlo sul dispositivo Arduino.



Mappando i fili con i pin JTAG sulla scheda target, si identificano i **pin JTAG** effettivi. Dopo aver connesso il dispositivo al **JTAG interface**, è possibile **debuggare** il dispositivo target e i programmi in esecuzione. Lo strumento **OpenOCD** è utilizzato per il debug, consentendo operazioni come:

- debug dei chip presenti sul dispositivo;
- impostazione di **breakpoint** e analisi dei **registri**;
- analisi e interazione con la **memoria flash** del dispositivo;
- dump del **firmware** e di altre informazioni sensibili.

**OpenOCD** deve essere installato sul computer tramite il comando **apt** o compilando il codice sorgente. Un altro strumento utile è **GDB-Multiarch**, che consente di usare **GDB** per il debug di binari destinati a diverse architetture. In alternativa, si può usare l'**Attify Badge** che preinstalla tutti gli strumenti necessari.

Per il debugging hardware, si può usare l'**Attify Badge**, **BusPirate** o **Segger J-Link**. L'Attify Badge funziona come un adattatore JTAG e deve essere configurato con i file di configurazione di **OpenOCD** per il dispositivo target. È necessario anche verificare se il **controller** del dispositivo è supportato da **OpenOCD**.

Il passo successivo è connettere l'**Attify Badge** all'interfaccia **JTAG** del dispositivo target. I pin **TCK (D0)**, **TDI (D1)**, **TDO (D2)** e **TMS (D3)** vanno collegati rispettivamente ai pin **CLK**, **TDI**, **TDO** e **TMS** del dispositivo target. I pin per **CLK**, **TDI**, **TDO** e **TMS** possono variare a seconda del **processore** o **controller** del dispositivo. Un esempio di dispositivo target considerato è un **microcontrollore STM32F103C8**.

Una volta che i dispositivi sono collegati tra loro e al computer tramite **USB**, è necessario configurare i file di configurazione per **OpenOCD**. Un esempio di file di configurazione per l'**Attify Badge** è **badge.cfg**, che include:

- **interface ftdi**: definisce l'interfaccia FTDI;
- **ftdi\_vid\_pid 0x0403 0x6014**: Vendor ID e Product ID per FTDI;
- **ftdi\_layout\_init 0x0c08 0x0f1b**: configurazione del layout;

▪ **adapter\_khz 2000**: definizione della velocità di clock (2000 kHz).

Per il **microcontrollore STM32**, il file di configurazione può essere ottenuto direttamente da **OpenOCD**.

**JTAG** è uno strumento utile per diversi tipi di **test** e **debugging**:

- poiché è disponibile fin dall'inizio del processo di avvio del sistema, **JTAG** consente a **tester** e **ingegneri** di esaminare facilmente tutti i componenti di un dispositivo embedded;
- Per **penetration tester** e **ricercatori di sicurezza**, **JTAG** è utile per il **debugging** del sistema target e dei suoi componenti. Se il dispositivo ha accesso **JTAG** e una **memoria flash onboard**, è possibile **estrarre** i dati dalla memoria tramite **JTAG**;

Dopo aver esplorato le fasi principali dello sviluppo sicuro di un prodotto IoT, è importante anche comprendere come utilizzare **JTAG** (Joint Test Action Group) per analizzare, debug e manipolare dispositivi a livello hardware. La connessione JTAG è fondamentale per l'accesso a basso livello dei dispositivi e può essere utilizzata per eseguire il debug, esaminare i componenti hardware, e, in alcuni casi, estrarre dati sensibili dalla memoria del dispositivo.

- **Scrittura del Firmware**: Dopo aver configurato la connessione JTAG e OpenOCD, è possibile scrivere direttamente il firmware nel dispositivo. È fondamentale conoscere l'indirizzo di partenza della memoria flash (ad esempio, 0x08000000) e verificare il successo con comandi come **flash banks** o **dump\_image** per eseguire un dump della memoria;
- **Lettura Dati dalla Memoria**: JTAG consente di leggere dati da specifici indirizzi di memoria, utile per estrarre informazioni sensibili come una password, utilizzando comandi come **mdw**;
- **Debugging del Firmware**: Con OpenOCD, si può eseguire il debugging del codice, monitorando e manipolando l'esecuzione. Comandi come breakpoint, info functions, e disassembly permettono di analizzare funzioni e registri durante l'esecuzione, per comprendere come il firmware interagisce (ad esempio, nel confronto di password).
- **Interazione con il Dispositivo**: Utilizzando UART, è possibile interagire con il dispositivo tramite comandi come screen per inviare input (come la password) e osservare come il firmware la gestisce;
- **Controllo della Memoria e Sblocco Sicurezza**: Dopo aver esaminato i registri e la memoria, è possibile manipolare i valori per forzare il dispositivo a sbloccare funzionalità di sicurezza, come l'autenticazione, permettendo l'accesso al sistema o l'esecuzione di altre operazioni.

## 5.5. PROTEGGERE LE COMUNICAZIONI A BASSO LIVELLO

### PROTEZIONI DELLE COMUNICAZIONI SERIALI:

Un **Security Controller (SC)** è utilizzato per garantire **performance stabili** e **veloci** nei protocolli di comunicazione (come **UART**, **I2C**, **SPI**) e per fornire una **sicurezza elevata** tramite un **certificato di sicurezza** (ad esempio **CC EAL 5+**), combinato con una **performance crittografica avanzata**.

L'**SC** è un modulo hardware discreto che può essere programmato per eseguire funzioni di sicurezza specifiche, operando su **credenziali crittografiche** memorizzate in **memoria protetta** o generate direttamente in **hardware**.

L'**SC** fornisce meccanismi di protezione contro attacchi **locali** e **fisici**, come il **monitoraggio delle linee di bus**. Per garantire una resistenza avanzata agli attacchi, gli SC possono includere **circuiti integrati sicuri** con **doppio processore**, **bus di comunicazione criptati** e **coprocessori**.

La crittografia interna assicura che **nessun dato in chiaro** venga mai lasciato accessibile, aumentando significativamente la sicurezza contro i tentativi di manomissione.

I **diritti di accesso** al **firmware**, al **sistema operativo** dell'utente e alle **applicazioni** nelle **memorie** sono controllati e applicati dall'**unità di gestione della memoria (MMU)**.

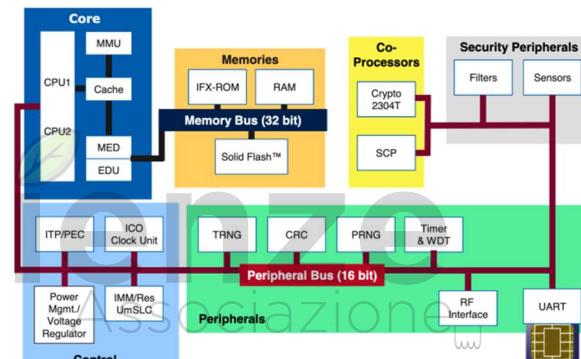
I due **processori** della **CPU** (CPU1 e CPU2) si controllano a vicenda per rilevare **guasti** e mantenere l'integrità dei dati. Un **comparatore** verifica se i calcoli sono stati eseguiti senza errori, consentendo la **rilevazione degli errori** anche durante l'elaborazione. La CPU, composta da due parti, funziona come un'unica entità ed è **più veloce** rispetto a una CPU standard alla stessa frequenza di clock.

La **CPU** accede alla memoria tramite l'unità integrata di **Crittografia e Decrittografia della Memoria (MED)**, che trasferisce i dati dallo schema di crittografia della memoria allo schema di crittografia della CPU senza decrittografare i dati in forma intermedia. L'**unità di rilevamento degli errori (EDU)** gestisce automaticamente la **rilevazione degli errori** nelle memorie, individuando eventuali trasferimenti di dati errati tra le memorie tramite il confronto dei **codici di errore**.

Il **blocco di memoria** comprende **ROM**, **RAM** e **flash NVM**. Tutti i dati del blocco di memoria sono **crittografati** e ogni tipo di memoria è dotato di un **codice di rilevamento degli errori (EDC)**. La **NVM** (memoria non volatile) è inoltre equipaggiata con un **codice di correzione degli errori (ECC)**. La **ROM non volatile** contiene le **parti del firmware** ed è accessibile solo dalla **CPU**, mentre la **RAM** è una memoria **volatile** utilizzata dal **core**.

Il **blocco coprocesso** contiene un **coprocesso** per il calcolo degli algoritmi **asimmetrici** come **RSA** e **Elliptic Curve (EC)**, e un altro per i calcoli con **dual-key** o **triple-key triple-DES** e **AES**.

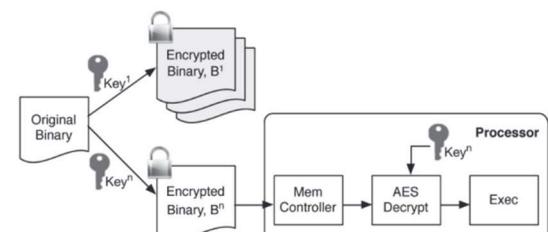
Il **blocco di periferiche di sicurezza** include un piccolo set di **sensori** e **filtr** per rilevare deviazioni eccessive dal **range operativo** specificato, senza essere troppo sensibile. Questi sensori non sono necessari per la sicurezza del chip, ma migliorano la **robustezza** complessiva del sistema.



### INSTRUCTION SET RANDOMIZATION:

**Instruction Set Randomization (ISR)** è un approccio generale per proteggere i sistemi da attacchi di **code-injection** (iniezione di codice), alterando casualmente le istruzioni utilizzate dalla macchina host, dall'applicazione o dall'esecuzione.

Ad esempio, l'**opcode** 0xa potrebbe rappresentare l'istruzione **XOR** in una specifica applicazione, ma essere **non valida** in un'altra. Questo impedisce a un attaccante di

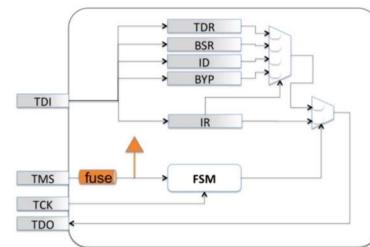


utilizzare la stessa **vulnerabilità** su **più target**.

Le implementazioni di **ISR** (Instruction Set Randomization) generalmente "emulano" l'**ISA** (Instruction Set Architecture) casuale utilizzando la **crittografia**. Il codice viene **crittografato** a livello di **binario** e **decrittografato in memoria** prima dell'esecuzione.

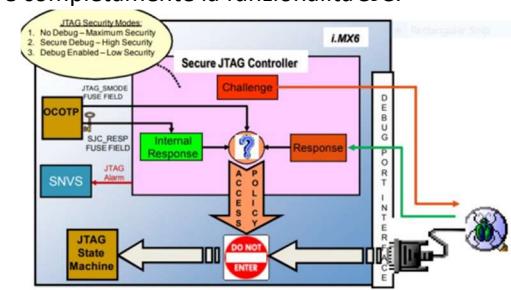
### JTAG PROTECTION:

Un modo per proteggere un chip dall'**exploitation** di **JTAG** è **disabilitare completamente l'accesso JTAG**. Questo può essere fatto disabilitando il segnale **TMS**, mettendo permanentemente la macchina a stati di **JTAG** nello **stato Test-Logic-Reset**, disabilitando così tutte le funzioni JTAG senza possibilità di riattivarle. Una soluzione migliore è far sì che il dispositivo sia in uno stato "**bloccato**" di produzione, relativamente immune agli attacchi, ma con alcune **strumentazioni sbloccate** quando necessario per l'esecuzione di analisi di **root-cause**.

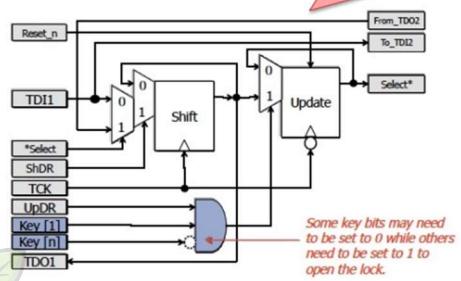


Un approccio più flessibile è rappresentato dal **Controller JTAG della serie i.MX6 (SJC)**, che regola l'accesso JTAG. I **mode di sicurezza JTAG** sono tre e possono essere configurati utilizzando **eFuse** programmabili una sola volta (OTP), che vengono "bruciati" dopo la produzione del dispositivo. Oltre a questi tre mode, c'è anche l'opzione di disabilitare completamente la funzionalità **SJC**.

L'uso di soluzioni **PKI** (Public Key Infrastructure) non è praticabile a causa della grande quantità di calcolo e latenza richiesti. Un meccanismo **challenge-response** è più efficiente e supportato da un **PUF** (Physical Unclonable Function) e un **hash**. Per l'autenticazione e il controllo degli accessi, il **SJC** invia una sfida all'utente, alla quale corrisponde una risposta corretta. Se la risposta fornita non è corretta, i messaggi dell'utente non vengono inoltrati alla macchina a stati **JTAG**.



Un metodo simile può essere implementato internamente utilizzando un **bit di inserimento segmento di blocco (LSIB)**, che può essere aperto solo quando sono presenti **valori predefiniti** corrispondenti a una **chiave** in specifici bit della **catena**.



# CoScienze

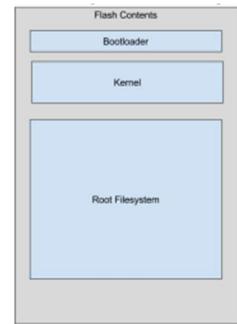
Associazione

## 6. HACKING ED EMULAZIONE DEL FIRMWARE

### HACKING DEL FIRMWARE (FIRMWARE HACKING):

Il firmware è un componente fondamentale per i dispositivi IoT, poiché permette di eseguire operazioni di ricerca e sfruttamento senza avere accesso fisico diretto al dispositivo. Si tratta di un dato binario memorizzato nella sezione non volatile del dispositivo, che consente al dispositivo di eseguire compiti diversi e gestire i componenti hardware necessari per il suo funzionamento.

Il bootloader è responsabile dell'inizializzazione della memoria RAM, delle porte seriali, e del kernel, mentre il kernel stesso funge da intermediario tra l'hardware e il software. I dispositivi IoT utilizzano diversi tipi di file system, come SquashFS, che è un file system in sola lettura e compresso, e vengono anche utilizzati vari tipi di compressione come LZMA, Gzip e Zlib.



Esistono diverse modalità per ottenere il firmware di un dispositivo IoT, tra cui il download online, l'estrazione fisica dal dispositivo o l'intercettazione dell'aggiornamento del firmware durante una trasmissione **Over The Air (OTA)**. Una volta ottenuto il firmware, si può estrarre il file system utilizzando strumenti come "dd" o "Binwalk", che automatizza il processo di estrazione e permette di analizzare i contenuti per identificare valori sensibili, come credenziali **hardcoded** o **backdoor**.

Se il firmware è cifrato, può essere necessario eseguire un'analisi con strumenti come hexdump o utilizzare script di decriptazione, come nel caso di cifrature XOR. Dopo aver estratto il file system, strumenti come radare2 possono essere utilizzati per analizzare i binari e identificare vulnerabilità come buffer overflow.

```
pi@ubuntu:~/Downloads$ binwalk -t dvrfl.bin
DECIMAL      HEXADECIMAL   DESCRIPTION
-----      -----      -----
0            0x0          BIN-HDR: board ID: 1550, hardware version: 4701, firmware version: 1.0.0, build date: 2012-02-08
32           0x20         TRX firmware header, little endian, image size: 7753728 bytes, CRC32: 0x436822f6, flags: 0x0, version: 1, header size: 28 bytes, loader offset: 0x1c, linux kernel offset: 0x192708, rootfs offset: 0x0
60           0x3c         gzip compressed data, maximum compression, has original file name "dvrfl" from Unix, last modified: 2016-03-09 08:08:31
1648424     0x192728    Squashfs filesystem, little endian, non-standard signature, version 3.0, size: 6099215 bytes, 447 inodes, blocksize: 65536 bytes, created: 2016-03-10 04:34:22
```

### EMULAZIONE DEL FIRMWARE (FIRMWARE EMULATION):

Inoltre, è possibile emulare il firmware per testarlo in ambienti virtualizzati, utilizzando strumenti come Qemu per eseguire firmware destinati ad architetture diverse. L'emulazione del firmware consente di eseguire attacchi di rete, eseguire il debug e analizzare le interfacce web.

### INSERIMENTO DI BACKDOOR NEL FIRMWARE (BACKDOORING FIRMWARE):

Infine, la backdooring è una tecnica di attacco che prevede l'inserimento di una backdoor nel firmware, modificando il file system estratto per consentire l'accesso remoto al dispositivo. Questa modifica può essere automatizzata utilizzando strumenti come il Firmware Mod Kit. Un altro approccio per identificare vulnerabilità nel firmware è l'uso di script automatizzati, come Firmwalker, per cercare stringhe interessanti che potrebbero indicare falle di sicurezza.



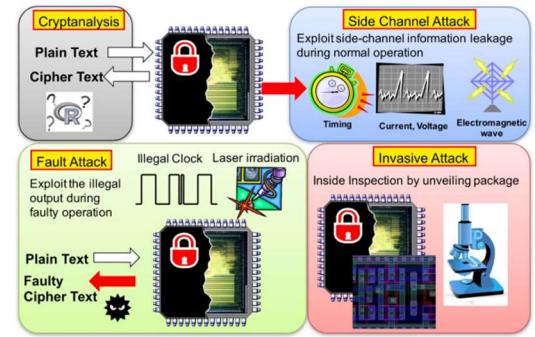
## 7.1. ATTACCHI FISICI

### ATTACCHI NON INVASIVI:

Gli **attacchi non invasivi** non richiedono l'accesso fisico diretto ai dispositivi; quindi, non lasciano segni evidenti di manomissione e non danneggiano i componenti. Questi attacchi sfruttano principalmente il comportamento fisico o il funzionamento del dispositivo per raccogliere informazioni sensibili, come chiavi segrete o password. Tra gli strumenti utilizzati ci sono multimetri digitali, stazioni di saldatura, oscilloscopi, analizzatori logici, schede di prototipazione e simulatori FPGA.

Le principali tipologie di attacchi non invasivi sono:

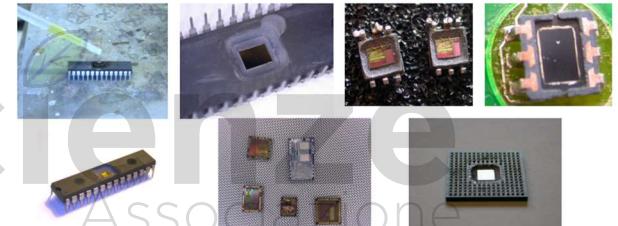
- **Attacchi a canale laterale (Side-channel attacks):** Questi attacchi si basano sull'analisi di informazioni non direttamente legate ai dati in ingresso o in uscita, ma piuttosto a come il sistema esegue il calcolo (per esempio, il tempo di esecuzione di un'operazione o il consumo di energia);
- **Attacchi di temporizzazione:** si analizza quanto tempo impiega il dispositivo per eseguire operazioni particolari, ad esempio, il tempo di decrittazione in un algoritmo di cifratura come RSA. Le variazioni nel tempo di esecuzione possono rivelare informazioni sensibili, come la chiave segreta;
- **Analisi del consumo di potenza:** misurando il consumo energetico durante l'esecuzione di operazioni, un attaccante può dedurre informazioni sul funzionamento interno del dispositivo, come la sequenza di istruzioni eseguite. Le tecniche principali includono l'analisi di potenza semplice (SPA) e l'analisi di potenza differenziale (DPA);
- **Analisi delle emissioni elettromagnetiche (EMA):** in modo simile all'analisi del consumo di potenza, si misurano le emissioni elettromagnetiche generate dal dispositivo mentre esegue le operazioni. Questo tipo di attacco può essere effettuato con strumenti come sonde magnetiche che rilevano le emissioni durante l'elaborazione dei dati.
- **Glitching (iniezione di guasti):** Gli attacchi di glitching implicano manipolare intenzionalmente l'ambiente operativo del chip, come la fornitura di clock o di potenza, per indurre errori nei calcoli. Questi attacchi possono essere utilizzati per bypassare meccanismi di sicurezza o rivelare informazioni come chiavi di cifratura;
- **Brute force (attacchi di forza bruta):** Questi attacchi cercano di forzare la scoperta di una password o di una chiave segreta provando sistematicamente tutte le combinazioni possibili. Anche se molto potenti, sono resi impraticabili dalla lunghezza delle chiavi moderne.



### ATTACCHI INVASIVI:

Gli **attacchi invasivi** richiedono l'accesso fisico ai dispositivi, lasciando tracce visibili di manomissione e potenzialmente danneggiando i chip. Gli strumenti utilizzati per questi attacchi comprendono stazioni di saldatura per la manipolazione dei chip, microscopi ad alta risoluzione per l'analisi dei circuiti, e strumenti avanzati come i microscopi elettronici a scansione (SEM) e i sistemi FIB (Focused Ion Beam).

Le principali tecniche di attacco invasivo includono:



- **Decapsulazione:** consiste nell'eliminare il rivestimento protettivo del chip per accedere ai circuiti interni. Questo può essere fatto manualmente con acidi come l'acido nitrico o mediante processi automatici che utilizzano miscele di acidi;
- **Immagine ottica e reverse engineering:** l'uso di microscopi ottici per analizzare il chip a livello microscopico, studiando la disposizione dei componenti e la loro funzionalità. Questo può essere un passo preliminare per decifrare il funzionamento interno di un chip.
- **Microprobing:** questa tecnica impiega sonde estremamente sottili per sondare i circuiti interni del chip e monitorare i segnali elettronici o iniettare segnali di test, il che permette di estrarre informazioni, come chiavi segrete o dati sensibili dalla memoria;
- **FIB (Focused Ion Beam):** un metodo avanzato che utilizza fasci di ioni focalizzati per fare incisioni di alta precisione su chip, permettendo di eseguire modifiche dirette ai circuiti, come il taglio di connessioni o l'aggiunta di nuovi contatti per il probing.

### ATTACCHI SEMI-INVASIVI:

Gli **attacchi semi-invasivi** cercano di sfruttare una via di mezzo tra i metodi non invasivi e invasivi, cercando di ottenere informazioni senza danneggiare troppo il dispositivo. Questi attacchi richiedono comunque un accesso fisico al chip, ma tendono ad essere meno distruttivi rispetto agli attacchi invasivi pur rivelandosi comunque efficaci.

Le principali tecniche di attacco semi-invasivo includono:



- **Immagine infrarossa e analisi delle emissioni ottiche:** utilizzo di camere a infrarossi o microscopi ad alta risoluzione per analizzare i circuiti interni senza danneggiarli fisicamente. L'infrarosso permette di visualizzare aree del chip che sono altrimenti non visibili;
- **Photon probing e LIVA (Light-induced voltage alteration):** si utilizzano laser per iniettare energia nei chip e alterare il comportamento dei transistor in modo tale da ottenere informazioni sugli stati logici dei circuiti interni. Questa tecnica può essere usata per l'analisi dei dati memorizzati o per forzare il sistema a cambiare stato.
- **Iniezione di guasti ottici:** l'uso di luce o fotoni per alterare lo stato dei chip e causare malfunzionamenti temporanei, rivelando informazioni non disponibili attraverso metodi puramente elettronici. In particolare, i laser possono indurre guasti nei sistemi a memoria flash o EEPROM;

Gli **attacchi invasivi** e **semi-invasivi** rimangono una minaccia considerevole, anche per i dispositivi con tecnologie avanzate (come i chip da 130 nm), ma la crescente complessità dei dispositivi e l'adozione di protezioni avanzate (ad esempio, crittografia hardware, difese

contro il glitching) stanno migliorando la sicurezza. Tuttavia, gli **attacchi non invasivi** continuano a rappresentare una seria minaccia, poiché non richiedono danni fisici e possono essere effettuati a distanza, anche su dispositivi virtualizzati o protetti da misure di sicurezza avanzate. I progressi nei metodi semi-invasivi stanno anche sfidando la sicurezza dei chip moderni, mantenendo alta la necessità di migliorare le difese contro queste tecniche.

## 7.2. PROTEGGERE I DISPOSITIVI IOT DAGLI ATTACCHI FISICI

### TAMPER RESISTANCE:

La **tamper resistance** (resistenza alla manomissione) si riferisce a tecniche di progettazione e imballaggio di dispositivi elettronici per rendere difficile manomettere i dispositivi stessi, oppure per renderlo evidente quando ciò accade. Le soluzioni tipiche includono:

- **Custodie in acciaio rinforzato;**
- **Serrature;**
- **Incapsulamento e "potting"** (riempimento con materiale protettivo);
- **Viti di sicurezza;**
- **Canali di flusso d'aria sigillati;**

La maggior parte di queste soluzioni fornisce una **evidenza di manomissione**, cioè permettono di osservare facilmente se il dispositivo è stato manipolato fisicamente. Tuttavia, la **tamper evidence** non è invulnerabile e può essere aggirata in molti casi.

### LIVELLI DI TAMPER RESISTANCE:

I dispositivi di protezione contro la manomissione sono classificati in **livelli di protezione** che vanno da ZERO (nessuna protezione) a HIGH (massima protezione). Ogni livello rappresenta un grado di difficoltà nell'effettuare un attacco, ma anche un aumento del costo e del tempo necessario per manomettere il dispositivo.

#### ▪ **Livello ZERO (nessuna protezione speciale):**

- **Descrizione:** Dispositivi come microcontrollori o FPGA con memoria ROM esterna;
- **Caratteristiche:** Nessuna protezione speciale, facilmente accessibili e vulnerabili;
- **Costo e tempo di attacco:** Molto basso, attacchi che richiedono da minuti a ore.

#### ▪ **Livello BASSO:**

- **Descrizione:** Microcontrollori con algoritmi di accesso proprietari o IC contrassegnati;
- **Caratteristiche:** Alcune protezioni di base, facilmente aggirabili con strumenti minimi;
- **Costo e tempo di attacco:** Basso, tempo di attacco che varia da ore a giorni.

#### ▪ **Livello MOD (Moderato Basso):**

- **Descrizione:** Microcontrollori con protezione di sicurezza, dongle hardware a basso costo;
- **Caratteristiche:** Protezione contro attacchi a basso costo;
- **Costo e tempo di attacco:** Moderato, attacco che richiede da giorni a settimane.

#### ▪ **Livello MOD (Moderato):**

- **Descrizione:** Smartcard, microcontrollori ad alta sicurezza, ASIC, CPLD, dongle hardware;
- **Caratteristiche:** Richiesta di strumenti e competenze speciali per l'attacco;
- **Costo e tempo di attacco:** Alto, tempo di attacco che va da settimane a mesi.

#### ▪ **Livello MODH (Moderato Alto):**

- **Descrizione:** i-Buttons sicuri, FPGA sicuri, smartcard avanzate, ASIC personalizzati;
- **Caratteristiche:** Protezione estremamente curata e tecnologie avanzate di sicurezza;
- **Costo e tempo di attacco:** Molto alto, attacco che richiede mesi o anni.

#### ▪ **Livello ALTO:**

- **Descrizione:** Equipaggiamento militare e bancario ad alta sicurezza;
- **Caratteristiche:** Ricerche e attacchi complessi da parte di specialisti;
- **Costo e tempo di attacco:** Estremamente alto, attacco che può durare anni.

### VALUTAZIONE DELLA SICUREZZA:

La divisione in livelli di protezione (da ZERO a HIGH) è relativa, e non tutte le soluzioni sono perfette. Un dispositivo progettato per essere altamente sicuro potrebbe avere delle vulnerabilità sconosciute, mentre un dispositivo non pensato per la sicurezza potrebbe essere comunque molto difficile da attaccare. La **tecnologia in continua evoluzione** può portare ad attacchi più economici e più efficaci, riducendo di fatto il livello di protezione.

Una corretta **valutazione della sicurezza** è essenziale per determinare se un prodotto soddisfa i requisiti di sicurezza. Devono essere eseguiti test completi e revisioni di progettazione per identificare possibili difetti di sicurezza.

### PROTEZIONI AGGIUNTIVE:

Le tecnologie moderne di **fabbricazione dei chip** hanno reso molto più difficile manomettere i dispositivi:

- **Pianificazione:** La fabbricazione a livello di 0,5 µm o inferiore ha reso i chip più difficili da attaccare;
- **Logica di adesione:** La progettazione di circuiti logici complessi rende più difficile il reverse engineering;
- **Strati multipli di metallo:** Creano barriere fisiche che ostacolano l'accesso diretto ai circuiti;
- **Miniaturizzazione:** L'integrazione di transistor più piccoli e chip più compatti rende l'attacco più difficile e costoso;

Altri strumenti di protezione includono:

- **Sensori di tensione, temperatura e frequenza;**
- **Protezione dell'accesso alla memoria e coprocessori crittografici;**

- *Progettazione con logica asincrona e dual-rail;*

## PROBLEMI DI SICUREZZA E FALLIMENTI NOTI:

Molti dispositivi con protezioni avanzate hanno avuto **problemi di sicurezza** legati a difetti di progettazione o vulnerabilità non corrette.

Alcuni esempi includono:

- **Microchip PIC:** Bug nel fuse di sicurezza che poteva essere ripristinato senza cancellare la memoria;
- **Smartcard Hitachi:** Fuoriuscita di informazioni sensibili a causa di un errore nel CD di un prodotto;
- **FPGA Actel:** Bug nel software di programmazione che impediva la protezione dei dispositivi;
- **Memoria sicura Dallas SHA-1:** Bug di inizializzazione in fabbrica che disabilitava le funzioni di sicurezza

## MINACCE INTERNE E DIPENDENZE DA FORNITORI ESTERNI:

Anche i difetti provenienti da **insider** (personale interno) o **subappaltatori** possono causare vulnerabilità di sicurezza. È necessario monitorare attentamente le **fasi di progettazione e produzione** per prevenire la creazione di **backdoor** o trojan (software dannoso inserito intenzionalmente). Ad esempio, la **sicurezza attraverso l'oscurità** (security through obscurity), in cui la protezione si basa sul nascondere i dettagli, è una strategia che non sempre si rivela efficace.

In conclusione, la protezione contro la manomissione è una componente cruciale nella progettazione dei dispositivi elettronici, e la sicurezza deve essere costantemente valutata e aggiornata in base ai progressi tecnologici e alle minacce emergenti.



CoScienze  
Associazione

## 8. INTRODUZIONE ALLE PUF

### PUF:

Le **Physical Unclonable Functions (PUF)** si ispirano al concetto di biometria, come ad esempio l'identificazione tramite impronta digitale, per identificare oggetti, sistemi e persone sfruttando caratteristiche fisiche intrinseche di circuiti elettronici. Un PUF funziona come una "funzione" che, quando riceve una certa "sfida" (challenge), restituisce una "risposta" (response). La relazione tra sfide e risposte è definita come "comportamento del PUF", ed è generalmente imprevedibile e influenzata da fattori fisici casuali.

Il processo di utilizzo di un PUF avviene in due fasi principali:

- **Fase di registrazione:** Vengono raccolti diversi "challenge-response pair" (CRP) e memorizzati in un database;

- **Fase di verifica:** Viene applicata una sfida casuale al PUF e la risposta ottenuta viene confrontata con quella nel database; Un aspetto fondamentale del PUF è la **distanza intra-risposta**, che misura la ripetibilità di una risposta dalla stessa istanza del PUF usando la stessa sfida, e la **distanza inter-risposta**, che misura la differenza tra risposte di PUF differenti usando la stessa sfida. Le risposte vengono misurate tramite la distanza di Hamming.

Un PUF è considerato **unclonable** (non clonabile), ma il termine "unclonable" non è definito con precisione. Esistono due definizioni: una **matematica**, in cui un clone è una costruzione che replica il comportamento CRP di un PUF, e una **fisica**, dove un clone è una copia fisica che riproduce perfettamente comportamento e aspetto del PUF.

Inoltre, un PUF può essere definito come una **funzione probabilistica**, dato che il suo comportamento è influenzato da variabili casuali e imprevisti legati al processo di produzione. I PUF possono essere suddivisi in due categorie principali: **PUF intrinseci**, che misurano caratteristiche casuali derivate dai processi di produzione, e **PUF esplicativi**, che includono una randomizzazione artificiale durante la produzione.

I PUF intrinseci presentano dei vantaggi significativi, come il fatto che le caratteristiche casuali non possono essere modificate nemmeno dai produttori, offrendo un livello di sicurezza maggiore. Inoltre, i PUF possono essere classificati in **forti** e **deboli** in base alla loro capacità di resistere agli attacchi. Un **PUF forte** rende difficile per un avversario prevedere le risposte, anche dopo un lungo periodo di osservazione.

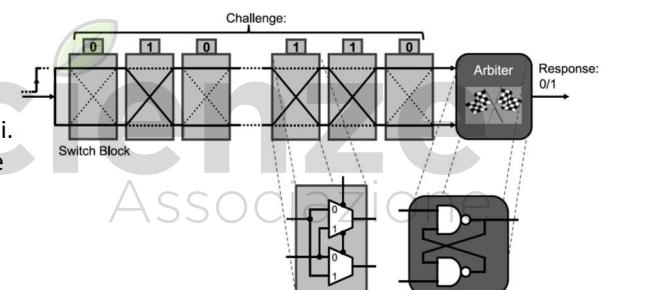
I PUF basati su **silicio** sono i più comuni e utilizzano variazioni casuali nel ritardo dei circuiti digitali o nei parametri dei dispositivi di memoria per generare risposte uniche.

### ARBITER PUF:

L'**Arbiter PUF** è un tipo di PUF basato su ritardi e utilizza una condizione di "race" tra due percorsi digitali su un chip in silicio. Ogni blocco di commutazione, costituito da due multiplexer 2-1, può propagare un segnale di ingresso su due percorsi configurabili, che possono essere diritti o incrociati. Il circuito *arbiter* (un latch SR) determina quale dei due percorsi è più veloce e restituisce un valore binario in base al risultato.

Il ritardo effettivo di ogni percorso non sarà mai esattamente uguale a causa delle variazioni casuali nel processo di produzione del silicio. Ciò rende il comportamento del PUF specifico per ogni dispositivo. Se i due ritardi sono molto simili, si può verificare uno stato metastabile, in cui il circuito *arbiter* non produce una risposta determinata, ma una risposta casuale. Un PUF di alta qualità richiede che i percorsi di ritardo siano simmetrici e che il circuito arbiter sia imparziale.

Tuttavia, questo tipo di PUF è vulnerabile agli attacchi di modellizzazione. Se un attaccante raccoglie abbastanza CRP, può costruire un modello matematico per predire correttamente le risposte anche senza avere accesso diretto al PUF.



### RING OSCILLATOR PUF:

Il **Ring Oscillator PUF** è un altro tipo di PUF basato su ritardi e sfrutta le variazioni casuali della frequenza di oscillazione di un circuito oscillante. L'oscillatore è composto da un numero dispari di porte NOT disposte a cerchio, il cui segnale di uscita oscilla tra due livelli di tensione. Le variazioni casuali nel ritardo dei componenti digitali causano variazioni nella frequenza di oscillazione.

Un contatore di frequenza misura il numero di cicli dell'oscillazione in un intervallo di tempo fisso, producendo un valore che rappresenta la frequenza dell'oscillatore. Per ridurre l'influenza di variazioni ambientali (come temperatura e tensione), si può utilizzare una tecnica di misurazione compensata, che confronta due oscillatori sullo stesso dispositivo.

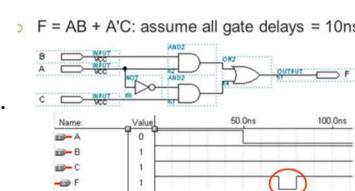
Anche questo tipo di PUF è vulnerabile ad attacchi di modellizzazione, e le risposte devono essere quantizzate per essere utilizzate come stringhe di bit.

### GLITCH PUF:

Il **Glitch PUF** sfrutta il fenomeno dei "glitch" nei circuiti combinatori.

Un circuito combinatorio, che non ha uno stato interno, può produrre effetti di transizione quando l'ingresso cambia, generando dei "glitch". La forma, il numero e l'occorrenza di questi glitch sono influenzati da variazioni casuali nel processo di produzione del circuito.

Per misurare il comportamento di glitch, il segnale di uscita del circuito può essere collegato a un flip-flop, che memorizza la parità del numero di glitch che si sono verificati. Per migliorare l'affidabilità delle risposte, viene utilizzata una tecnica di mascheramento dei bit instabili, ignorando quelli che non producono un valore stabile su più



- An input change from ABC = 111 to 011 results in a glitch → output changes from 1 → 0 → 1 again
- A static hazard: result should have stayed statically at 1

misure consecutive.

### SRAM PUF:

La **SRAM** (*Static Random-Access Memory*) è una tecnologia di memoria digitale basata su circuiti bistabili, in cui una cella SRAM tipica è composta da sei transistor (MOSFETs) in un'implementazione CMOS. Ogni cella SRAM è costituita da due inverter CMOS che si retroalimentano tra loro, stabilizzando ciascun inverter nel proprio stato. Questi inverter possono essere configurati per memorizzare un bit binario.

Il comportamento dell'SRAM è diverso rispetto alla **DRAM** (Dynamic RAM), in quanto mantiene le informazioni finché è alimentata, mentre la **DRAM** necessita di rinfreschi periodici. Quando il sistema SRAM è in modalità standby, le linee di parola (wordlines) sono basse, e i transistor di accesso sono spenti. Quando si scrive un dato, viene applicata una tensione sulle linee dei bit; quindi, i transistor di accesso si accendono e i dati vengono immagazzinati.

In termini logici, una cella SRAM è bistabile, e conserva un bit in uno dei suoi due stati stabili. La struttura di retroazione degli inverter CMOS genera tre possibili punti operativi, di cui solo due sono stabili, mentre uno è metastabile. Questo stato metastabile è influenzato dalle piccole deviazioni causate da rumore casuale, il che fa sì che la cella SRAM si sposti rapidamente verso uno dei due stati stabili, in modo casuale.

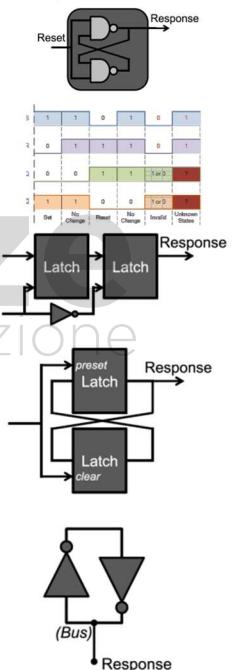
Il principio operativo di un **SRAM PUF** si basa sul comportamento transitorio della cella SRAM durante l'accensione. Quando alimentata, la cella si stabilisce in uno dei suoi due punti operativi stabili, ma non è immediatamente chiaro quale. La differenza di "forza" tra i MOSFETs nei due inverter, causata da variazioni casuali nel processo di produzione del silicio, determina quale stato sarà preferito, rendendo ogni cella unica.

Il **core** dell'SRAM PUF è una matrice bidimensionale di celle di memoria, dove le righe orizzontali sono chiamate "wordlines" e le due linee verticali attraverso ogni cella sono la "bitline" e la "complement bitline". Ogni cella può essere selezionata da un decodificatore di righe/colonne con un indirizzo di ingresso esterno. Una volta che il **SRAM PUF** è impostato in modalità PUF, viene riavviato e alimentato, e a causa delle variazioni casuali nei MOSFETs, ogni cella genera un bit di risposta. Una sfida in ingresso è applicata alle wordlines delle celle selezionate, accendendo i transistori di accesso, e i bit di risposta vengono letti dalle bitlines.

### ALTRI PUF BASATI SULLA MEMORIA:

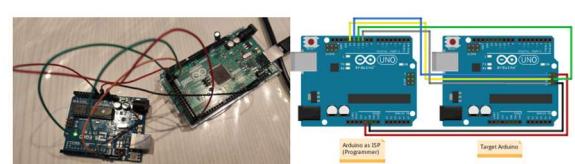
Oltre alle celle SRAM, esistono altri elementi di memorizzazione digitali avanzati basati sul principio della bistabilità, che mostrano comportamenti simili ai PUF basati su mismatch casuali tra dispositivi incrociati nominalmente abbinati.

- **Latch PUF**: si basa su un latch SR costituito da due NOR-gate incrociati. Il comportamento del PUF dipende dalle differenze interne tra i NOR-gate, e può essere riattivato ogni volta che il dispositivo è alimentato. Ha il vantaggio di non dipendere da una condizione di accensione, e può essere riattivato in qualsiasi momento quando il dispositivo è alimentato. Questo consente di generare e misurare più valutazioni della risposta, migliorando l'affidabilità tramite tecniche post-elaborazione come il voto di maggioranza;
- **Flip-Flop PUF**: i flip-flop D sono composti da strutture di latch che causano un comportamento simile a un PUF all'avvio. Tuttavia, l'entropia nei valori di avvio è limitata a causa di un forte bias che favorisce lo stato iniziale zero;
- **Butterfly PUF**: Per le piattaforme FPGA, gli SRAM PUF sono difficili da implementare poiché gli array SRAM vengono svuotati immediatamente dopo l'accensione. Il Butterfly PUF simula il comportamento di una cella SRAM utilizzando latch trasparenti incrociati nel logico riconfigurabile FPGA, ma l'implementazione di questi PUF non è triviale a causa delle limitazioni nei percorsi di routing;
- **Buskeeper PUF**: Questo tipo di PUF è basato su celle buskeeper, che vengono utilizzate per mantenere l'ultimo valore su una linea di bus. È un latch con una forza di guida debole e il comportamento PUF è causato dal mismatch casuale tra i dispositivi incrociati nel circuito. Il vantaggio di questa costruzione è la sua compattezza rispetto ai latch o flip-flop tipici.



### SRAM PUF CON ARDUINO:

È possibile utilizzare un **SRAM PUF con le schede Arduino**, dove il contenuto della memoria SRAM può essere letto tramite un bootloader personalizzato. Il bootloader preesistente sulle schede Arduino deve essere sostituito per ottenere questa funzionalità. Questo può essere fatto collegando due schede Arduino: una per programmare l'altra con il bootloader personalizzato. Una volta programmata, la scheda può essere accesa ripetutamente e il contenuto della memoria SRAM può essere letto e stampato tramite comunicazione seriale.



Il **SRAM PUF** sfrutta le deviazioni casuali causate dalle variazioni del processo di produzione nelle celle di memoria. Dopo la programmazione, il contenuto della memoria, che appare come dati casuali, può essere catturato e stampato. Questi dati possono essere utilizzati come una risposta unica basata sulle caratteristiche fisiche della SRAM, rendendola utile per scopi crittografici, come la generazione di chiavi.

### CHIAVI FISICAMENTE OFFUScate (POK):

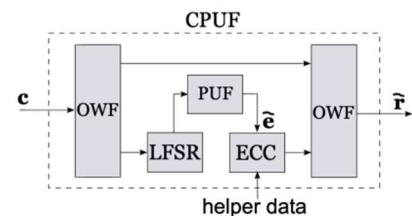
Una **POK** è una chiave fisica memorizzata in un modo che la rende difficile da estrarre tramite attacchi di probing. La chiave è solitamente combinata con l'output di un **PUF**, che non può essere facilmente scelto o previsto. L'output del PUF e la chiave possono essere combinati tramite un'operazione XOR e memorizzati in una memoria non volatile, come un EEPROM. La POK è progettata in modo che qualsiasi

attacco invasivo distrugga la chiave e lasci evidenza di manomissione, fornendo così sicurezza contro l'accesso non autorizzato.

### PUF CONTROLLATI (CPUF):

Un **PUF controllato (CPUF)** è un PUF che lavora in combinazione con altre primitive crittografiche per aggiungere ulteriori strati di sicurezza. In questa modalità, il PUF può essere accessibile solo tramite un algoritmo fisicamente legato al PUF stesso, impedendo l'accesso non autorizzato. Diversi metodi vengono utilizzati per rafforzare la sicurezza dei CPUF:

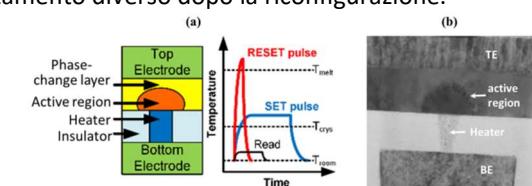
- **Funzioni hash crittografiche:** Queste possono essere utilizzate per generare le sfide per il PUF, impedendo attacchi di sfida scelta e rendendo più difficili gli attacchi di modellizzazione;
- **Algoritmi di correzione degli errori (ECC):** Vengono utilizzati per migliorare l'affidabilità delle risposte del PUF correggendo eventuali errori, rendendo la risposta finale altamente affidabile;
- **Post-elaborazione con funzioni hash:** Questo interrompe efficacemente il legame tra i dettagli fisici del PUF e le sue risposte, aggiungendo un ulteriore strato di sicurezza;



### PUF RICONFIGURABILI (RPUF):

I **PUF riconfigurabili (RPUF)** ampliano i PUF tradizionali aggiungendo la possibilità di riconfigurare il comportamento di sfida-risposta in modo casuale e preferibilmente irreversibile, portando a un nuovo PUF con comportamento diverso dopo la riconfigurazione.

- **Memoria a cambiamento di fase (NVRAM):** Questo tipo di PUF riconfigurabile utilizza celle di memoria a cambiamento di fase, che possono passare da uno stato amorfo a uno cristallino. Scrivere in questa memoria altera il suo stato fisico, e leggere la resistenza della cella fornisce una risposta unica. La possibilità di riscrivere queste celle cambia il comportamento della sfida-risposta in modo casuale ogni volta che la memoria viene riconfigurata;
- **PUF Riconfigurabili Logicamente (LRPUF):** Questi PUF utilizzano porzioni di memoria non volatile per memorizzare uno stato, con trasformazioni dipendenti dallo stato applicate alle risposte di un PUF classico.



### SVANTAGGI DEI PUF RICONFIGURABILI LOGICAMENTE:

La riconfigurazione di un **LRPUF** non è veramente irreversibile, poiché se uno stato precedente venisse ripristinato, il comportamento della sfida-risposta potrebbe ripetersi.

La sicurezza della riconfigurazione logica dipende dall'integrità della memoria di stato, che deve essere garantita da misure di protezione fisiche indipendenti. Tuttavia, la loro relativa facilità di costruzione rispetto ai più esotici PUF riconfigurabili fisicamente li rende immediatamente utilizzabili.

### IDENTIFICAZIONE BASATA SU PUF:

Un'identità può essere:

- **Inerente:** cioè basata su una caratteristica specifica di un'entità che emerge durante il processo di creazione dell'entità;
- **Assegnata:** come un codice a barre o una password.

L'inalterabilità del comportamento specifico di ogni entità è una delle condizioni principali affinché una costruzione venga definita come PUF. Le stringhe di bit uniche memorizzate in una memoria non volatile incorporata nel chip erano fino a poco tempo fa l'unico modo per identificare un chip specifico nelle interazioni digitali. Con la tecnologia silicon PUF, ora è possibile utilizzare anche caratteristiche uniche inerenti al chip di silicio per identificare l'entità.

Le tecniche di identificazione basate su identità assegnate e su identità inerenti generalmente funzionano in due fasi:

- **Fase 1 (provisioning/enrollment):** per le identità assegnate, consiste nell'attribuire a ogni entità una identità unica permanente; per le identità inerenti, consiste nel raccogliere le identità inerenti di ogni entità che deve essere identificata;
- **Fase 2 (identificazione):** entrambe le tipologie sono simili e consistono nell'entità che presenta la propria identità, assegnata o inerente, quando richiesta.

Le differenze tra la fase di provisioning e quella di enrollment evidenziano vantaggi pratici:

- Un'identità assegnata deve essere generata prima di essere assegnata a un'entità, richiedendo una fonte di casualità (come un generatore di numeri casuali veri o un contatore monotono). Le identità inerenti, invece, sono uniche di per sé, senza necessità di fonti di casualità;
- Quando si assegna un'identità, sono necessarie modifiche fisiche permanenti o almeno non volatili all'entità. Al contrario, le identità inerenti non richiedono capacità di archiviazione aggiuntive;
- L'enrollment (lettura di un'identità) è generalmente meno invasivo rispetto al provisioning (scrittura di un'identità); quindi può essere eseguito più rapidamente e con maggiore affidabilità. Tuttavia, non si ha il controllo diretto sui valori effettivi assunti dagli identificatori inerenti;
- Una caratteristica particolare degli identificatori inerenti è che mostrano un comportamento fuzzy e casuale. Si dice che una variabile casuale, come una risposta PUF o una caratteristica biometrica, mostri un comportamento fuzzy se:
  - Non è distribuita uniformemente;
  - Non è perfettamente riproducibile se misurata più volte;

Per le risposte PUF:

- Le caratteristiche fuzzy sono causate dalla natura fisica della loro generazione. I processi fisici casuali che introducono caratteristiche specifiche per entità durante la produzione non sono tipicamente distribuiti uniformemente;
- I meccanismi di valutazione della risposta di una costruzione PUF sono soggetti a rumore fisico e condizioni ambientali che causano

una riproducibilità non perfetta di una risposta PUF;

Le identità assegnate, invece, non sono generalmente fuzzy: il soggetto che assegna può garantire che le identità assegnate siano generate da una distribuzione uniforme, e un'entità può tipicamente riprodurre la sua identità assegnata con una riproducibilità quasi perfetta.

La fuzziness di una risposta PUF è mostrata chiaramente dalle distribuzioni delle distanze intra- e inter- risposte. Quando consideriamo vettori di risposta binari e la distanza di Hamming frazionaria come metrica di distanza, risposte perfettamente uniformemente casuali avrebbero una distanza inter di esattamente il 50%.

- Nessuna delle costruzioni PUF intrinseche esistenti soddisfa questa condizione, anche se alcune hanno una distanza inter-media molto vicina al 50%;
- Nessun PUF intrinseco mostra una riproducibilità perfetta con una distanza intra-fissa dello 0%, e molti sono riproducibili solo con una distanza intra-media del 10% o più.

Quando una distanza osservata tra le risposte della fase di enrollment e di identificazione ricade in questa regione di sovrapposizione, può essere il risultato di una distanza intra (indica che è la stessa entità) o di una distanza inter (indica che si tratta di un'entità diversa). Non c'è modo di distinguere tra i due casi.

Soglia di identificazione: viene impostata una soglia di distanza:

- Distanze inferiori o uguali a questa soglia sono considerate distanze intra tra risposte dalla stessa entità;
- Distanze superiori a questa soglia sono considerate distanze inter tra risposte di entità diverse. Un sistema di identificazione fuzzy basato su tale soglia pragmatica non è completamente affidabile al 100%.

Quando si confronta la risposta presentata da un'entità con una risposta della lista di enrollment, possono verificarsi quattro situazioni:

- L'entità presentata è la stessa dell'entità che ha prodotto la risposta iscritta, e riesce a riprodurre la risposta iscritta con una distanza intra inferiore alla soglia di identificazione. Accettazione corretta: l'entità è correttamente identificata;
- L'entità presentata è la stessa dell'entità che ha prodotto la risposta iscritta, ma non riesce a riprodurre la risposta con una distanza intra inferiore alla soglia di identificazione. Rifiuto falso: l'entità è erroneamente rifiutata;
- L'entità presentata non è la stessa dell'entità che ha prodotto la risposta iscritta, ma per caso produce una risposta la cui distanza inter rispetto alla risposta iscritta è inferiore alla soglia di identificazione. Accettazione falsa: l'entità viene erroneamente identificata;
- L'entità presentata non è la stessa dell'entità che ha prodotto la risposta iscritta, e produce una risposta la cui distanza inter è maggiore della soglia di identificazione. Rifiuto corretto: l'entità è correttamente rifiutata

Il **tasso di rifiuti falsi (FRR)** e il **tasso di accettazioni false (FAR)** sono entrambi indesiderabili in un sistema di identificazione pratico.

Il FAR esprime la sicurezza del sistema di identificazione: un basso FAR significa che c'è un basso rischio di identificazione errata che potrebbe portare a problemi di sicurezza.

Il FRR esprime la robustezza o usabilità del sistema, in quanto rappresenta il rischio di rifiutare erroneamente entità legittime, il che sarebbe molto impratico. Entrambi FAR e FRR devono essere il più piccoli possibile, ma è evidente che non possono essere minimizzati contemporaneamente. È necessario trovare un compromesso accettabile tra sicurezza e usabilità in base alla scelta del valore della soglia di identificazione.

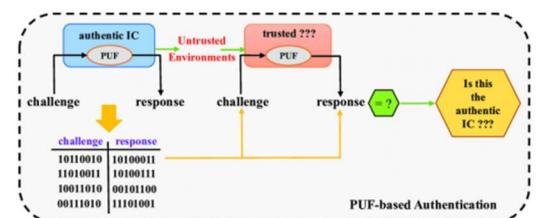
Quando un'entità vuole autenticarsi con un'altra parte, solitamente chiamata il verificatore, deve fornire, oltre alla semplice identificazione, anche una prova corroborante della sua identità, cioè una prova che può essere creata solo da quella particolare entità. Un'entità tipicamente dimostra al verificatore di possedere una chiave segreta che solo quell'entità può conoscere o possedere.

## AUTENTICAZIONE BASATA SU PUF:

Quando un'entità vuole autenticarsi con un'altra parte, solitamente chiamata il verificatore, deve fornire, oltre alla semplice identificazione, anche una prova corroborante della sua identità, cioè una prova che può essere creata solo da quell'entità particolare. Un'entità tipicamente dimostra di possedere una chiave segreta che solo quella entità può conoscere o avere. Inoltre, l'entità deve convincere il verificatore che è stata coinvolta attivamente, cioè al momento dell'autenticazione, nella creazione di tale prova.

Il protocollo di autenticazione basato su sfida-risposta PUF consiste in due fasi: enrollment e verifica:

- Prima della distribuzione, ogni entità passa attraverso la fase di enrollment da parte del verificatore. Il verificatore registra l'ID dell'identità e raccoglie un sottoinsieme significativo di coppie sfida-risposta che vengono memorizzate nel database del verificatore, indicizzate dall'ID dell'entità;
- Durante la fase di verifica, l'entità invia il proprio ID. Il verificatore cerca l'ID nel suo database e seleziona una coppia sfida-risposta. La sfida PUF viene inviata all'entità, che risponde con la risposta ottenuta, che viene verificata dal verificatore;



La correttezza di questo schema di autenticazione è garantita dal fatto che le risposte PUF sono riproducibili nel tempo, fino a una piccola distanza intra. Per la sicurezza dell'autenticazione, il verificatore si affida al fatto che i PUF sono unici e non clonabili, e solo il PUF genuino può riprodurre con alta probabilità una risposta vicina a una sfida precedentemente non osservata e casuale. Tuttavia, questo protocollo di base presenta diversi svantaggi:

- Le coppie sfida-risposta non possono essere riutilizzate per evitare attacchi di ripetizione, e una coppia utilizzata viene quindi rimossa dal database dopo che il protocollo è stato completato. Il verificatore deve memorizzare un gran numero di coppie per ogni entità, e mantenere un database così grande è un impegno significativo. Non tutti i PUF hanno un ampio insieme di sfide-risposte;
- Quando le coppie sfida-risposta memorizzate nel database di un'entità terminano, l'entità non può più essere autenticata dal verificatore;
- Il protocollo di base fornisce solo l'autenticazione di un'entità verso il verificatore; non supporta l'autenticazione reciproca senza estendere significativamente il protocollo di base.

Il protocollo di autenticazione reciproca basata su PUF è limitato dalla quantità di imprevedibilità nel comportamento della sfida-risposta di una singola istanza PUF, e aumentarla comporta un elevato costo di implementazione. Non è consigliabile divulgare pubblicamente le

coppie sfida-risposta durante le comunicazioni del protocollo, come fa il protocollo di base, poiché ogni coppia divulgata riduce significativamente l'imprevedibilità rimanente del comportamento del PUF. Nei tradizionali protocolli crittografici di autenticazione basati su tecniche a chiave simmetrica, un'entità non si autentica divulgando la propria chiave segreta, ma dimostra solo di conoscere la chiave segreta.

Nell'autenticazione basata su PUF, un'entità dimostra di possedere una istanza PUF mostrando di essere in grado di calcolare una valutazione di funzione che prende una risposta PUF imprevedibile come input, senza divulgare completamente la natura imprevedibile della risposta nel risultato di questa valutazione.

Poiché i PUF sono intrinsecamente rumorosi, vengono tipicamente combinati con fuzzy extractors, cioè una mappatura sicura che associa risposte simili di PUF allo stesso valore. Una funzione hash viene utilizzata per generare la prova, in quanto non divulgare alcuna informazione significativa sul valore della risposta. Grazie alla non clonabilità fisica dei PUF impiegati, un tentativo di impersonificazione di un'entità con un PUF diverso fallirà con alta probabilità. Il tasso di accettazione falsa (FAR) del protocollo complessivo dipende anche dalla resistenza alle collisioni della funzione hash utilizzata e potrebbe essere considerevolmente più alto. I nonce generati rispettivamente dall'entità e dal verificatore impediscono attacchi di ripetizione da entrambe le parti, introducendo freschezza nelle comunicazioni del protocollo. Un avversario che tenta di ripetere i messaggi del protocollo per impersonare il verificatore fallirà, così come un avversario che tenta di impersonare un'entità riproducendo messaggi registrati precedentemente.

## GENERAZIONE DI CHIAVI BASATA SU PUF:

Un presupposto indispensabile per la maggior parte delle implementazioni crittografiche è la capacità di generare, memorizzare e recuperare chiavi in modo sicuro. I requisiti minimi per una generazione e memorizzazione sicura delle chiavi sono:

- Una fonte di casualità per garantire che le chiavi appena generate siano imprevedibili e uniche;
- Una memoria protetta che memorizza in modo affidabile le informazioni sulla chiave, proteggendola completamente da accessi non autorizzati.

Tuttavia, implementare entrambe queste richieste non è semplice. La necessità di casualità imprevedibile è tipicamente soddisfatta applicando un generatore di numeri pseudo-casuali (PRNG) a seme. Tuttavia, tali generatori sono difficili da implementare correttamente.

Implementare una memoria protetta è una sfida di progettazione considerevole, che spesso comporta un aumento dei costi di implementazione e/o una riduzione delle possibilità applicative per garantire la sicurezza fisica della chiave memorizzata. Un generatore di chiavi basato su PUF cerca di affrontare entrambi i requisiti contemporaneamente, utilizzando un PUF per raccogliere una casualità statica ma unica per dispositivo, e trasformandola in una chiave crittografica.

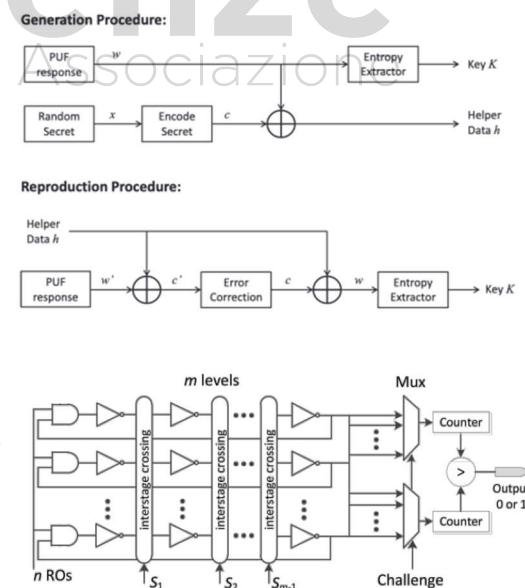
- Si evita l'uso di un PRNG, poiché la casualità è già intrinsecamente presente nel dispositivo utilizzato;
- Non è necessaria una memoria protetta non volatile, poiché la casualità utilizzata è considerata statica per tutta la durata del dispositivo e può essere misurata ripetutamente per rigenerare la stessa chiave dalle caratteristiche casuali, altrimenti illeggibili.

### Procedura di generazione:

Sia  $C[n, k, d]$  un codice di correzione degli errori. Dato  $w \in \{0,1\}^n$ , questa procedura sceglie  $x \in \{0,1\}^k$  casualmente e calcola  $c = C(x)$ , la codifica di  $x$ . I dati ausiliari sono  $h = w \oplus c$ , mentre la chiave è  $K = H(w)$ , dove  $H$  è una funzione hash.

### Procedura di riproduzione:

Dato un elemento  $w' \in \{0,1\}^n$ , una stringa di dati ausiliari  $h$ , e una funzione hash universale  $H$ , la procedura calcola prima  $c' = w' \oplus h$ . Poi decodifica  $c'$  per ottenere  $c$ , se  $\text{dist}(w, w') \leq t$ . Riproduce  $w = h \oplus c$  e restituisce la chiave  $K = H(w)$ .



La chiave condivisa è necessaria nelle comunicazioni multi-parte tra dispositivi diversi, per esempio, per stabilire un canale di comunicazione sicuro. I PUF tradizionali generano una chiave unica per ogni dispositivo, mentre CRO PUF è in grado di generare la stessa chiave condivisa per tutti i dispositivi.

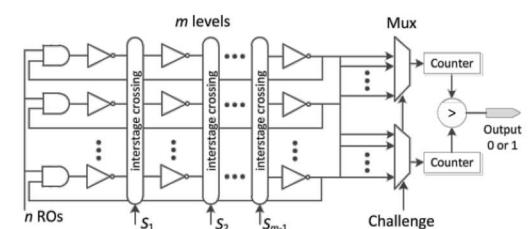
## CONDIVISIONE DI CHIAVI BASATA SU PUF:

La chiave condivisa è necessaria nelle comunicazioni multi-parte tra dispositivi diversi, per esempio, per stabilire un canale di comunicazione sicuro. I PUF tradizionali generano una chiave unica per ogni dispositivo, mentre CRO PUF è in grado di generare la stessa chiave condivisa per tutti i dispositivi.

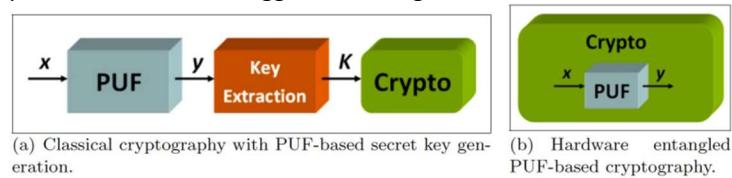
Il CRO PUF può generare la stessa chiave condivisa per tutti i dispositivi, evitando la trasmissione di una chiave segreta. Le informazioni di configurazione non sono segrete e possono essere memorizzate in SRAM. Questo approccio fornisce alta sicurezza e bassi costi per realizzare la condivisione della chiave tra più parti.

## CRITTOGRAFIA HARDWARE ENTANLED:

In alcune primitive crittografiche esistenti, la generazione di chiavi segrete dai PUF è progettata per scenari specifici. Tuttavia, i PUF possono essere anche completamente integrati nella stessa crypto-primitiva, un concetto noto come hardware entangled cryptographic primitives. Queste primitive sono senza chiave, in quanto non viene passata né memorizzata alcuna chiave segreta, né in memoria



volatile né non volatile, offrendo quindi una sicurezza maggiore contro gli attacchi.



## 9.1. INTRODUZIONE ALLA CRITTOGRAFIA

### CRITTOGRAFIA:

La **crittografia** riguarda la costruzione e l'analisi di protocolli che impediscono la lettura di messaggi privati e garantiscono vari aspetti della sicurezza delle informazioni, come la riservatezza dei dati, l'integrità dei dati, l'autenticazione e la non ripudiabilità.

La **cifratura** è la conversione delle informazioni da uno stato leggibile a un apparente nonsenso, utilizzando una funzione matematica e una stringa casuale di bit difficile da indovinare. La crittografia è ampiamente classificata in **Crittografia a chiave simmetrica** e **Crittografia a chiave asimmetrica**.

Nella crittografia simmetrica troviamo due categorie di metodi di crittografia ovvero:

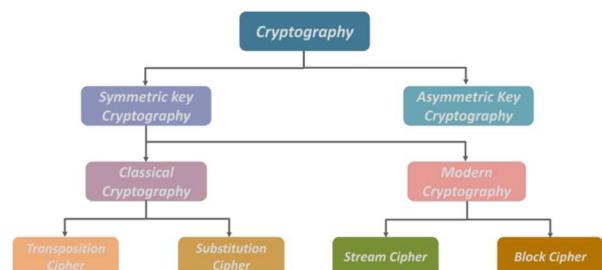
- **Crittografia Classica:**

- o **Cifratura per Trasposizione:** Il testo cifrato costituisce una permutazione del testo in chiaro;
- o **Cifratura per Sostituzione:** Le unità di testo in chiaro vengono sostituite con il testo cifrato, secondo un sistema fisso;

- **Crittografia Moderna:**

- o **Cifratura a Flusso:** Lo stesso bit o byte di testo in chiaro verrà cifrato in un bit o byte diverso ogni volta che viene cifrato;
- o **Cifratura a Blocchi:** Un algoritmo deterministico, insieme a una chiave simmetrica, viene applicato per cifrare un blocco di testo, piuttosto che un singolo bit alla volta;

**Crittografia a Chiave Asimmetrica:** Due chiavi vengono utilizzate in ogni operazione e sono diverse ma matematicamente correlate, in modo che il recupero del testo in chiaro decifrando il testo cifrato sia possibile

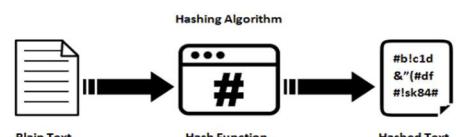
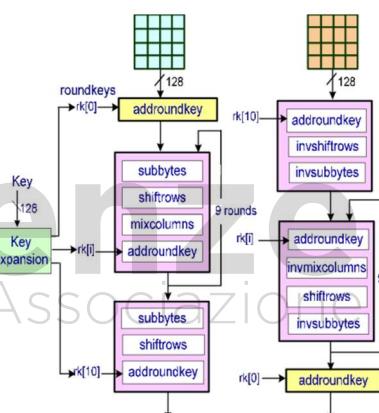


### L'IMPLEMENTAZIONE DELL'AES:

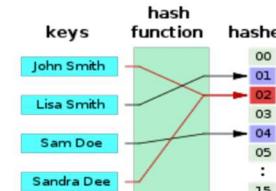
**Advanced Encryption Standard (AES)** è una specifica per la cifratura dei dati elettronici ed è uno dei cifrari a blocchi. AES si basa su un principio di progettazione noto come rete di sostituzione-permutazione, con una dimensione del blocco fissa di 128 bit e una dimensione della chiave di 128, 192 o 256 bit.

**Rivest Cipher 4 (RC4)** è il più utilizzato di tutti i cifrari a flusso, particolarmente in software in vari protocolli come WEP e WPA.

- **Addroundkey:** Ogni byte viene combinato con un byte della sottochiave del round utilizzando l'operazione XOR.
- **Subbytes:** Ogni byte nello stato viene sostituito con la sua voce in una tabella di lookup fissa a 8 bit.
- **Shiftrows:** I byte in ogni riga dello stato vengono spostati ciclicamente a sinistra.
- **Mixcolumns:** Ogni colonna viene moltiplicata con un polinomio fisso  $c(x)$ ;

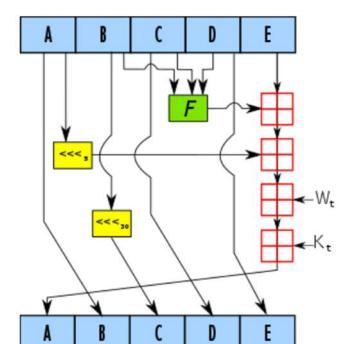


L'hashing è il processo di conversione di un dato di dimensione arbitraria in un altro valore di dimensione fissa utilizzando un algoritmo appropriato, che dovrebbe essere unidirezionale, in modo che l'hash non possa essere riconvertito nel dato originale. Gli **Algoritmi di Hash Sicuri** (SHA) sono una famiglia di funzioni crittografiche di hash. SHA-1 è l'algoritmo più utilizzato e costituisce la base di molte applicazioni e protocolli, tra cui TLS e SSL, SSH, IPsec, o per la firma digitale di documenti.



SHA-1 produce un message digest di 160 bit a partire dall'input (di massimo  $2^{160}$  - 1 bit), composto da blocchi da 512 bit. Consiste in una serie di operazioni in un round, per un totale di 80 round.

- **A, B, C, D ed E** sono parole a 32 bit dello stato, con un valore iniziale;
- **F** è una funzione non lineare;
- **$\lll n$**  indica una rotazione a sinistra dei bit di **n** posizioni, dove **n** varia;
- **W<sub>t</sub>** è la parola del messaggio espanso del round t;
- **K<sub>t</sub>** è la costante del round t.
- Il quadrato rosso indica l'addizione modulo  $2^{32}$



## **CRITTOGRAFIA NELL'IOT - LIMITI:**

Mentre AES e SHA funzionano bene all'interno dei sistemi informatici, faticano nel mondo dell'Internet of Things (IoT) poiché richiedono:

- troppa potenza di elaborazione;
- troppo spazio fisico;
- troppa energia consumata dalla batteria;

Sia le organizzazioni nazionali che internazionali delineano una serie di metodi che possono essere utilizzati per la crittografia leggera e che potrebbero essere utili nei dispositivi IoT e RFID. Definiscono lo spettro dei dispositivi come segue:

- **Crittografia convenzionale:** Server e Desktop; Tablet e smartphone;
- **Crittografia leggera:** Sistemi embedded; Reti RFID e di sensori;

## **9.2. INTRODUZIONE ALLA CRITTOGRAFIA LEGGERA**

### **CRITTOGRAFIA LEGGERA:**

La crittografia leggera nasce per adattarsi a dispositivi con risorse molto limitate (come sensori IoT o smart card), che non possono permettersi i blocchi, le chiavi e le S-box grandi degli algoritmi tradizionali. Tipicamente si utilizzano blocchi da 64–80 bit, chiavi sotto i 90 bit e S-box da 4 bit, in modo da ridurre area (misurata in Gate Equivalents), consumo di potenza, RAM e ROM richieste. Queste primitive (standardizzate ad es. da ISO/IEC 29121 e integrate in librerie come wolfSSL o sharkSSL) offrono throughput e latenza inferiori rispetto ad AES o SHA-256, ma garantiscono sufficienti margini di sicurezza nelle reti IoT. Il vero obiettivo è trovare il miglior compromesso tra costi (area e potenza), prestazioni (larghezza di banda, latenza) e resistenza agli attacchi ai canali laterali: un chip economico ed efficiente rischia di essere vulnerabile, uno economico e protetto può essere lento e uno veloce e sicuro risulta necessariamente più costoso. Gli algoritmi di crittografia leggera puntano proprio a bilanciare al meglio queste esigenze.

### **IMPLEMENTAZIONE HARDWARE:**

Quando una primitiva leggera viene realizzata in hardware, il progetto deve ruotare attorno a quattro parametri chiave, sempre in conflitto fra loro:

- **Area (Gate Equivalents – GE):** misura quanta superficie di silicio occupa il circuito, inclusi memoria e logica. Un valore basso di GE è preferibile per ridurre costi e dimensioni del chip;
- **Throughput:** quantità di dati cifrati al secondo. Un throughput elevato è desiderabile per applicazioni a elevata velocità di trasmissione;
- **Latenza:** tempo che intercorre tra l'applicazione dell'input e la produzione dell'output. Una bassa latenza è cruciale quando serve risposta istantanea, ad esempio in sistemi real-time;
- **Consumo di potenza:** energia necessaria durante il funzionamento, fondamentale in dispositivi alimentati a batteria;

Dalla pratica emerge che abbassare uno di questi valori tende a far crescere almeno un altro. Ad esempio, un'implementazione "unrolled" – che esegue più round crittografici in parallelo in un singolo ciclo di clock – riduce drasticamente la latenza, ma moltiplica l'area e spesso anche il consumo. Al contrario, un approccio seriale lavora solo su pochi bit (tipicamente la dimensione di uno S-box) per ciclo, risultando in un circuito molto compatto e a basso consumo, ma richiede più cicli di clock per completare l'intera permutazione, aumentando la latenza percepita anche se, grazie alla semplicità della logica, è possibile spingere molto in alto la frequenza di clock.

In pratica si può scegliere tra:

- **Implementazioni round-based:** memorizzano l'intero stato e lo trasformano simultaneamente, valutando tutti i bit di output in parallelo;
- **Unrolled:** estendono il round-based facendo girare più round insieme, ottimo per latenza minima ma molto "pesante" in termini di area;
- **Serializzate:** aggiornano a ogni ciclo solo una porzione dello stato (es. 4 bit di S-box), contenendo l'area a scapito di un certo ritardo complessivo

Infine, poiché la memoria è spesso l'elemento più ingombrante, nei cifrari leggeri si preferiscono blocchi e chiavi piccoli e talvolta si "bruciano" le chiavi nel silicio (hard-coding) usando strutture di sola lettura, eliminando la necessità di RAM per lo storage delle chiavi stesse. Progetti di successo, come PRINCE, Mantis e Qarma, mostrano come scelte mirate su questi parametri possano portare a un compromesso ottimale tra velocità, dimensione e consumo energetico.

### **IMPLEMENTAZIONE SOFTWARE:**

Quando si implementa un cifrario leggero in software (soprattutto su microcontrollori), l'efficienza si valuta su tre fronti:

- **Consumo di RAM:** quantità di dati temporanei caricati in memoria durante ogni cifratura. Meno operazioni di caricamento/stoccaggio si fanno, meno RAM si usa e meno si penalizza il throughput;
- **Dimensione del codice:** spazio occupato dal programma crittografico in ROM/flash. Va mantenuto basso per non esaurire lo storage limitato dei dispositivi embedded;
- **Throughput (larghezza di banda):** byte cifrati per ciclo di clock; va massimizzato per ottenere buone prestazioni.

Questi tre parametri sono in conflitto: per esempio, pre-calcolare le sottochiavi accelera la cifratura (migliora il throughput) ma richiede più RAM o codice. Allo stesso modo, ridurre i caricamenti da RAM nei registri della CPU abbassa il consumo di memoria e aumenta il throughput, ma può far crescere la complessità del codice.

Un'ulteriore complicazione viene dal set di istruzioni del microcontrollore: su CPU a 32 bit tutte le rotazioni hanno costo uniforme e basso, mentre su architetture a 8 bit rotazioni diverse da multipli di otto richiedono sequenze di istruzioni costose. Anche qui, un buon progetto terrà conto di quali operazioni hardware siano più economiche.

Infine, tecniche come il bit-slicing possono sfruttare al massimo il parallelismo delle parole macchina: rappresentando ogni bit di più dati in registri diversi, si applicano operazioni bit-wise in parallelo come se fosse SIMD, ottenendo un netto aumento del throughput a fronte di un sovraccarico iniziale di conversione dei dati. In sintesi, scegliere la combinazione giusta di pre-calcolo, ottimizzazione dei caricamenti e uso di tecniche avanzate come il bit-slicing è fondamentale per ottenere la migliore efficienza software su dispositivi a risorse ristrette.

## BIT-SLICING:

La tecnica del bit-slicing trasforma il modo in cui i dati vengono rappresentati in memoria per sfruttare al massimo le istruzioni native della CPU. Invece di memorizzare un valore w-bit in una singola parola macchina, si utilizza un array di w parole: nella parola i-esima si colloca il bit i-esimo di tutti i valori da processare, allineato sempre alla stessa posizione.

Così, operazioni bit-wise come XOR e AND, che normalmente lavorano su tutti i bit di una parola in parallelo, applicano simultaneamente l'operazione a w diversi dati "in parallelo" (simile al SIMD). Questo elimina l'overhead di dover spostare e mascherare bit dentro la parola per combinarli fra loro.

Certo, serve un passaggio iniziale per convertire i dati dalla forma standard a quella bit-slice (e uno finale per tornare indietro), ma se l'algoritmo richiede molte operazioni logiche component-wise il guadagno complessivo in throughput è spesso significativo.

## IMPLEMENTAZIONI LEGGERE:

Nelle "implementazioni leggere" si cerca di rendere più snella l'adozione di algoritmi come AES sia in hardware sia in software, ma emergono limiti intrinseci:

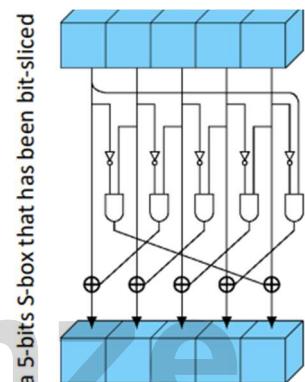
- **Acceleratori hardware AES** possono cifrare/decifrare molto velocemente grazie a istruzioni dedicate, ma restano vulnerabili agli attacchi su canali laterali e richiedono tabelle di lookup per lo S-Box da 8 bit, che occupano spazio non trascurabile;
- **AES in software** è rapido, ma l'S-Box e l'ampiezza del blocco impongono un codice e una RAM minimi che non sempre si possono ridurre per dispositivi estremamente vincolati;

Per queste ragioni, quando serve un footprint ancora più piccolo (ad esempio per funzioni di hash su sistemi embedded) si progettano cifrari leggeri che soddisfano vincoli precisi di area, consumo e memoria. Tutti mantengono comunque un meccanismo di non-linearità (tipicamente uno S-Box, oppure operazioni ARX) per garantire confusione e resistenza agli attacchi crittanalitici, ma in forma molto più compatta rispetto ai corrispettivi "full-size".

## TENDENZE NELLA CRITTOGRAFIA LEGGERA:

Nel panorama della crittografia leggera, i progettisti affrontano vincoli di memoria, area hardware, consumo energetico e velocità, spingendo verso soluzioni che semplificano ogni componente del cifrario mantenendo però un sufficiente livello di sicurezza.

Il cuore di ogni cifrario è lo strato non lineare, che tradizionalmente viene realizzato tramite S-box. Le S-box basate su lookup-table (LUT) offrono proprietà crittografiche quasi ottimali in un solo colpo, ma richiedono di memorizzare intere tabelle in ROM o RAM, un lusso che i dispositivi ultraleggeri non possono sempre permettersi. In alternativa, le S-box bit-slice vengono definite fin dall'inizio come sequenze di operazioni bit-wise (AND, XOR, OR) che non hanno bisogno di tabella: costano poche porte logiche in hardware, si prestano a mascherature molto più semplici contro gli attacchi ai canali laterali e mantengono una buona efficienza in software, sacrificando però un po' di margine crittografico rispetto alle LUT "piene".



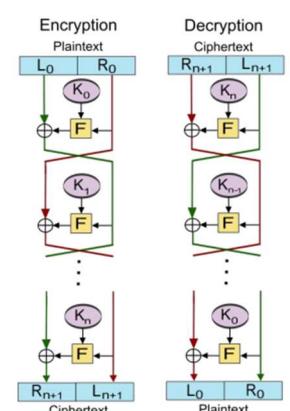
Un'altra linea di design si basa sui cifrari ARX (Addition–Rotation–XOR), che usano l'addizione modulare per introdurre non linearità e rotazioni/XOR per la diffusione. Questo approccio è estremamente leggero in termini di codice e registri su microcontrollori, ma rende più complessa la sicurezza formale, perché non esistono strumenti generali come la "strategia del wide trail" per analizzarne le debolezze differenziali o lineari.

Lo strato di diffusione stesso può essere implementato in modi diversi: matrici MDS garantiscono una diffusione teoricamente ottimale ma sono costose; semplici permutazioni di bit sono quasi gratuite in hardware ma lente in software; e la combinazione "rotazioni a livello di parola + XOR" fornisce un buon compromesso, sfruttando l'efficienza delle rotazioni nelle CPU e la facilità di implementazione in silicio.

Sul fronte della gestione delle chiavi, gli algoritmi leggeri preferiscono schemi minimi. Spesso si selezionano direttamente alcuni bit della chiave principale o si usano costrutti tipo Even–Mansour (pre-whitening, permutazione pubblica, post-whitening) per evitare logiche o stati complessi ad ogni round. In certi casi la chiave viene "hard-codata" in ROM, annullando del tutto la necessità di memoria di scrittura per lo stato delle sottochiavi.

Infine, poiché la resistenza agli attacchi fisici è cruciale, si integrano contromisure come la mascheratura o le implementazioni a soglia basate su secret sharing, tecniche che però richiedono bit casuali esterni e aumentano il carico di calcolo, soprattutto se lo strato non lineare è composto da molte porte AND. Le S-box bit-slice risultano qui particolarmente vantaggiose, perché ogni AND può essere mascherato indipendentemente ed eseguito parallelamente, abbattendo in parte il sovraccarico di protezione.

In sintesi, le tendenze nella crittografia leggera ruotano attorno a tre grandi temi: scegliere lo strato non lineare più adatto al contesto (LUT vs bit-slice vs ARX), bilanciare vari tipi di diffusione (MDS, permutazioni, rotazioni+XOR) e semplificare la key schedule, il tutto integrando contromisure SCA e ottimizzando costantemente i compromessi tra risorse limitate e sicurezza.



## COMPROMESSI NELLA CRITTOGRAFIA LEGGERA:

Nei cifrari leggeri, ogni decisione progettuale è un compromesso tra risorse e sicurezza.

**Key schedule semplificato:** Per ridurre la logica e lo stato hardware, molti algoritmi leggeri scelgono direttamente porzioni della chiave principale come sottochiavi ("bit-selection"), evitando di aggiornare uno stato di chiave complesso a ogni round. In alternativa si usa la stessa funzione di round, o parte di essa, per rigenerare la chiave, così da non aggiungere ulteriore logica.

**Numero di round vs sicurezza:** Meno round significano latenza e consumo inferiori e potenzialmente maggiore throughput, ma riducono

il margine di sicurezza e facilitano analisi differenziali o lineari. Scegliere il numero giusto è un equilibrio tra velocità e resilienza.

**Forza della S-Box vs costo:** Un S-Box con alto grado algebrico e bassa uniformità differenziale offre ottima resistenza, ma richiede più porte logiche o memoria per le lookup-table. Implementazioni più leggere usano S-Box bit-slice con poche porte non lineari, che però possono avere un margine crittografico leggermente inferiore.

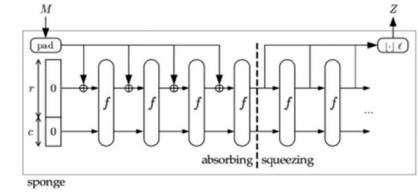
**Specializzazione vs versatilità:** Alcuni cifrari sono pensati per hardware minimizzato (pochi GE), sacrificando facilità di serializzazione o semplicità di contromisure; altri sono generici, basati su operazioni a livello di parola e rotazioni, garantendo buone prestazioni sia in hardware sia in software ma con un'area o un codice un po' più grandi.

In sostanza, ogni algoritmo leggero si posiziona lungo un continuum che va dalla massima velocità e compattezza (con scelte audaci e margini di sicurezza più bassi) fino alla versatilità generale e a margini di sicurezza più conservativi.

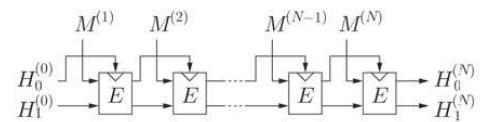
## HASHING:

I metodi di hashing leggeri, come SPONGENT, Lesamnta-LW e PHOTON, utilizzano tecniche diverse per bilanciare sicurezza, prestazioni e efficienza hardware.

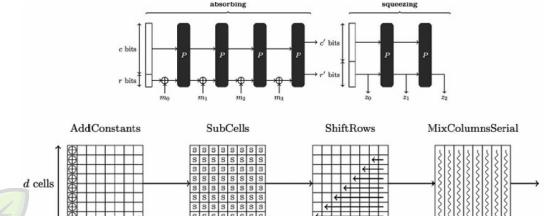
- **SPONGENT** utilizza la costruzione **sponge**, che mappa un input di lunghezza variabile in un output di lunghezza variabile. Funziona combinando i blocchi di input con uno stato interno ( $r + c$  bit) tramite una permutazione  $f$ . L'input viene prima **paddeggia**to, quindi i blocchi vengono **assorbiti** (XOR nei primi  $r$  bit dello stato) e infine **estratti** nella fase di spremitura. Gli ultimi  $c$  bit dello stato sono isolati e non vengono mai restituiti;



- **Lesamnta-LW** è un algoritmo basato su AES, ottimizzato per l'hashing. Usa una **struttura Merkle-Damgård** con una funzione di compressione basata su AES. La gestione delle chiavi e la miscelazione sfruttano una **rete Feistel generalizzata** (GFN) e operazioni non lineari (AES S-Box, moltiplicazione MDS). Questo design è più efficiente rispetto ai tradizionali algoritmi come SHA-256;



- **PHOTON** è un algoritmo leggero basato su AES che usa una funzione sponge con tre fasi principali: **inizializzazione, assorbimento e spremitura**. In fase di assorbimento, i blocchi di input vengono XORati con lo stato interno e alternati con una funzione di permutazione  $t$ -bit. Nella fase di spremitura, i bit vengono estratti e ulteriormente permutati. La **funzione di permutazione** di PHOTON è simile ad AES con 12 round e include operazioni come XOR con costanti, S-box di PRESENT, shift delle righe e moltiplicazione MDS. Le due architetture hardware possibili sono una **seriale** (per ridurre la dimensione) e una **parallela** (per migliorare le prestazioni);



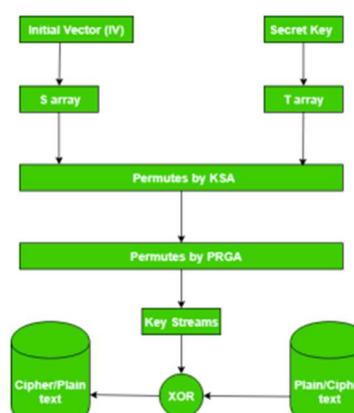
## STREAMING CIPHER:

È un cifrario a flusso che utilizza una **chiave di lunghezza variabile** e cifra un byte alla volta. La **sequenza di chiavi** viene generata tramite un generatore di numeri pseudocasuali che si combina con il testo in chiaro tramite l'operazione XOR. Il processo include due fasi principali:

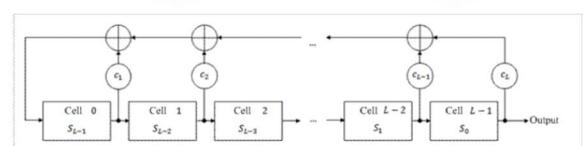
- **Key Scheduling Algorithm (KSA):** crea una permutazione iniziale dell'array di stati.
- **Pseudo-Random Generation Algorithm (PRGA):** genera byte pseudocasuali che vengono XORati con il testo in chiaro.

Un modo semplice per costruire un cifrario a flusso è usare un normale cifrario a blocchi in modalità contatore, generando il blocco successivo di chiavi cifrando valori successivi di un "contatore".

Alcuni meccanismi di implementazione di base possono essere introdotti per ridurre i costi di implementazione di RC4 o di altri cifrari a flusso:

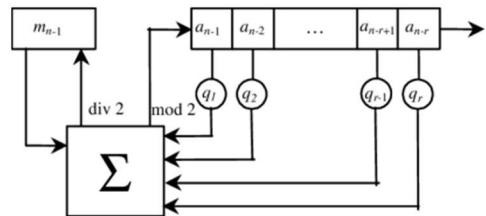


- **Linear Feedback Shift Register (LFSR):** Un registro a scorrimento di parole di bit, in cui il bit di ingresso è una funzione lineare (es. XOR) del suo stato precedente (cioè il contenuto delle celle del registro). Sono disponibili implementazioni sia hardware che software per generare numeri pseudocasuali.

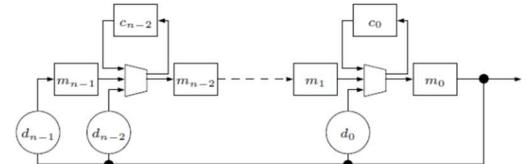


- **Feedback with Carry Shift Register (FCSR)** — Una variazione dell'LFSR dove un registro secondario viene utilizzato per memorizzare il riporto. FCSR ha due implementazioni:

- La versione **Fibonacci** consiste in una singola funzione di feedback che dipende da più ingressi;



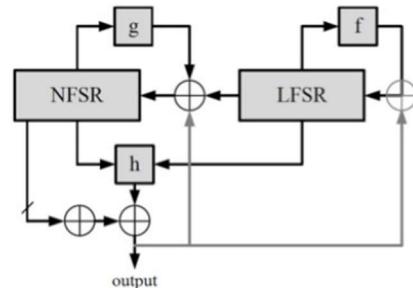
- La versione **Galois** ha più funzioni di feedback con un ingresso comune;



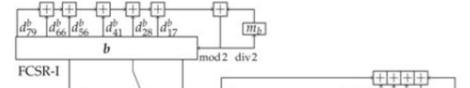
Il **F-FCSR Galois** è utilizzato in quanto più efficace; tuttavia, è possibile avere un **ring FCSR**, visto come un compromesso tra le due rappresentazioni precedenti.

Un **Nonlinear Feedback Shift Register (NLFSR)** è un registro a scorrimento il cui bit di ingresso è una funzione non lineare del suo stato precedente. È più resistente agli attacchi crittografici rispetto agli LFSR.

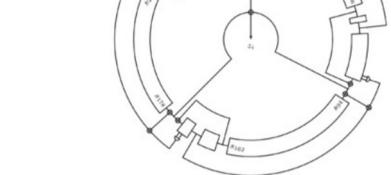
**Grain** è un cifrario a flusso sincrono con due registri a scorrimento: uno con **feedback lineare (LFSR)** e uno con **feedback non lineare (NFSR)**. Il **funzionamento** include l'uso di polinomi di grado 80 per determinare il feedback e la **funzione di uscita** utilizza bit provenienti da entrambi i registri. È progettato per essere efficiente in hardware con un **basso consumo di area**.



**Bean** Simile a Grain, ma utilizza **FCSRs** (Feedback with Carry Shift Registers) anziché LFSR e NFSR. Gli FCSRs combinano **non linearità e lungo periodo**, garantendo un'efficace resistenza agli attacchi crittografici e un'implementazione efficiente in hardware.



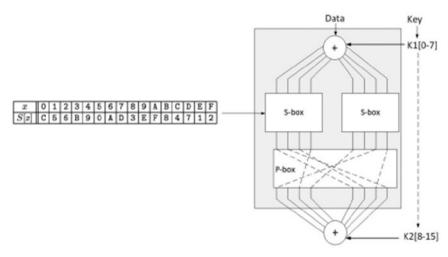
**Trivium** ha uno stato interno di 288 bit e tre registri a scorrimento di lunghezze diverse. Ad ogni ciclo, un **bit** viene spostato nei registri usando una combinazione non lineare. La **fase di inizializzazione** imposta la chiave e l'input aggiuntivo, e lo stato viene aggiornato 1152 volte per garantire una **dipendenza complessa e non lineare** tra lo stato e la chiave.



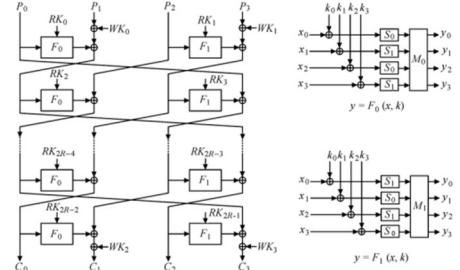
## CIFRARIO A BLOCCHI:

I cifrari a blocchi leggeri sono progettati per garantire sicurezza e ridurre al minimo l'uso delle risorse, il che li rende ideali per dispositivi con capacità limitate. Alcuni esempi di questi cifrari sono **PRESENT**, **CLEFIA**, **SPARX**, e **KATAN**, che si distinguono per approcci diversi ma altrettanto mirati a ridurre l'ingombro fisico e migliorare l'efficienza.

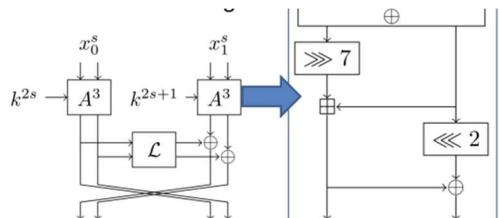
**PRESENT** è un cifrario molto compatto che lavora su blocchi da 64 bit e supporta chiavi di 80 o 128 bit. Usa una struttura chiamata **Rete Sostituzione-Permutazione (SPN)**, che applica un **S-box 4x4 bit** al posto degli S-box più grandi tipici di cifrari come AES. In totale, ci sono 32 round, durante i quali i dati vengono combinati con la chiave tramite **XOR**, garantendo un'efficace diffusione del messaggio. Grazie alla sua struttura semplice, **PRESENT** è circa 2,5 volte più compatto di AES, pur mantenendo un buon livello di sicurezza.



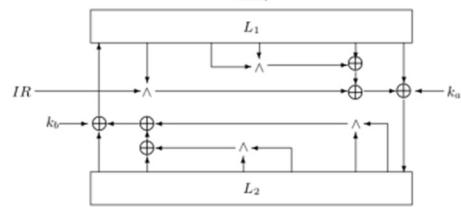
**CLEFIA**, invece, adotta una **rete Feistel generalizzata** con 4 rami e 8 rami, migliorando l'efficienza delle funzioni di miscelazione grazie all'uso di due funzioni **F** parallele per ogni round. Le sue **S-box** a 8 bit e l'uso di matrici **4x4** operanti su **GF(28)** lo rendono sicuro e abbastanza efficiente. Inoltre, l'algoritmo prevede operazioni di **key whitening** all'inizio e alla fine per rafforzare ulteriormente la sicurezza.



**SPARX** è un altro cifrario a blocchi, ma con una struttura **ARX** (Addition, Rotation, XOR), che si basa su operazioni semplici e veloci come l'addizione modulo 216, XOR e rotazioni. È progettato usando una strategia chiamata **Long Trail Strategy**, che si distingue dalla tradizionale **Wide-Trail Strategy**, ed è pensato per essere efficiente anche su hardware a risorse limitate. La non linearità in SPARX è fornita dalla funzione **SPECKEY**, che mescola il testo in chiaro e la chiave in modo efficiente.



**KATAN** è progettato specificamente per **ottimizzare l'ingombro fisico**. Utilizza due registri che operano in modo semplice e efficace: i dati vengono XORati con la chiave e passano attraverso funzioni booleane non lineari. Questo cifrario, che richiede ben **254 round** per garantire una buona mescolanza, è particolarmente adatto per applicazioni che richiedono cifratura leggera ma sicura.



## CRYPTOSISTEMI A CURVE ELLITTICHE:

La **Crittografia a Curve Ellittiche (ECC)** offre un livello di sicurezza simile a quello di altre crittografie basate su problemi difficili come il **Discrete Logarithm Problem (DLP)** o il **Integer Factorization Problem (IFP)**, ma con **chiavi più corte**.

### FUNZIONAMENTO DELLA ECC:

ECC si basa sull'uso di curve ellittiche definite su un campo finito. La sicurezza di ECC deriva dalla difficoltà del problema **elliptic-curve discrete logarithm problem (ECDLP)**, che consiste nel determinare un **intero  $k$**  dato un punto base  **$P$**  e un punto  **$Q$**  sull'ellisse, in modo tale che  **$Q = kP$** .

### OPERAZIONI MATEMATICHE IN ECC:

ECC è un gruppo di punti  $(E, \oplus)$  definito su un campo  $(F, +, \times)$ , e le operazioni di **addizione** e **moltiplicazione scalare** si eseguono sui punti della curva.

- **Operazioni di Aggiunta ( $P \oplus Q$ ):**

- Per sommare due punti **P** e **Q** sulla curva, si usa una **retta di corda** che passa per i punti **P** e **Q**. L'intersezione di questa retta con la curva ci dà il punto **R**, che è la somma dei due punti. Se il punto **P** è uguale a **Q**, si usa la **tangente** alla curva;
- La somma viene poi riflessa sull'asse **x** per ottenere il punto finale **R**;

- **Moltiplicazione per un intero  $k$  ( $kP$ ):**

- La moltiplicazione di un punto **P** per un intero **k** è eseguita tramite **ripetute addizioni** del punto **P**, ovvero  $kP = P \oplus P \oplus \dots \oplus P$  (**k** volte);

### SICUREZZA DELL'ECDLP:

Il problema **ECDLP** diventa molto difficile da risolvere quando si lavora con curve definite su **campi primi grandi**. Ad esempio, con un campo di ordine  **$2^{160}$** , è facile calcolare  **$Q = kP$**  se si conoscono  **$k$**  e  **$P$** , ma determinare  **$k$**  conoscendo  **$Q$**  e  **$P$**  è computazionalmente difficile.

### OPERAZIONE DI ECC CON IL CAMPO F11:

In un esempio pratico con il campo primo **F11**, la curva ellittica è definita dall'equazione:

$$y^2 = x^3 + 4x + 3 \bmod 11 \quad y^2 = x^3 + 4x + 3 \bmod 11$$

In questo campo, 13 soluzioni per l'equazione della curva sono trovate tramite ricerca esaustiva, e i punti risultanti formano un gruppo di **14 elementi** (incluso il punto all'infinito **O**).

- **Addizione di punti:** Le regole per aggiungere due punti sono definite da formule matematiche che coinvolgono operazioni mod 11.

### CRITTOGRAFIA PUBBLICA ECC:

ECC viene utilizzato in vari sistemi di **crittografia a chiave pubblica**, dove la sicurezza si basa sull'ECDLP. Ad esempio:

- Alice crea una **chiave privata** (un valore scalare casuale) e una **chiave pubblica** (un punto sull'ellisse, calcolato come **aP**).
- Bob usa la chiave pubblica di Alice per cifrare un messaggio **M** e inviarlo ad Alice in modo sicuro, calcolando **K = kP** e **C = KA + M**.
- Alice, una volta ricevuti **K** e **C**, può decifrare il messaggio calcolando **S = aK** e poi determinando il messaggio con **M = C - S**

### OPERAZIONI ECC PIÙ COSTOSE:

Le principali operazioni computazionali in ECC includono:

1. **Moltiplicazione del punto fisso  $P$  ( $k \cdot P$ )**.
2. **Moltiplicazione casuale del punto  $Q$  ( $k \cdot Q$ )**.
3. **Moltiplicazione scalare doppia base ( $k \cdot P + l \cdot Q$ )**.

Poiché queste operazioni si basano su una sequenza di **raddoppiamenti e addizioni di punti**, è importante ottimizzare il numero di raddoppiamenti e addizioni per ridurre i costi computazionali.

### VANTAGGI DI ECC:

- **Chiavi più corte:** ECC offre livelli di sicurezza simili a quelli di **RSA** e **DSA** ma con **chiavi molto più corte**, rendendolo più efficiente in

termini di spazio e potenza di calcolo, particolarmente utile per dispositivi con risorse limitate come quelli dell'***Internet delle cose (IoT)***.

- **Sicurezza comprovata:** La sicurezza di ECC è ben studiata e basata su problemi matematici difficili da risolvere, come l'ECDLP, che garantiscono la protezione dei dati



CoScienze  
Associazione

## 10.1. OS-LEVEL PROTECTION MEANS

### CONTIKISEC:

Il sistema operativo Contiki offre ContikiSec, un layer di rete sicuro per le reti di sensori wireless, progettato per bilanciare il basso consumo energetico e la sicurezza, pur mantenendo una piccola impronta di memoria. Di default, Contiki non fornisce servizi di confidenzialità o autenticità nelle comunicazioni tra nodi. ContikiSec offre tre modalità di sicurezza:

- **ContikiSec-Enc**: garantisce confidenzialità e integrità;
- **ContikiSec-Auth**: garantisce autenticità e integrità;
- **ContikiSec-AE**: supporta confidenzialità, autenticità e integrità

### SECURITY IN TINYOS:

TinyOS dispone di una vasta gamma di librerie legate alla sicurezza, inclusi algoritmi di crittografia simmetrica e asimmetrica. Alcuni esempi includono:

- **Crittografia simmetrica**: implementazioni di AES e Trivium, e funzioni hash universali (MMH, Poly32);
- **Crittografia a chiave pubblica**: TinyECC (Crittografia su curve ellittiche) e TinyPBC (Crittografia basata su accoppiamenti);
- **Librerie di comunicazione sicura**: TinySec, SenSec e MiniSec, che offrono protocolli per la crittografia e l'autenticazione dei dati;

### SECURITY IN ZERYNTH OS:

Zerynth OS, invece, offre una serie di moduli di sicurezza, tra cui:

- Implementazione quasi completa dell'interfaccia SSL/TLS;
- Operazioni su certificati X.509 e primitive per la crittografia e le firme dei dati;
- Funzionalità di sicurezza del firmware, come la protezione della memoria flash contro accessi esterni (e.g. JTAG), la riserva di memoria per parti critiche del sistema e la protezione contro i guasti del firmware, tramite meccanismi come il watchdog;

Tuttavia, la maggior parte delle funzionalità di sicurezza del firmware dipende fortemente dalla piattaforma.

## 10.2. TRUSTED EXECUTION ENVIRONMENT

### SOFTWARE-FIRST ISOLATION (SFI):

SFI è una tecnica di strumentazione software a livello di codice macchina per creare domini di protezione logica all'interno di un processo. Assegna una regione di memoria a componenti non affidabili e modifica le istruzioni per limitare l'accesso alla memoria. Questo approccio è noto come *code sandboxing*.

### ISOLATED EXECUTION (ESECUZIONE ISOLATA):

L'isolamento dell'esecuzione del codice è una tecnica fondamentale per la sicurezza. La virtualizzazione, pur creando ambienti di esecuzione isolati per strumenti difensivi, ha alcuni limiti:

- Dipendenza da hypervisor con una base di fiducia (**TCB**) ampia;
- Incapacità di gestire rootkit basati su hypervisor o firmware;
- Sovraccarico di prestazioni del sistema (ad esempio, i cambi di contesto da una VM all'hypervisor).

Il **Trusted Computing Base (TCB)** include tutti i componenti hardware, firmware e software critici per la sicurezza di un sistema, vulnerabilità all'interno di questa base possono compromettere la sicurezza del sistema.

### VIRTUALIZZAZIONE BASATA SU CONTAINER:

Approcci basati su container, come Docker, consentono di isolare le applicazioni senza creare macchine virtuali complete. Tuttavia, i container sono vulnerabili ad attacchi come l'uso di immagini compromesse, il download di payload dannosi durante l'esecuzione, e l'accesso ai file di sistema ospite.

### ESECUZIONE ISOLATA CON HARDWARE:

L'esecuzione isolata supportata dall'hardware utilizza tecnologie di **Trusted Execution Environment (TEE)**, che eseguono strumenti difensivi in ambienti separati dal sistema compromesso. L'uso di tecnologie hardware riduce il carico di prestazioni ed elimina la necessità di hypervisor, offrendo un livello di privilegio più elevato e maggiore sicurezza.

### SECURE EXECUTION ENVIRONMENT:

Un ambiente sicuro garantisce l'esecuzione di applicazioni in isolamento, proteggendo i dati sensibili e prevenendo che le applicazioni maligne accedano o alterino i dati e il codice di altre applicazioni. Comprende anche la protezione dei dati memorizzati e la protezione contro manomissioni del firmware.

### SOLUZIONI BASATE SU HARDWARE:

Le soluzioni di sicurezza hardware, come l'uso di elementi sicuri (SE) e moduli di piattaforma sicura (TPM), offrono protezione contro attacchi che tentano di intercettare o manipolare i dati sensibili. Gli **Elementi Sicuri (SE)** sono chip microprocessori che memorizzano e proteggono dati sensibili, come chiavi di pagamento. Questi dispositivi limitano l'esecuzione delle applicazioni alle sole applicazioni firmate dal produttore, riducendo il rischio di attacchi.

### E3 (ENCRYPTED EXECUTION ENVIRONMENT):

Un **Encrypted Execution Environment (E3)** è un ambiente in cui il software dell'applicazione è criptato per proteggere le istruzioni da svelamenti non autorizzati, impedendo la duplicazione e l'uso non autorizzato del software.

## TRUSTED PLATFORM MODULE (TPM):

Il **TPM** è un microcontrollore con capacità crittografiche avanzate, utilizzato per proteggere i dati. Fornisce funzionalità crittografiche, come la generazione di numeri casuali e l'accelerazione RSA, e supporta tecniche di protezione dei dati come **memory curtaining** e **sealed storage**, che legano i dati allo stato specifico del sistema, garantendo che possano essere utilizzati solo in determinate configurazioni hardware e software.

## TEE:

I Trusted Execution Environments (TEE) combinano hardware e software per creare ambienti separati e sicuri all'interno di un sistema. Questi ambienti sono progettati per proteggere operazioni sensibili come la crittografia, isolando i dati critici da quelli dell'utente e riducendo la superficie di attacco. Un sistema TEE è diviso in due ambienti principali:

- **Rich Execution Environment (REE)**: È il sistema operativo standard, che offre tutte le funzionalità ordinarie di un dispositivo, ma aumenta la superficie di attacco;
- **Trusted Execution Environment (TEE)**: È un sistema operativo sicuro che esegue operazioni critiche e sensibili, come la gestione della crittografia e delle chiavi, separandole dall'ambiente principale. Inoltre, il TEE fornisce comunicazione sicura tra il processore e le periferiche, come display e dispositivi di input, proteggendo i dati da attacchi esterni;

Una delle principali caratteristiche del TEE è la memoria sicura, che isola i dati sensibili da quelli utilizzati dalle applicazioni normali. In più, il TEE supporta un processo di boot sicuro per garantire che il sistema parta in modo protetto, attraverso la lettura di una ROM di fiducia e la verifica dell'integrità del sistema operativo sicuro.

Il TEE separa il processore in due zone distinte: una per le operazioni sicure (TEE) e una per quelle non sicure (REE). Il passaggio tra queste due zone avviene tramite una modalità monitor che consente di salvare e caricare il contesto quando si passa dall'ambiente sicuro a quello normale. In questa modalità, il sistema operativo sicuro utilizza un set di istruzioni limitato per ridurre la superficie di attacco, garantendo la protezione dei dati sensibili.

L'hardware che supporta il TEE include crittoprocessori specializzati nell'esecuzione di algoritmi crittografici, come i Trusted Platform Modules (TPM) e Hardware Security Modules (HSM). Questi dispositivi possono essere integrati o esterni e accelerano le operazioni crittografiche senza mai memorizzare chiavi segrete. I TEE possono anche utilizzare coprocessori di sicurezza che sollevano compiti critici di sicurezza, operando in modo isolato, ma richiedendo un sovraccarico nella gestione dei dati.

Alcuni esempi noti di TEE includono ARM TrustZone e Intel Trusted Execution Technology (TXT), che supportano la creazione di ambienti sicuri attraverso l'isolamento dei core virtuali. ARM TrustZone, ad esempio, usa un'istruzione chiamata Secure Monitor Call (SMC) per passare tra la modalità "normale" e quella "sicura", permettendo che solo un mondo (normale o sicuro) sia attivo alla volta.

## OPEN-TEE:

Open-TEE è una versione virtuale di un ambiente TEE che implementa le specifiche di Global Platform (GP), uno standard internazionale per la gestione dei TEE. Open-TEE permette agli sviluppatori di creare applicazioni sicure senza la necessità di hardware dedicato, riducendo le barriere per l'accesso e l'uso dei TEE.

Open-TEE agisce come un demone nello spazio utente e fornisce un'architettura che permette di eseguire Trusted Applications (TA) in un ambiente sicuro. L'architettura di Open-TEE comprende diversi componenti:

- **Base Process**: Gestisce le funzionalità del TEE e la creazione di Trusted Applications (TA)
- **Manager**: Monitora lo stato delle TA, gestisce la comunicazione e fornisce uno spazio sicuro di archiviazione;
- **Launcher**: Crea nuovi processi TA in modo efficiente, separando la logica di comunicazione interprocesso (IPC) e l'esecuzione della logica della TA.

Open-TEE utilizza un protocollo di comunicazione basato su socket Unix e segnali interprocesso per trasferire i messaggi tra le Client Applications (CA) e le Trusted Applications (TA). Le librerie TEE Client API e TEE Core API sono condivise per ridurre il consumo di memoria e codice.

L'interfaccia di Open-TEE consente di interagire con l'ambiente sicuro anche senza dispositivi hardware dedicati, semplificando lo sviluppo e migliorando l'accesso alle capacità di sicurezza offerte dal TEE. In questo modo, Open-TEE è un ottimo strumento per sviluppatori che desiderano integrare funzionalità di sicurezza nelle loro applicazioni senza dipendere dall'hardware specifico.

## SAMSUNG ARTIK:

La piattaforma **ARTIK** di Samsung è progettata per garantire un elevato livello di sicurezza in tutte le fasi di utilizzo, sfruttando capacità hardware avanzate. Tra le principali caratteristiche di sicurezza, ARTIK integra un **Secure Element** certificato EAL5, che consente la registrazione sicura dei dispositivi e l'autenticazione tramite **ARTIK Cloud**, assicurando che i dispositivi siano autentici e che le comunicazioni siano verificate.

Oltre alla sicurezza di base, ARTIK supporta funzionalità avanzate come **secure boot**, **secure JTAG**, e l'integrazione di **TrustZone** e **Secure Element** per garantire l'archiviazione sicura e l'esecuzione di algoritmi crittografici in ambienti protetti, separati dalle operazioni utente.

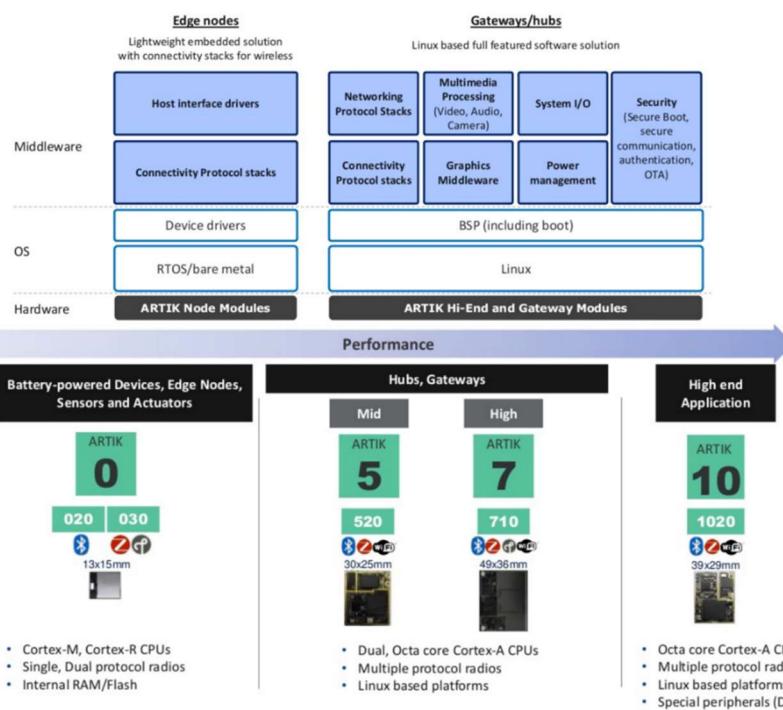
In particolare, ARTIK accelera i protocolli di crittografia **AES** e **RSA** e utilizza **Elliptic Curve Diffie-Hellman (ECDH)** per la condivisione sicura delle chiavi.

Il **secure boot** è fondamentale per la sicurezza della piattaforma, poiché ogni fase di avvio viene verificata e firmata, impedendo l'esecuzione di software non autorizzato. Inoltre, il sistema utilizza un **Key Management System (KMS)** con **Hardware Security Modules (HSM)** per una gestione sicura delle chiavi crittografiche.

Per proteggere fisicamente i dispositivi, ARTIK include un **secure JTAG** che impedisce accessi non autorizzati alla porta JTAG tramite una password legata al numero di serie del modulo. Inoltre, i dispositivi ARTIK delle famiglie 5 e 7 supportano **TEE (Trusted Execution Environment)**, basato su **ARM TrustZone**, che consente di eseguire operazioni sensibili in un ambiente separato e sicuro dal sistema operativo non sicuro. Questo è supportato dal **SEE (Secure Execution Environment)**, che fornisce un sistema operativo sicuro e set di API per la comunicazione sicura tra ambienti.

ARTIK offre anche un **Storage Sicuro**, che permette di proteggere i dati sensibili utilizzando un sistema di archiviazione sicura e un **Secure**

**Element** che gestisce in modo protetto le chiavi crittografiche e i dati memorizzati. Infine, il **Security SubSystem (SSS)** gestisce tutte le funzioni hardware relative alla sicurezza, come l'esecuzione sicura del codice, l'archiviazione sicura, l'accelerazione hardware per la crittografia e l'uso di una **Funzione Uncloneable Fisica (PUF)** per migliorare ulteriormente la sicurezza del dispositivo.



# CoScienze

Associazione

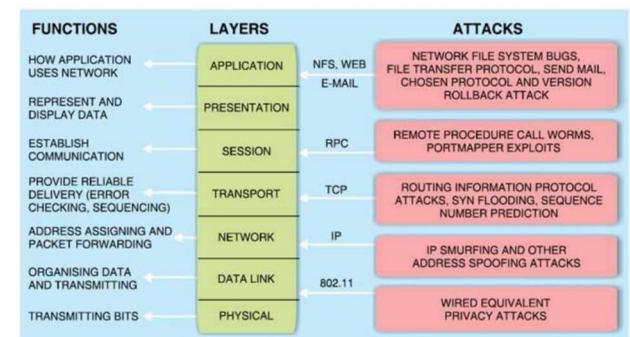
## 11.1. SECURE COMMUNICATION MEANS

### SECURE COMMUNICATION:

In uno scenario di comunicazione sicura, Alice (la mittente) e Bob (il destinatario) scambiano dati attraverso un canale insicuro, con l'obiettivo di proteggere la privacy e l'integrità della loro comunicazione contro potenziali attaccanti, come Trudy, che potrebbero cercare di intercettare, modificare, inserire o eliminare i messaggi. Alice vuole garantire che solo Bob possa leggere i suoi messaggi, mentre Bob vuole verificare che il messaggio che riceve provenga effettivamente da Alice e che non sia stato alterato durante il transito. Inoltre, entrambi vogliono impedire che qualcuno neghi loro l'accesso alle risorse necessarie o intercetti i loro messaggi.

Le principali proprietà che definiscono una comunicazione sicura sono:

- **Confidenzialità:** Solo il mittente e il destinatario devono essere in grado di comprendere il contenuto del messaggio;
- **Integrità del messaggio:** Il contenuto dei messaggi non deve essere alterato, sia in modo malizioso che accidentale;
- **Autenticazione dei terminali:** Entrambe le parti devono essere in grado di verificare l'identità dell'altra, per essere sicuri di comunicare con la persona giusta;
- **Sicurezza operativa:** Poiché le reti sono connesse a Internet, ci sono rischi di attacchi ai sistemi sottostanti, come worm, attacchi DoS e spionaggio aziendale, che devono essere mitigati.



La sicurezza della rete può essere discussa in relazione al modello OSI, in cui gli attacchi possono verificarsi a diversi livelli:

- **Livello 1 (Fisico):** Questo livello riguarda gli aspetti fisici, come cavi o segnali wireless. Gli attacchi in questa area includono il taglio dei cavi, l'interferenza nei segnali wireless o l'esecuzione di attacchi **DoS** per interrompere il mezzo fisico di comunicazione. Le reti wireless sono vulnerabili a **packet sniffing** e **radio jamming**;
- **Livello 2 (Data Link):** A questo livello, gli attacchi riguardano la manomissione dei dispositivi come switch, **MAC flooding** o **ARP poisoning**, dove un attaccante associa il proprio indirizzo MAC con l'indirizzo IP di un altro dispositivo per reindirizzare il traffico. **Compromettere la cache ARP** o sovraccaricare la tabella degli indirizzi MAC di uno switch può interrompere la comunicazione o deviare i dati sensibili;
- **Livello 3 (Rete):** Questo livello si occupa del routing e degli attacchi **DoS** come le **flooding di Ping**. Gli attacchi come il **packet sniffing** sono anche comuni, dove un attaccante intercetta dati sensibili. Gli attacchi DoS mirano ai router con un traffico eccessivo, come le **flooding Ping/ICMP echo**, con l'obiettivo di sovraccaricare la rete.
- **Livello 4 (Trasporto):** Questo livello riguarda i protocolli come **TCP** e **UDP**. Gli attacchi comuni includono la **port scanning** (per scoprire le porte aperte), il **SYN flooding** o il **TCP session hijacking**. Nel **SYN flooding**, un attaccante invia numerosi richieste di connessione false, causando l'esaurimento delle risorse del server. Il **TCP session hijacking** consente a un attaccante di iniettare dati in una connessione TCP esistente, prendendo il controllo della sessione.
- **Livello 5 (Sessione):** L'**attacco di session hijacking** avviene quando un attaccante prende il controllo di una sessione attiva, sfruttando le vulnerabilità nei protocolli di comunicazione a questo livello.
- **Livello 7 (Applicazione):** Questo livello, dove si trovano i protocolli come **HTTP**, è il livello dove avvengono la maggior parte degli attacchi. Gli attacchi comuni includono gli **attacchi DoS** contro i server HTTP o attacchi su protocolli a livello di applicazione come i protocolli di trasferimento file o e-mail.

### AUTENTICAZIONE DEI TERMINALI:

L'autenticazione dei terminali è il processo in cui un'entità dimostra la propria identità a un'altra entità attraverso una rete di computer. Questo processo deve avvenire esclusivamente tramite i messaggi scambiati come parte di un **Protocollo di Autenticazione (AP)**, che viene eseguito prima che le due parti comunicanti utilizzino qualsiasi altro protocollo:

- **AP 1.0 - Autenticazione semplice:** Nel primo esempio (AP 1.0), Alice invia semplicemente un messaggio a Bob dicendo "sono Alice". Tuttavia, non c'è alcun modo per Bob di sapere con certezza che la persona che invia il messaggio "sono Alice" sia effettivamente Alice, quindi questo metodo non è sicuro;
- **AP 2.0 - Autenticazione tramite indirizzo di rete:** Nel secondo esempio (AP 2.0), se Alice ha un indirizzo di rete noto, come un indirizzo IP, da cui comunica sempre, Bob potrebbe tentare di autenticare Alice verificando che l'indirizzo IP del messaggio di autenticazione corrisponda a quello conosciuto. Tuttavia, questa soluzione è vulnerabile allo **spoofing IP**, poiché è facile per un attaccante creare un pacchetto IP con l'indirizzo di origine desiderato;
- **AP 3.0 - Autenticazione con password segreta:** Un approccio più sicuro (AP 3.0) prevede l'uso di una password segreta condivisa tra l'autenticatore (Bob) e la persona da autenticare (Alice). Tuttavia, la vulnerabilità in questo caso è che Trudy, l'attaccante, potrebbe intercettare la comunicazione e ottenere la password di Alice, dato che la password viene inviata in chiaro e non cifrata. Una soluzione sarebbe quella di cifrare la password e far sì che Alice e Bob condividano una chiave segreta simmetrica (AP 3.1).
- **AP 3.1 - Autenticazione con password cifrata:** Nel caso dell'AP 3.1, anche l'uso della crittografia non risolve completamente il problema di autenticazione. Bob potrebbe essere vittima di un **attacco di riproduzione**: Trudy potrebbe semplicemente intercettare la password cifrata e riprodurla successivamente a Bob per far credere di essere Alice.
- **AP 4.0 - Autenticazione con nonce:** Il problema degli attacchi di riproduzione in AP 3.1 viene risolto nell'AP 4.0, introducendo l'uso di un **nonce**. Un nonce è un numero utilizzato solo una volta nella vita di un protocollo, garantendo che non venga riutilizzato. Questo valore viene inviato da Bob ad Alice come parte della comunicazione di autenticazione. Passaggi dell'AP 4.0:
  - Alice invia un messaggio "Sono Alice" a Bob;
  - Bob sceglie un nonce (R) e lo invia ad Alice;
  - Alice cifra il nonce utilizzando la chiave segreta simmetrica condivisa tra Alice e Bob (KA-B) e invia il nonce cifrato (KA-B(R)) a Bob;
  - Bob decifra il messaggio ricevuto. Se il nonce decrittato corrisponde al nonce che ha inviato, Alice viene autenticata.

Con l'uso del nonce, Bob è sicuro che Alice sia:

- sia chi dice di essere (poiché conosce la chiave segreta necessaria per cifrare il nonce);
- sia "viva" (poiché ha cifrato un valore appena generato da Bob).

Questo sistema elimina il rischio di attacchi di riproduzione, poiché il nonce è unico per ogni sessione di autenticazione.

## ATTACCHI DI JAMMING:

Un attacco di **jamming** consiste nella trasmissione di segnali radio che interferiscono con le comunicazioni, riducendo il rapporto

**Signal-to-Interference-plus-Noise Ratio (SINR)**, ovvero il rapporto tra la potenza del segnale e la somma della potenza del rumore e dell'interferenza.

Gli attacchi di jamming danneggiano le comunicazioni wireless bloccando il canale di trasmissione, facendo in modo che il trasmettitore si fermi quando rileva un canale occupato o corrompendo i segnali ricevuti dai destinatari. I tipi di Jammers sono:

- **Jammer costanti**: Questi dispositivi emettono segnali elettromagnetici in modo continuo, causando interferenze nei segnali legittimi e facendo apparire il canale occupato. Il difetto principale è il rapido esaurimento dell'energia a causa della trasmissione continua;
- **Jammer ingannevoli**: Simili ai jammer costanti, ma invece di inviare sequenze casuali, emettono una sequenza di bit legittima, dando l'impressione che un nodo legittimo sia presente;
- **Jammer casuali**: Conservano energia alternando tra stati di jamming casuale e stati di "sleep" (sonno);
- **Jammer reattivi**: Non emettono segnali in continuazione, ma ascoltano il canale di trasmissione e reagiscono emettendo segnali quando rilevano una trasmissione di dati.

Il **jamming** può avvenire in diversi modi, ad esempio:

- **Corruzione dei pacchetti di richiesta di trasmissione (RTS)**: L'attaccante distorce il pacchetto RTS, impedendo al destinatario di rispondere con un pacchetto di autorizzazione (CTS), causando così il fallimento della trasmissione dei dati;
- **Corruzione dei pacchetti di dati o di conferma (ACK)**: I pacchetti corrotti portano a ritrasmissioni dei dati da parte del mittente, aumentando la latenza e riducendo l'efficienza della rete;
- **Frequenza di salto rapido**: L'attaccante cambia canale continuamente, "saltando" da un canale all'altro molto velocemente, causando interferenze brevi ma frequenti;

Per **contrastare gli attacchi di jamming**, sono possibili alcune strategie difensive:

- **Bassa potenza di trasmissione**: Ridurre la potenza di trasmissione per rendere difficile per i dispositivi di jamming rilevare le trasmissioni legittime;
- **Spreading del segnale con FHSS (Frequency Hopping Spread Spectrum)**: Cambiare rapidamente tra frequenze diverse per evitare che l'attaccante possa interferire su tutte le frequenze utilizzate. FHSS ha tre vantaggi principali:
  - Resistenza agli interferenti a banda stretta;
  - Difficoltà nell'intercettazione se il pattern di salto delle frequenze è sconosciuto;
  - Interferenza minima con trasmissioni convenzionali.
- **Antenne direzionali**: Utilizzare antenne direzionali che trasmettono e ricevono segnali solo in una direzione specifica. Un jammer deve trovarsi nella stessa direzione dell'antenna direzionale per interferire efficacemente;
- **Channel Surfing**: Simile a FHSS, ma senza che i nodi finali debbano concordare su un pattern segreto di salto di frequenza. Quando un nodo rileva il jamming, cambia il canale wireless. Nella versione coordinata, tutti i nodi cambiano canale simultaneamente;
- **Canali multipli**: I jammer possono cambiare canali e interferire con diverse larghezze di banda in momenti diversi, ottimizzando l'uso dell'energia e l'impatto dell'attacco. Inoltre, i jammer possono saltare tra i canali in sequenze pseudocasuali predeterminate

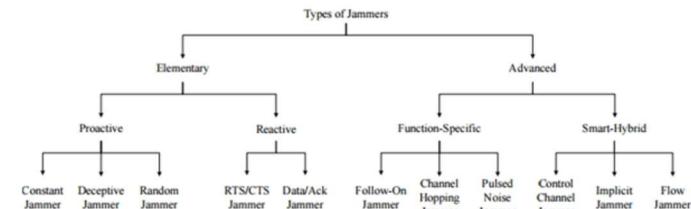
I jammers puntano a:

- **Interferire con il canale di controllo**: Il canale utilizzato per coordinare l'attività di rete, causando disfunzioni nei nodi della rete;
- **Disabilitare i nodi di destinazione**: Attacchi di jamming che causano un'interruzione nel servizio o una denial-of-service (DoS);
- **Ridurre il flusso di traffico**: Bloccare o ridurre il traffico di rete per rallentare o interrompere la comunicazione;

Il **rilevamento di un attacco di jamming** è difficile, ma un nodo può raccogliere misurazioni prima dell'attacco e costruire un modello statistico che descrive le misurazioni di base nella rete, per confrontarle con quelle successive all'attacco.

## PROTOCOLLI SICURI SU INTERNET:

- **Sicurezza a livello applicazione**:
  - **Vantaggi**: La sicurezza a livello di applicazione è progettata specificamente per i requisiti dell'applicazione in questione, offrendo soluzioni molto mirate e personalizzate per le necessità specifiche;
  - **Svantaggi**: Richiede l'implementazione di molteplici meccanismi di sicurezza, il che può aumentare la complessità e la difficoltà di gestione;
- **Sicurezza a livello di trasporto**:
  - **Vantaggi**: Fornisce un'interfaccia comune ai servizi di sicurezza, il che rende più facile l'implementazione di soluzioni di sicurezza coerenti per diverse applicazioni;
  - **Svantaggi**: Richiede modifiche, anche se minori, alle applicazioni esistenti per integrare i servizi di sicurezza;
- **Sicurezza a livello di rete**:
  - **Vantaggi**: Funziona con applicazioni che non sono progettate per la sicurezza (soprattutto quelle "ignoranti" riguardo alla sicurezza), quindi è possibile implementare la sicurezza senza la necessità di modificare le applicazioni;
  - **Svantaggi**: Potrebbe richiedere modifiche a livello di sistema operativo, il che può comportare modifiche più complesse e costose



rispetto ad altre soluzioni;

## 11.2. DATA LINK-LEVEL SECURITY

### WEP (WIRED EQUIVALENT PRIVACY):

WEP è un meccanismo di sicurezza inizialmente standardizzato nel protocollo 802.11 per fornire una sicurezza simile a quella delle reti cablate. Si basa su un approccio di chiave simmetrica condivisa per l'autenticazione e la cifratura dei dati tra un host e un punto di accesso wireless. Tuttavia, WEP non specifica un algoritmo per la gestione delle chiavi, il che può risultare problematico.

- **Autenticazione in WEP:** Il processo di autenticazione avviene in più fasi:

- o Un host wireless invia una richiesta di autenticazione a un punto di accesso;
- o Il punto di accesso risponde con un valore nonce di 128 byte;
- o L'host wireless cifra il nonce usando la chiave simmetrica condivisa e lo invia al punto di accesso;
- o Il punto di accesso decifra il nonce e, se corrisponde a quello inviato, l'host è autenticato;

- **Problemi di sicurezza in WEP:**

- o La vulnerabilità principale risiede nell'uso del cifrario **RC4** che, se utilizzato con chiavi deboli, è facilmente attaccabile;
- o Gli attaccanti possono alterare il contenuto cifrato e ricalcolare il **CRC** per creare un frame 802.11 che il ricevitore accetti senza rilevare l'integrità;
- o La gestione della chiave può diventare complessa, soprattutto in ambienti aziendali, poiché se una chiave viene compromessa, tutti i dispositivi della rete devono modificarla;
- o Le schede di rete (NIC) possono solo autenticare i punti di accesso, ma non viceversa, rendendo possibile per gli attaccanti indirizzare il traffico verso punti di accesso non autorizzati.

### WPA (WI-FI PROTECTED ACCESS):

Il Wi-Fi Protected Access (WPA) è stato sviluppato per migliorare WEP, con l'adozione di una chiave più lunga (256 bit invece di 128 bit in WEP) e un **IV (Initialization Vector)** più lungo. WPA è un protocollo intermedio che prepara il passaggio al più sicuro **WPA2**.

- **Modalità di WPA:**

- o **WPA personale** (WPA-PSK): Semplice da configurare, basato su una passphrase condivisa tra i dispositivi. Tuttavia, se un dispositivo viene compromesso, tutti i dispositivi devono cambiare la password;
- o **WPA aziendale**: Richiede una configurazione più complessa, con l'autenticazione degli utenti tramite un server RADIUS. Se un dispositivo viene compromesso, l'amministratore può revocare l'accesso in modo indipendente rispetto agli altri dispositivi;

- **Elementi chiave di WPA:**

- o **Protocollo di integrità della chiave temporale (TKIP):** Sostituisce il CRC con il **MICHAEL**, un miglior protocollo per il controllo dell'integrità del messaggio (MIC), progettato per evitare i problemi di WEP come il tentativo di indovinare la chiave e il bit-flipping. Il TKIP agisce sull'intero frame e previene gli attacchi di frammentazione;
- o **RADIUS:** Un protocollo per l'autenticazione centralizzata degli utenti, che consente di gestire l'accesso alla rete in modo più sicuro;
- **Problemi con WPA:** Sebbene WPA migliori WEP, l'algoritmo di cifratura sottostante è ancora basato su WEP, che utilizza un cifrario lineare vulnerabile al **bit flipping**. Inoltre, se due frame con **MIC** errati vengono ricevuti in un intervallo di 60 secondi, l'access point disconnette tutti i client e richiede una nuova negoziazione delle chiavi. Sebbene questa sia una misura di sicurezza, introduce una vulnerabilità di **denial-of-service**

### IEEE 802.11i (WPA2):

**IEEE 802.11i**, conosciuto anche come **WPA2**, è un miglioramento significativo rispetto ai meccanismi di sicurezza precedenti (come WEP e WPA) applicati alla famiglia di standard 802.11. È progettato per garantire una sicurezza molto più forte nelle comunicazioni wireless.

#### Vantaggi rispetto a WEP e WPA:

- **WEP** forniva una cifratura relativamente debole, un solo metodo di autenticazione e nessun algoritmo di gestione delle chiavi;
- **WPA** migliorava la sicurezza di WEP utilizzando una chiave più robusta per l'integrità del messaggio (MIC) e introduceva l'uso di RADIUS per un'autenticazione migliore;
- **IEEE 802.11i** utilizza l'Advanced Encryption Standard (AES) invece di TKIP (utilizzato in WPA), e introduce una cifratura dinamica delle chiavi, che cambia regolarmente la chiave, rendendo più difficile l'attacco. Inoltre, fornisce un set estensibile di meccanismi di autenticazione e un protocollo di distribuzione delle chiavi;

**CCM** (Counter Mode with CBC-MAC) è il protocollo di cifratura utilizzato in IEEE 802.11i. Basato su AES, fornisce sia l'autenticazione che la riservatezza dei dati. IEEE 802.11i definisce un **server di autenticazione** con cui il punto di accesso (AP) può comunicare. Separare il server di autenticazione dall'AP consente di servire più AP, centralizzando le decisioni riguardo all'autenticazione e l'accesso. Ciò riduce i costi e la complessità degli AP.

### FASI DI AUTENTICAZIONE:

- **Fase di scoperta:** Il punto di accesso pubblicizza la sua presenza e le modalità di autenticazione e cifratura supportate. Il client richiede i metodi di autenticazione e cifratura desiderati;
- **Autenticazione reciproca:** L'autenticazione avviene tra il client wireless e il server di autenticazione, con l'AP che agisce come un relay. Il server di autenticazione invia un "Master Key" (MK) al client e all'AP per consentire la generazione di chiavi successive, come la **Pairwise Master Key (PMK)**;
- **Generazione della chiave temporale (TK):** Con la PMK condivisa, il client e l'AP possono generare chiavi aggiuntive, come la chiave temporale (TK), utilizzata per la cifratura dei dati trasmessi sul link wireless

EAP definisce i formati di messaggio end-to-end tra client e server di autenticazione. I messaggi EAP sono incapsulati in EAPoL (EAP over

LAN) e inviati tramite il link wireless 802.11. Vengono quindi incapsulati nel protocollo RADIUS per la trasmissione su UDP/IP. Anche se 802.11i non impone un metodo di autenticazione specifico, l'autenticazione EAP-TLS è spesso utilizzata.

### BLUETOOTH SECURITY:

La sicurezza in Bluetooth è fondamentale e viene implementata solo sui link radio; quindi, l'autenticazione e la cifratura del link sono possibili, ma la sicurezza completa a livello end-to-end richiede soluzioni aggiuntive a livello di applicazione.

Bluetooth utilizza un sistema di **frequency hopping** (salto di frequenza) con 1.600 salti al secondo, combinato con il controllo della potenza del segnale radio. Esistono tre modalità di sicurezza in Bluetooth:

- **Modalità 1:** Non sicura. Non vengono impiegati meccanismi di protezione per impedire la connessione di dispositivi non autorizzati;
- **Modalità 2:** Una modalità in cui un gestore di sicurezza centralizzato controlla l'accesso ai servizi e dispositivi specifici;
- **Modalità 3:** La sicurezza è attivata prima che il canale venga stabilito, una funzione di sicurezza integrata che non dipende dalla sicurezza dell'applicazione

### CHIAVI DI LINK E CIFRATURA:

Durante la fase di inizializzazione, due dispositivi Bluetooth "associati" generano una chiave di link utilizzando un PIN identico in entrambi i dispositivi. La cifratura protegge i payload dei pacchetti scambiati tra i dispositivi utilizzando un cifrario a flusso con registri di scorrimento a retroazione lineare (LFSR).

### AUTENTICAZIONE BLUETOOTH:

Bluetooth utilizza un protocollo di "**challenge-response**" per autenticare i dispositivi. Il richiedente trasmette il proprio indirizzo e il verificatore invia una sfida casuale. Il richiedente restituisce una risposta calcolata usando una chiave segreta. Se la risposta è corretta, la connessione può proseguire.

### LIVELLI DI FIDUCIA E DI SERVIZIO:

Bluetooth supporta due livelli di fiducia:

- **Dispositivi fidati:** Con una relazione permanente, hanno accesso completo ai servizi;
- **Dispositivi non fidati:** Non hanno una relazione permanente, con accesso limitato;

Bluetooth definisce anche tre livelli di sicurezza dei servizi:

- **Livello 1:** Richiede autorizzazione e autenticazione;
- **Livello 2:** Richiede solo autenticazione;
- **Livello 3:** È aperto a tutti i dispositivi, senza necessità di autenticazione

### SICUREZZA DI ZIGBEE:

La sicurezza di **ZigBee** è uno degli aspetti fondamentali della sua architettura e segue le linee guida definite in **IEEE 802.15.4**. Fornisce meccanismi per il controllo dell'accesso ai dispositivi di rete (autenticazione), cifratura (crittografia a chiave simmetrica) e integrità, utilizzando **Message Integrity Checks (MIC)**.

L'architettura di sicurezza di ZigBee si basa sull'uso della crittografia a **chiave simmetrica** e include un protocollo di gestione delle chiavi molto elaborato.

ZigBee definisce tre tipi di dispositivi logici, ognuno con un ruolo specifico:

- **ZigBee Coordinator:** Un dispositivo responsabile dell'istituzione, esecuzione e gestione dell'intera rete ZigBee. Gestisce il livello di sicurezza della rete e la configurazione dell'indirizzo del **Trust Center**;
- **ZigBee Trust Center:** Un'applicazione che gira su un dispositivo di fiducia, responsabile per la distribuzione delle chiavi agli altri dispositivi nella rete;
- **Router:** Gestisce la comunicazione tra i dispositivi nella rete ZigBee;
- **End Device:** Comunica solo con un nodo "genitore", che è solitamente un router o il coordinator.

ZigBee definisce i livelli di rete e applicazione sopra il livello **MAC** (Medium Access Control):

- **ZigBee Network Layer:** Garantisce l'integrità e la cifratura dei frame trasmessi applicando la cifratura **AES** (Advanced Encryption Standard) e l'integrità utilizzando il Cipher Block Chaining **Message Authentication Code** (CBC-MAC);
- **ZigBee Application Layer:** La sicurezza dei frame in questo livello può essere basata su **link keys** o **network key**:
  - **Network Key:** Viene utilizzata per proteggere la comunicazione **broadcast** (da un dispositivo a più dispositivi);
  - **Link Key:** Viene utilizzata per proteggere la comunicazione **unicast** (tra due dispositivi). Le **link keys** possono essere acquisite tramite **key-transport**, **key-establishment** o **preinstallation**;

### 11.3. NETWORK-LEVEL SECURITY

#### IPSEC (IP SECURITY):

**IPSec** è un protocollo di sicurezza progettato per proteggere i datagrammi IP a livello di rete, fornendo autenticazione e cifratura tra entità di rete come host e router. IPSec è fondamentale per garantire la sicurezza delle comunicazioni sulla rete, proteggendo da minacce come l'**IP Spoofing** e il **packet sniffing**.

Vulnerabilità di IP senza IPSec:

- **Integrità e Autenticazione:** L'IP spoofing consente a un attaccante di impersonare un altro sistema, creando pacchetti IP con un indirizzo IP di origine falso;
- **Confidenzialità:** Senza IPSec, i pacchetti IP sono vulnerabili al sniffing, dove un attaccante può intercettare i dati in transito.

## BENEFICI DI IPSEC:

- **Sicurezza per il traffico di rete:** Quando implementato in un firewall o in un router, IPsec fornisce sicurezza per tutto il traffico che attraversa il perimetro della rete, senza impatti sul traffico interno;
- **Trasparente per le applicazioni:** IPsec opera al di sotto del livello TCP/UDP, il che significa che non è necessario modificare il software esistente sui sistemi utente o server;
- **Sicurezza per utenti remoti:** IPsec fornisce una protezione adeguata anche per lavoratori remoti o reti virtuali aziendali sicure.

## I PRINCIPALI PROTOCOLLI DI IPSEC:

IPSec include due protocolli principali:

- **Authentication Header (AH):**
  - Fornisce autenticazione della fonte e integrità dei dati;
  - Non fornisce confidenzialità;
- **Encapsulation Security Payload (ESP):**
  - Fornisce autenticazione della fonte (opzionale), integrità dei dati e confidenzialità;
  - È molto più usato rispetto ad AH, poiché la confidenzialità è spesso critica per molte applicazioni;

## CIFRATURA E ALGORITMI DI AUTENTICAZIONE:

- **Algoritmi di Cifratura:** L'algoritmo di cifratura utilizzato per ESP è definito nel documento che descrive i vari algoritmi supportati;
- **Algoritmi di Autenticazione:** IPsec utilizza algoritmi specifici per garantire l'integrità dei dati e l'autenticazione delle comunicazioni tramite AH e ESP;
- **Gestione delle Chiavi:** IPsec definisce come le chiavi vengono scambiate tra il mittente e il destinatario attraverso protocolli come **ISAKMP** (Internet Security Association and Key Management Protocol).

## ASSOCIAZIONI DI SICUREZZA (SA):

IPSec stabilisce una **Security Association (SA)**, che è una connessione logica unidirezionale tra due entità di rete. Se entrambe le entità desiderano inviare datagrammi sicuri tra di loro, devono essere stabilite due SA, una per ogni direzione.

Le entità IPsec memorizzano le informazioni sulle SA nella **Security Association Database (SAD)**, una struttura di dati nel kernel del sistema operativo. L'**SPI (Security Parameter Index)** è un tag utilizzato per identificare univocamente i flussi di traffico che richiedono diverse regole di cifratura e autenticazione.

## ISAKMP E IKE:

- **ISAKMP:** È un protocollo per stabilire SA e chiavi crittografiche in un ambiente Internet, per l'autenticazione e lo scambio di chiavi. È progettato per essere indipendente dal metodo di scambio delle chiavi;
- **IKE (Internet Key Exchange):** Fornisce il materiale di chiave autenticato per l'uso con ISAKMP. IKE ha due fasi:
  - **Fase 1:** Utilizza **Diffie-Hellman** per creare un SA IKE bidirezionale;
  - **Fase 2:** Le due entità si autenticano reciprocamente, negoziano gli algoritmi di cifratura e autenticazione e scambiano il materiale per creare chiavi di sessione per le SA di IPsec;

## MODALITÀ DI IPSEC:

- **Transport Mode:** In questa modalità, solo il payload del pacchetto IP viene cifrato o autenticato. L'intestazione IP rimane invariata, il che consente una trasmissione diretta e senza alterazioni nel routing. Adatta per la comunicazione end-to-end.
- **Tunnel Mode:** In questa modalità, l'intero pacchetto IP, inclusa l'intestazione, viene cifrato e autenticato. Il pacchetto originale viene quindi incapsulato in un nuovo pacchetto IP con una nuova intestazione. È più sicuro, ma comporta un overhead maggiore e viene comunemente utilizzato in **VPN** (Virtual Private Networks);

## PROBLEMI E LIMITAZIONI DI IPSEC:

- **Dimensione del pacchetto:** I pacchetti IPsec potrebbero crescere oltre la **Maximum Transmission Unit (MTU)** di un gateway, portando a problemi di frammentazione o al rifiuto di trasmissione di pacchetti più grandi;
- **NAT (Network Address Translation):** IPsec può avere problemi nel traversare i firewall NAT, poiché i protocolli IPsec non utilizzano porte, complicando la gestione delle connessioni;
- **Costi e complessità:** IPsec richiede software client di terze parti e una configurazione più complessa rispetto ad altre soluzioni, ma offre una sicurezza maggiore.
- **Prestazioni:** IPsec può soffrire di prestazioni inferiori a causa della mancanza di concorrenza, la necessità di operazioni crittografiche inline e l'accesso casuale alla memoria durante l'elaborazione di ogni pacchetto. Ciò comporta una latenza maggiore rispetto ad altri protocolli, come TLS.

## 11.4. TRANSPORT-LEVEL SECURITY

### SSL (SECURE SOCKET LAYER):

**SSL** è un protocollo di sicurezza utilizzato per garantire la confidenzialità, l'integrità dei dati, l'autenticazione del server e, se necessario, l'autenticazione del client. SSL è ampiamente utilizzato per proteggere le comunicazioni Web, come la connessione sicura a un sito di commercio elettronico.

Problemi risolti da SSL:

- **Confidenzialità:** Protegge i dati inviati tra Bob (l'utente) e Alice Incorporated, impedendo che un intruso intercetti informazioni sensibili come il numero di carta di credito di Bob;
- **Integrità dei dati:** Impedisce che un intruso modifichi l'ordine di Bob (ad esempio, facendogli acquistare dieci volte il numero desiderato di bottiglie di profumo);
- **Autenticazione del server:** Impedisce che un attaccante impersoni il sito web di Alice Incorporated, riducendo i rischi di furto di identità o frodi;

Come funziona SSL:

SSL migliora il protocollo TCP aggiungendo servizi di sicurezza:

- **Fase di Handshake:** Stabilisce una connessione sicura e autenticata tra il client e il server;
- **Protocollo di Record SSL:** Incapsula i messaggi in blocchi cifrati e autenticati, garantendo la sicurezza durante la trasmissione.

Fasi del protocollo SSL:

- **Handshake:**
  - Il client stabilisce una connessione TCP con il server;
  - Il server si autentica inviando un certificato digitale;
  - Il client genera una chiave segreta (pre-master secret), la cifra con la chiave pubblica del server e la invia al server;
  - Entrambi, client e server, calcolano una chiave segreta (master secret) condivisa per la sessione SSL;
- **Scambio di dati:** Il client e il server utilizzano la chiave segreta per cifrare i dati e garantire l'integrità tramite un **Message Authentication Code (MAC)**
- **Conclusione della sessione:** La sessione SSL può essere terminata tramite un messaggio di fine, proteggendo contro attacchi di troncamento della sessione.

## SICUREZZA CON SSL:

**Man-in-the-middle:** L'uso dei numeri di sequenza e l'inclusione nei calcoli MAC impedisce gli attacchi di tipo "man-in-the-middle", dove un attaccante potrebbe intercettare e manipolare i messaggi.

## TLS (TRANSPORT LAYER SECURITY):

**TLS** è il successore di SSL ed è un protocollo crittografico che fornisce autenticazione e cifratura dei dati sulla rete. Sebbene SSL e TLS siano simili, TLS è una versione più sicura e moderna di SSL.

Differenze principali tra SSL e TLS:

- **Supporto per le suite di cifratura:** TLS non supporta più le suite di cifratura **Forteza/DMS** supportate da SSL.
- **Autenticazione:** TLS utilizza **HMAC** (Hash-based Message Authentication Code) per garantire l'integrità dei dati, mentre SSL utilizza un approccio di MAC dopo la compressione dei dati.
- **Algoritmi di Hash:** TLS migliora la sicurezza rispetto a SSL utilizzando un metodo di generazione delle chiavi più robusto e un **pseudorandom function** (PRF) basato su hash come MD5 o SHA.

TLS vs SSL:

- **TLS** è più sicuro di **SSL** e viene utilizzato oggi in sostituzione di SSL per la maggior parte delle applicazioni di sicurezza.
- **SSL** è obsoleto, mentre **TLS** è ampiamente implementato nei protocolli come **HTTPS, FTPS, e SSH**.

## DTLS (DATAGRAM TRANSPORT LAYER SECURITY):

**DTLS** è un protocollo di comunicazione progettato per fornire sicurezza a livello di datagrammi, in modo che le applicazioni basate su datagrammi possano comunicare in modo sicuro, proteggendo contro il monitoraggio, la manomissione o la falsificazione dei messaggi.

Caratteristiche principali:

- **Record Layer:** La struttura di base del DTLS è il Record Layer, che si occupa di cifrare i payload, dividerli in frammenti e incapsularli in pacchetti strutturati chiamati "record". Ogni pacchetto ha un numero di sequenza e un'epoca che si incrementano rispettivamente con ogni messaggio e handshake, utilizzati per calcolare l'hash;
- **Protocollo di Handshake:** Durante l'handshake, client e server negoziano le impostazioni di sicurezza, eseguono l'autenticazione reciproca e stabiliscono un canale sicuro per l'invio dei record cifrati. Il processo inizia con un messaggio di **ClientHello** da parte del client, che contiene le suite di cifratura supportate, i parametri della chiave pubblica e le chiavi condivise per lo scambio della chiave;
- **Gestione dei pacchetti e dei cookie:** Il server può inviare un **HelloRetryRequest** contenente un cookie, una prova che il server è in grado di ricevere pacchetti dall'indirizzo IP richiesto. Ciò rende difficile eseguire attacchi DoS (Denial of Service) con indirizzi IP falsificati;
- **Cifratura dei messaggi:** Il server e il client calcolano insieme una **chiave segreta** per la sessione, che viene utilizzata per generare le chiavi simmetriche per la cifratura dei dati;
- **Retransmissione e timer:** DTLS utilizza un timer e un meccanismo di **ritrasmissione** per garantire che i messaggi vengano inviati fino a quando non vengono ricevuti correttamente. Le transizioni vengono determinate dall'arrivo dei frammenti di dati o dalla scadenza del timer;
- **Frammentazione:** Poiché **UDP** non ha un meccanismo di frammentazione dei pacchetti, DTLS fornisce un proprio meccanismo per frammentare i messaggi che potrebbero superare la dimensione massima del pacchetto di rete (Path-MTU). Ogni pacchetto ha un **Fragment Offset** e un **Fragment Length** per supportare questa operazione;
- **Confronto con TLS:** DTLS è progettato per affrontare problemi come la perdita di pacchetti e l'ordinamento errato dei pacchetti che sono comuni nei protocolli basati su UDP. A differenza di **TLS**, che è più robusto grazie alla sua capacità di garantire la consegna affidabile dei dati, DTLS è meno affidabile ma ha una latenza inferiore. Inoltre, DTLS non permette l'uso di cifrari a flusso, come fa TLS;
- **Uso in applicazioni:** DTLS si integra facilmente nelle applicazioni poiché adotta il modello di programmazione di TLS, che prevede una

relazione uno-a-uno tra i contesti di sicurezza e i canali di comunicazione. A differenza di **IPSec**, che ha un modello di gestione delle chiavi complesso, DTLS è più semplice da implementare.

## VPN (VIRTUAL PRIVATE NETWORK):

Un **VPN** è una soluzione che garantisce la sicurezza della connessione a una rete privata tramite Internet, una rete pubblica non sicura e non affidabile. I VPN consentono di trasferire i dati tra siti remoti e l'intranet di un'organizzazione in modo sicuro.

Tipi di tunneling:

- **IP-in-IP Tunneling:** In questo metodo, l'intero pacchetto, incluso l'intestazione, viene nascosto mentre attraversa Internet tra due siti. Questo garantisce che anche l'intestazione IP del datagramma sia protetta;
- **IP-in-TCP Tunneling:** In questo caso, due parti stabiliscono una connessione TCP e usano questa connessione per inviare datagrammi cifrati. La **TCP** assicura una consegna affidabile e ordinata dei pacchetti

**Vantaggi** di IP-in-TCP: Garantisce la **consegna affidabile** e l'**ordinamento dei pacchetti**, poiché TCP assicura che i datagrammi arrivino correttamente e in ordine.

**Svantaggi** di IP-in-TCP: **Head-of-line blocking:** Poiché TCP consegna i pacchetti in ordine, se un segmento TCP viene perso o ritardato, l'intero flusso di dati si blocca fino a quando il segmento mancante non viene ritrasmesso. Questo rallenta l'intero processo di trasmissione.

## 11.5. APPLICATION-LEVEL SECURE COMMUNICATION

### SICUREZZA NEI SERVIZI PUBLISH/SUBSCRIBE:

I servizi **publish/subscribe** offrono un modo per distribuire notifiche a diversi destinatari senza richiedere una comunicazione diretta tra il publisher e il subscriber. Tuttavia, questi servizi devono affrontare vari rischi di sicurezza, che possono essere mitigati con le tecniche appropriate. Le caratteristiche principali di sicurezza per questi servizi sono:

- **Confidenzialità:** Le notifiche pubblicate non devono essere accessibili agli utenti non autorizzati, anche se corrispondono ai loro interessi;
- **Integrità:** È essenziale che il contenuto delle notifiche non venga alterato in modo non autorizzato;
- **Disponibilità:** Le notifiche non devono essere negate agli utenti autorizzati quando corrispondono ai loro interessi

Altri attributi importanti includono:

- **Autenticità:** Verifica dell'identità dell'utente e dell'integrità del messaggio;
- **Autorizzazione:** Concessione dei diritti di accesso a notifiche o funzionalità specifiche;
- **Rendicontazione:** Tracciamento delle attività degli utenti per analisi forensi;
- **Non Ripudio:** Evidenza di chi ha eseguito determinate operazioni all'interno del servizio.

### MINACCIE NEI SERVIZI PUBLISH/SUBSCRIBE:

Le caratteristiche stesse dei servizi **publish/subscribe** li rendono vulnerabili a una serie di attacchi:

- **Masquerading (Mascheramento):** Un attaccante può fingere di essere un altro utente, sfruttando vulnerabilità nel processo di autenticazione;
- **Flooding:** Un attaccante può generare una quantità enorme di notifiche, sovraccaricando il sistema;
- **Trashing:** Un attaccante può cambiare frequentemente lo stato delle pubblicazioni o delle iscrizioni, creando caos nel sistema;
- **Eavesdropping (Snooping):** Un attaccante può leggere le notifiche senza il consenso dei destinatari, violando la confidenzialità;
- **Replay:** Un attaccante può ripubblicare notifiche con un certo ritardo, distorcendo il flusso delle informazioni.

### SOLUZIONI PER LA SICUREZZA NEI SERVIZI PUBLISH/SUBSCRIBE:

Per contrastare questi attacchi, è necessario implementare le seguenti soluzioni:

- **Autenticazione e Firma:** Per prevenire mascheramenti e garantire che solo i publisher autorizzati possano inviare notifiche;
- **Cifratura:** Per proteggere la confidenzialità delle notifiche e impedire che i dati vengano letti da utenti non autorizzati;
- **Controllo degli Accessi:** Per gestire quali utenti possono iscriversi e ricevere notifiche, garantendo che solo i destinatari autorizzati possano riceverle;
- **Gestione dei Key e Protocollo di Routing Sicuro:** Per proteggere la trasmissione e il matching delle notifiche in modo sicuro, senza rivelare informazioni sensibili durante il routing.

### PROBLEMI LEGATI ALLA CIFRATURA:

- **Confidenzialità delle Notifiche:** Le notifiche devono essere cifrate in modo che solo i subscriber autorizzati possano decrittarle;
- **Confidenzialità delle Iscrizioni:** Anche le iscrizioni agli argomenti devono essere cifrate per prevenire la raccolta di informazioni sensibili sugli interessi degli utenti.

### MODELLO DI CIFRATURA E PROTOCOLLI DI SICUREZZA:

La cifratura end-to-end è spesso preferita in questi servizi, poiché:

- **Non richiede fiducia nei broker intermediari,** che altrimenti potrebbero avere accesso ai dati sensibili;
- È più efficiente rispetto alla cifratura **link-by-link**, che implica un overhead maggiore.

### SICUREZZA NELLE COMUNICAZIONI IOT:

Nel contesto dell'IoT, molte soluzioni standard implementano il modello **publish/subscribe** utilizzando tecniche di cifratura punto-a-

punto come **TLS** o **DTLS** per garantire la sicurezza nelle comunicazioni tra nodi e broker.

### HTTPS: SICUREZZA NELLE COMUNICAZIONI WEB:

**HTTPS** è una versione sicura del protocollo HTTP che utilizza **TLS** (Transport Layer Security) per garantire la sicurezza delle comunicazioni su Internet, prevenendo attacchi come **man-in-the-middle** e proteggendo la privacy e l'integrità dei dati scambiati tra client e server.

Vantaggi:

- **Autenticazione del sito web:** Garantisce che il sito web a cui ci si sta connettendo sia quello legittimo;
- **Protezione della privacy:** Cifra l'URL, i parametri della query, le intestazioni e i cookie, impedendo che i dati sensibili vengano intercettati;
- **Protezione contro gli attacchi:** HTTPS impedisce agli attaccanti di manomettere o intercettare i dati trasmessi.

Problemi:

- Se una parte del sito è caricata tramite HTTP non sicuro, si rischia che i dati sensibili siano esposti;
- I **cookie HTTP** contenenti informazioni sensibili (come credenziali o sessioni) potrebbero essere rubati tramite tecniche di **hijacking** se non protetti correttamente.

### RACCOMANDAZIONI PER UNA SICUREZZA OTTIMALE:

- **Configurazione Completa di HTTPS:** Tutto il sito dovrebbe essere completamente ospitato su HTTPS per evitare vulnerabilità;
- **Abilitare HSTS (HTTP Strict Transport Security):** Questo meccanismo assicura che i browser comunichino solo tramite HTTPS, proteggendo contro possibili attacchi di downgrade;
- **Gestione dei Cookie:** I cookie devono avere l'attributo **Secure** per essere trasmessi solo su connessioni HTTPS e ridurre il rischio di intercettazione.



## 12.1. AUTENTICAZIONE

L'autenticazione degli utenti è fondamentale per garantire l'accesso solo a identità conosciute, mentre l'autorizzazione determina i diritti di accesso a risorse e funzionalità in base a fattori come il ruolo dell'utente o politiche di sicurezza. La responsabilità richiede la tracciabilità delle attività degli utenti per consentire analisi forensi e mappare le minacce alla sicurezza.

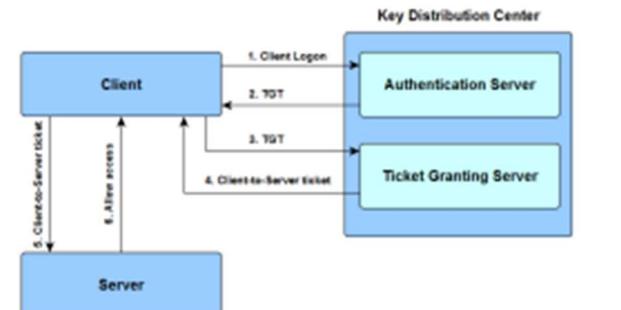
### IDM (GESTIONE IDENTITÀ):

Un'identità rappresenta un'entità in un contesto specifico ed è costituita da un identificatore unico e un set di attributi. I sistemi di gestione identità (IdM) sono responsabili del ciclo di vita delle identità, della verifica della loro veridicità e dell'interazione con altri sistemi.

I modelli di identità basati su password sono i più semplici: ogni utente ha un identificatore e una password univoci. Dopo l'autenticazione, l'utente riceve un token di accesso per interagire con i servizi.

**Kerberos** è un esempio di protocollo di autenticazione, utilizzato da

Microsoft per l'autenticazione all'interno di reti locali. Il processo include un server di autenticazione (AS) che verifica l'utente e rilascia un Ticket-Granting Ticket (TGT), e un Ticket-Granting Server (TGS) che rilascia i ticket di servizio per accedere a risorse. Questo sistema è sicuro, ma i metodi basati su password possono essere vulnerabili, a seconda della qualità della password.



### MODELLO BASATO SU CERTIFICATI:

Un'alternativa è l'autenticazione basata su certificati digitali, che utilizzano la crittografia a chiave pubblica. Un certificato contiene l'identificatore dell'utente, la sua chiave pubblica e attributi, firmati da un'autorità di certificazione (CA). X.509 è uno standard che definisce il formato dei certificati a chiave pubblica, inclusi i certificati di revoca e gli algoritmi di validazione dei percorsi di certificazione.

### AUTENTICAZIONE A UNO, DUE E TRE VIE:

Nel modello di autenticazione a **una via**, un utente (A) invia un messaggio autenticato a un altro utente (B) utilizzando la sua chiave privata per garantire l'integrità e l'originalità del messaggio.

Nel caso dell'autenticazione a **due vie**, entrambi gli utenti (A e B) si verificano reciprocamente utilizzando le loro chiavi private e pubbliche. Infine, l'autenticazione a **tre vie** include un ulteriore messaggio da A a B con una copia firmata di un nonce, per evitare attacchi di replay.



### SAML (SECURITY ASSERTION MARKUP LANGUAGE):

SAML è uno standard aperto per lo scambio di dati di autenticazione e autorizzazione tra un identity provider e un service provider, utilizzando un linguaggio basato su XML.

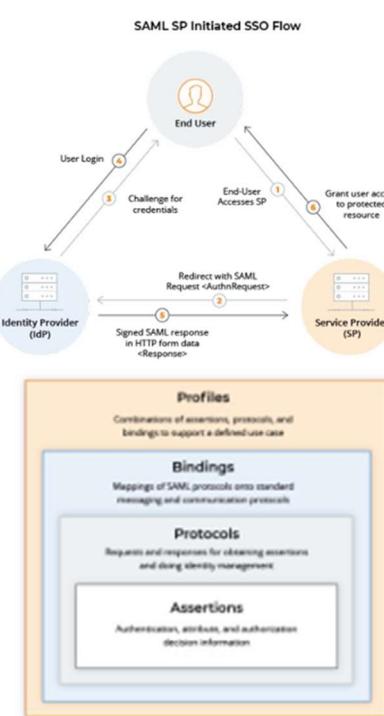
### AUTENTICAZIONE CON SAML:

Il processo di autenticazione inizia quando l'utente finale avvia il login presso il fornitore del servizio (SP). L'SP reindirizza l'utente al Provider di Identità (IdP) con una richiesta SAML (AuthnRequest), che contiene le informazioni necessarie per l'autenticazione dell'utente. L'IdP autentica l'utente e risponde con un'asserzione SAML (SAMLResponse). L'IdP può richiedere all'utente di completare la fase di login. Alla fine, l'SP consente o nega l'accesso all'utente.

Il profilo SAML descrive come le asserzioni, i protocolli e i binding SAML si combinano per supportare casi d'uso definiti. Un binding SAML è una mappatura di un messaggio del protocollo SAML su formati di messaggistica standard o protocolli di comunicazione. I protocolli SAML descrivono come vengono confezionati gli elementi SAML. Le asserzioni SAML contengono pacchetti di informazioni di sicurezza o di decisione.

Nonostante offrano alti livelli di sicurezza, le soluzioni basate su certificati possono essere costose e complesse. L'IdM può essere strutturato in tre architetture:

- **Isolata**, in cui il fornitore del servizio funge anche da IdM;
- **Centralizzata**, dove un singolo IdM gestisce le identità per più fornitori di servizi;
- **Federata**, dove più IdM sono usati in un dominio di fiducia federato per riconoscere identità provenienti da domini diversi, abilitando il Single Sign-On (SSO).



L'architettura isolata è semplice ma poco scalabile. L'architettura centralizzata consente di gestire più servizi con una sola identità, ma l'IdM centralizzato rappresenta un punto di guasto e un collo di bottiglia nelle prestazioni. La soluzione federata è più scalabile e efficiente, ma più complessa da implementare.

## MODELLO BASATO SU CERTIFICATI:

Il controllo accessi determina le risorse IoT a cui un nodo è autorizzato ad accedere per implementare l'autorizzazione. I metodi di controllo accessi includono:

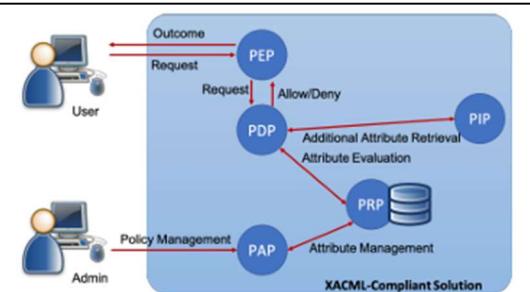
- **Access Control List (ACL)**: un elenco di permessi associati a una risorsa. È semplice ma presenta limiti, soprattutto quando è necessario gestire molti utenti e permessi;
- **Role-Based Access Control (RBAC)**: l'accesso alle risorse si basa sui ruoli degli individui nell'organizzazione. È più scalabile rispetto alle ACL, in quanto non è necessario specificare i permessi individuali, ma solo quelli legati al ruolo. Tuttavia, ha limitazioni, come il controllo accessi grossolano e l'incapacità di gestire accessi tra domini;
- **Attribute-Based Access Control (ABAC)**: consente un controllo più fine, decidendo i permessi in base agli attributi del richiedente, del contesto e/o della risorsa. Il vantaggio di ABAC è che non è necessario conoscere in anticipo il richiedente; l'accesso è permesso se gli attributi corrispondono. Tuttavia, può risultare complesso in ambienti grandi dove esistono attributi e meccanismi di controllo accessi disparati;
- **Policy-Based Access Control (PBAC)**: utilizza politiche digitali con regole logiche per mantenere e valutare dinamicamente l'accesso degli utenti. PBAC adotta l'approccio "zero-trust", che significa "non fidarsi di nessuno" e definisce rigidi controlli di accesso all'interno dell'organizzazione

Il modello PBAC è più uniforme rispetto all'ABAC, offrendo un controllo accessi armonizzato su tutta l'organizzazione.

## 12.2. XACML (EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE)

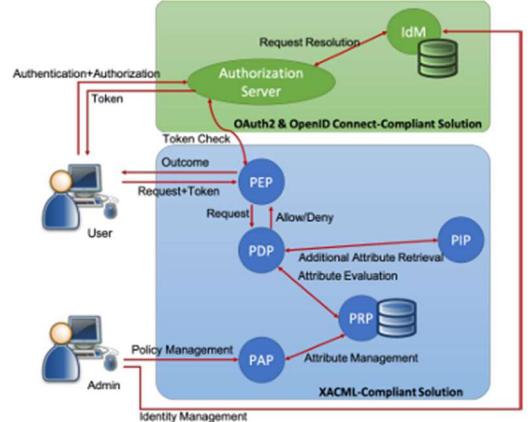
XACML è uno standard OASIS ampiamente utilizzato per progettare e implementare un servizio di controllo accessi. Fornisce un linguaggio per esprimere politiche di controllo accessi e un'architettura per valutarle tramite una serie di servizi web. L'architettura che valuta le politiche di controllo accessi è composta da vari componenti logici che interagiscono tra loro.

- **Policy Enforcement Point (PEP)**: è il punto di front-end che protegge le risorse dalle richieste esterne, estraendo i dettagli necessari per la verifica del controllo accessi;
- **Policy Decision Point (PDP)**: riceve i dettagli estratti dal PEP e fornisce la decisione su come gestire l'accesso;
- **Policy Information Point (PIP)**: funge da fonte per i valori degli attributi, come risorsa, soggetto e ambiente, necessari per prendere una decisione di accesso;
- **Policy Retrieval Point (PRP)**: è responsabile per il recupero e la gestione delle politiche XACML;
- **Policy Access Point (PAP)**: gestisce l'amministrazione delle politiche, consentendo agli amministratori di inserire, modificare o eliminare le politiche XACML



XACML viene usato in combinazione con **OAuth2**, un protocollo che specifica i flussi di messaggi per l'autenticazione e l'autorizzazione. Il flusso di OAuth2 funziona come segue:

- Il **server di autorizzazione** interagisce con l'utente, agendo come il proprietario della risorsa, e se la decisione è positiva, l'utente riceve un **token di autorizzazione**;
- OAuth2 realizza un accesso delegato e una pseudo-autenticazione, che è complementare a **OpenID**. Mentre OpenID fornisce un'asserzione di identità dell'utente, con OAuth2 l'utente ottiene una prova di permesso per accedere alla risorsa richiesta.

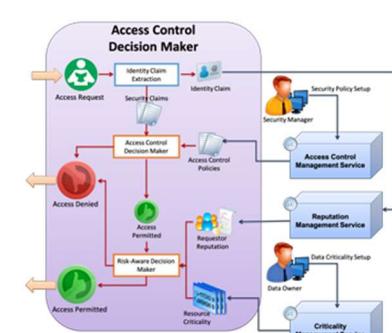


**OpenID Connect** è stato proposto come uno strato di autenticazione che utilizza il framework OAuth2. In questo caso, oltre al **token di accesso**, viene restituito anche un **ID token** con le informazioni sull'identità dell'utente.

## 12.3. TRUST MANAGER

### CONTROLLO ACCESSI DINAMICO:

Il controllo degli accessi dinamico va oltre il tradizionale approccio statico. Questo processo decide quando concedere o negare una richiesta di accesso in base non solo alle credenziali dell'utente, ma anche alla criticità dei dati richiesti e al grado di fiducia assegnato al richiedente. Il grado di fiducia viene stimato in base alle interazioni passate e ai punteggi di reputazione raccolti.



### GESTIONE DELLA FIDUCIA:

La gestione della fiducia è fondamentale per permettere ai nodi di valutare la fiducia e il comportamento corretto degli altri nodi con cui interagiscono. Questo

processo consente di implementare un controllo degli accessi dinamico, in grado di rilevare e allontanare nodi malintenzionati che sembrano presentare credenziali di sicurezza valide. La gestione della fiducia nell'IoT non è ancora completamente risolta e presenta due problematiche principali.

- Evitare stime maligne della fiducia e gestire gli attacchi mirati alla gestione della fiducia;
- La gestione della fiducia non deve consumare eccessivamente energia dai sensori.

## MODELLO DI RACCOMANDAZIONE BASATI SULLA FIDUCIA:

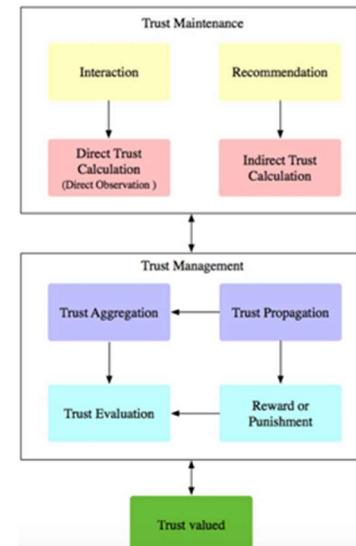
Un modello di raccomandazione basato sulla fiducia si compone di tre moduli principali: manutenzione della fiducia, gestione della fiducia e valore della fiducia. Quando un nodo non è in grado di valutare direttamente il comportamento di un altro nodo, costruisce un percorso di fiducia affidabile basato sulla conoscenza indiretta e le opinioni ottenute da un nodo intermedio o da una catena di parti fidate.

## RACCOLTA DELLA FIDUCIA:

Un nodo inferisce la fiducia prima tramite l'esperienza diretta, che può avvenire attraverso l'interazione uno-a-uno con i nodi vicini o l'osservazione diretta dei comportamenti sociali e delle attitudini tra i nodi. Per ottenere un grado complessivo di affidabilità, il nodo aggrega la fiducia diretta con le raccomandazioni ricevute. Il metodo di aggregazione della fiducia esclude raccomandazioni diffamatorie, assegnando un basso punteggio di fiducia ai nodi malintenzionati.

## FEEDBACK E AGGIORNAMENTI DELLA FIDUCIA:

Una volta raccolto il fattore di fiducia da un nodo target e valutato il valore di fiducia tramite il modello proposto, il risultato finale viene propagato come raccomandazioni. Non appena un nodo riceve una raccomandazione, esegue il processo di aggregazione. Al completamento della transazione, il nodo viene premiato o punito in base al comportamento mostrato, tramite un feedback positivo o negativo. I nodi con alta affidabilità vengono coinvolti nelle interazioni successive, mentre i nodi con basso punteggio vengono isolati.



## 12.4. AUDIT

L'audit consiste nella raccolta, memorizzazione e distribuzione di tutte le informazioni relative alle richieste ricevute e agli esiti conseguenti (insieme all'identità dell'utente che ha richiesto l'operazione). L'audit rappresenta un elemento fondamentale per garantire un alto livello di sicurezza, permettendo di rilevare tentativi di violazione della sicurezza volti a compromettere i meccanismi di protezione del sistema, promuovendo allo stesso tempo il miglioramento delle misure di sicurezza per prevenire future violazioni.

## OBIETTIVI DELL'AUDIT:

- La prima cosa da stabilire è ***l'oggetto dell'audit***. ISACA definisce l'IoT come qualsiasi cosa o persona che possieda software incorporato per interagire con altri oggetti animati o inanimati attraverso le reti. È necessario prendere in considerazione tutti i dispositivi IoT utilizzati nell'azienda e determinare l'oggetto dell'audit, rispondendo alla domanda fondamentale: *Cosa stiamo auditando?*;
- Una volta definito l'oggetto dell'audit, bisogna stabilire ***l'obiettivo dell'audit***: *Perché lo stiamo facendo?* Si consiglia di adottare una visione basata sul rischio per definire gli obiettivi dell'audit;
- Successivamente, bisogna determinare ***i limiti dell'audit***. Poiché i dispositivi non sono solo sensori, ma comprendono anche infrastrutture di supporto come apparecchiature di connettività, cloud, sistemi di archiviazione e algoritmi per l'elaborazione dei dati, è cruciale condurre una valutazione del rischio per stabilire l'ambito finale dell'audit

## STRATEGIA E APPROCCIO ALL'AUDIT:

- Durante il processo di audit, il team dovrebbe raccogliere sufficienti informazioni per identificare e selezionare l'approccio o la strategia di audit e iniziare a sviluppare il programma di audit. Non esistono standard universalmente accettati per la qualità, la sicurezza o la durabilità, né programmi di audit/assicurazione standardizzati;
- È essenziale un allineamento tra gli aspetti aziendali e i processi, assicurando che l'implementazione dell'IoT sia conforme alle esigenze aziendali e che i benefici aziendali vengano conseguiti come richiesto;

## CONTROLLI RELATIVI AI DATI E ALL'ANALISI:

I controlli relativi ai dati devono garantire che i dati che formano una parte fondamentale dell'IoT siano protetti. Inoltre, i controlli relativi all'analisi e all'apprendimento sono necessari per assicurarsi che l'analisi sia etica e consenta un uso affidabile dei dati, e che i risultati dell'analisi possano essere applicati alla presa di decisioni.

Alcune delle fonti di garanzia applicabili per ciascuna delle componenti precedentemente menzionate sono specifiche per l'IoT, mentre altre sono più generiche e dovrebbero essere applicate agli aspetti rilevanti dell'IoT.

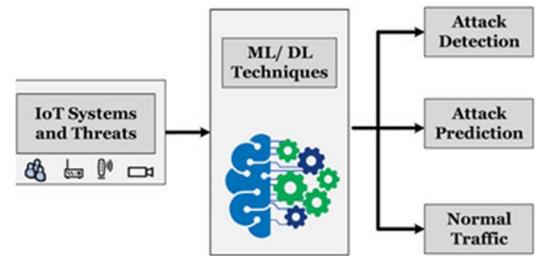
## IDS (SISTEMA DI RILEVAMENTO INTRUSIONI):

Un ***Intrusion Detection System (IDS)*** è un dispositivo o un'applicazione software che monitora una rete o sistemi alla ricerca di attività maligne o violazioni delle politiche. Quando viene rilevata un'intrusione o una violazione, solitamente viene segnalata a un amministratore o centralizzata tramite un sistema di ***Security Information and Event Management (SIEM)***. Un sistema SIEM raccoglie gli

output da più fonti e utilizza tecniche di filtraggio degli allarmi per distinguere tra attività maligne e falsi allarmi.

## STRUTTURA COMUNE DI UN IDS:

- **Modulo di raccolta dati:** raccoglie i dati che potrebbero contenere prove di un attacco;
- **Modulo di analisi:** rileva gli attacchi dopo aver elaborato i dati raccolti, questo modulo può essere implementato utilizzando diverse tecniche, tra cui l'**Machine Learning (ML)** e il **Deep Learning (DL)**, che sono particolarmente adatti per apprendere comportamenti benigni e anomali, basandosi su come i dispositivi IoT interagiscono tra loro. I metodi ML/DL possono anche prevedere nuovi attacchi, che spesso sono diversi da quelli precedenti, imparando dai campioni legittimi esistenti;
- **Meccanismo di reportistica:** segnala l'attacco identificato.



## TECNICA DI RILEVAMENTO:

- **Basato su firme (Signature-based):** utilizza un repository di firme di attacco e confronta il traffico di rete o le azioni del sistema con queste firme. Se viene trovata una corrispondenza, viene generato un allarme;
- **Basato su anomalie (Anomaly-based):** si basa su un profilo di comportamento normale per l'ambiente monitorato. Ogni deviazione fuori dai limiti stabiliti viene segnalata, ma senza classificare il tipo di attacco rilevato. Questo approccio permette di identificare attacchi sconosciuti, in quanto la normalità viene appresa dal sistema;
- **Basato su specifiche (Specification-based):** richiede che il comportamento normale venga definito manualmente tramite un repository di regole e intervalli di deviazione specificati da un esperto umano.



### 13. BLOCKCHAIN (capitolo preso dagli appunti di sicurezza dei dati tranne pezzo finale)

La blockchain nasce dall'esigenza di dover trasferire un bene sulla rete. In pratica, esistono due nodi di una rete che intendono scambiarsi delle risorse (asset), e l'obiettivo è riuscire a trasferire tale bene in maniera affidabile e sicura, anche se i nodi possano essere bizantini, la rete sia poco affidabile o ci sono degli attaccanti che vogliono manomettere il trasferimento.

I **modelli di fallimento** di questa tecnologia sono:

- **Fallimenti dei nodi:** crash o hang;
- **Fallimenti dei link:** crash persistente o intermittente;
- **Malfunzionamenti di rete:** perdita, alterazione o ritardo anormale di pacchetti o partizionamento della rete. I **modelli di attacco**, invece, sono:
  - **Replay Attack:** una transazione valida viene maliziosamente ripetuta o ritardata, con l'intento di manomettere il trasferimento;
  - **Man-in-the-middle Attack:** la comunicazione è osservata e alterata da una terza parte non autorizzata;
  - **Masquerade Attack:** un nodo malizioso utilizza un'identità forgiata ad hoc per la comunicazione. Un caso particolare (impersonation) è che un nodo impersona un altro legittimo;
  - **Byzantine Behaviour:** un nodo si comporta in modo non previsto;

#### TRANSAZIONI CON MEDIATORE:

La gestione delle transazioni è realizzata impiegando una entità di terze parti (es. una banca) che convalida, supervisiona e preserva le transazioni. Tale soluzione è implementata ad esempio con il protocollo di aggiornamento a due fasi (2PC - Two-phase commit protocol). In generale, quando si trasferiscono soldi lo si fa sempre con un **mediatore**. La soluzione con un mediatore consente di gestire attacchi del tipo Masquerade o Byzantine, e si richiede che la terza parte sia affidabile.

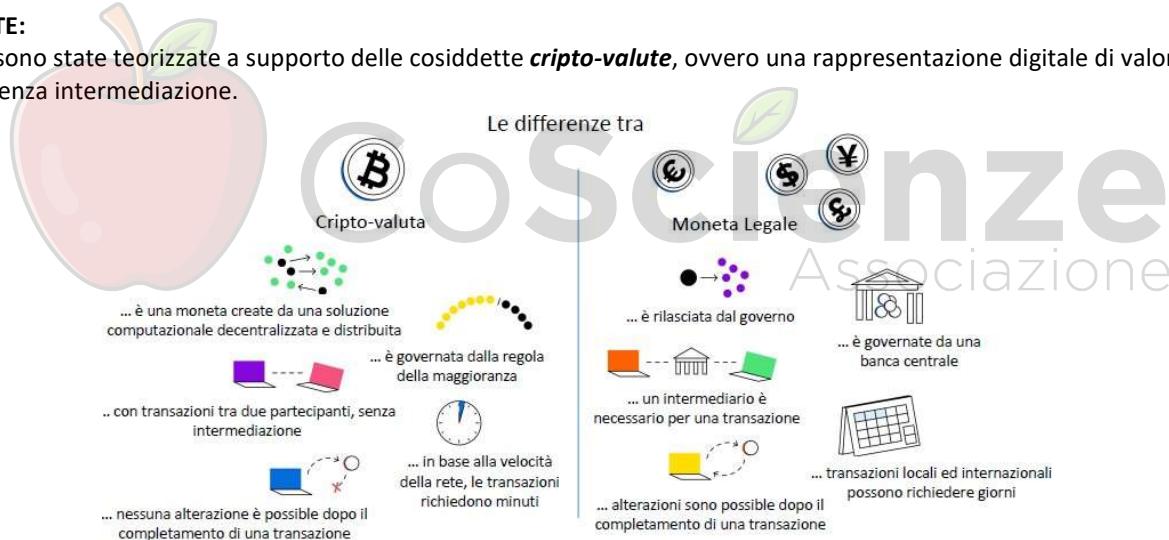
#### TRANSAZIONE SENZA MEDIATORE:

Per inviare un bene senza la collaborazione di un'entità di terze parti si può definire un protocollo per eseguire transazioni affidabili e sicure in un ambiente inaffidabile e insicuro. Tale soluzione presenta diverse problematiche legate a verifica dell'integrità del messaggio ricevuto, verifica dell'identità dei nodi partecipanti, verifica della validità delle transazioni, ecc...

Il primo ambito di applicazione (2008) è quello finanziario, in particolare per la gestione delle criptovalute. Lo scopo era quello di eseguire transazioni sicure utilizzando un'infrastruttura non sicura. In generale, l'obiettivo della tecnologia Blockchain è creare un ambiente decentralizzato in cui nessuna "terza parte" abbia il controllo delle transazioni e dei dati (**No trusted entities**, esempio banca).

#### CRIPTO-VALUTE:

Le blockchain sono state teorizzate a supporto delle cosiddette **cripto-valute**, ovvero una rappresentazione digitale di valore basata sulla crittografia e senza intermediazione.



Le cripto-valute rientrano nel caso delle valute virtuali ma sono diverse dalle valute digitali:

Una **valuta virtuale** è una rappresentazione digitale di valore che non è né emessa da una banca centrale né da un'autorità pubblica, né necessariamente collegata a una valuta a corso legale, ma è accettata come mezzo di pagamento e possono essere trasferiti, archiviati o scambiati elettronicamente (ad esempio, valuta in game).

Le cripto-valute sono **valute virtuali bidirezionali**.

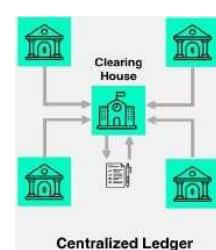
Una **valuta digitale** è una rappresentazione nel virtuale di una valuta a corso legale.



#### LEDGER (LIBRO MASTRO):

Il **ledger** (o **libro mastro**) è un registro in cui sono contenute tutte le transazioni eseguite tra i partecipanti di un sistema. Il libro mastro è caratterizzato da scritture sistematiche, ovvero le scritture sono registrate in base alla data e all'oggetto a cui si riferiscono. Con riferimento ad ogni oggetto, le operazioni sono indicate rispettando l'ordine cronologico.

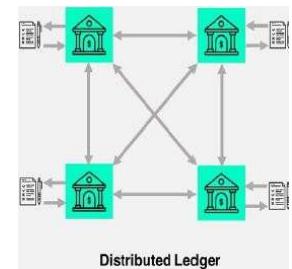
Si tratta di un "append-only multi-party system of record", ovvero un **"log"** di transazioni, "append-only" nel senso che si può scrivere in successione e non si può cancellare e "multi-party system of record" nel senso che le scritture non sono fatte da una sola organizzazione ma da più partecipanti (ognuno col proprio ledger).



In un **distributed ledger** (*libro mastro distribuito*), ciascun nodo della rete memorizza record (transazioni) in una copia locale del libro in maniera sequenziale, sotto forma di **blocchi** ordinati temporalmente.

Affinché i blocchi siano considerati validi, i partecipanti devono raggiungere un **quorum** di consenso (accettati dalla maggioranza).

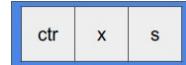
Il libro mastro è costruito come una catena di blocchi, ogni blocco (ad eccezione del **genesis block**, ovvero il primo blocco della catena) contiene un'informazione riguardo il blocco precedente nella catena.



## BLOCCO:

Un **blocco** è una collezione di *transazioni*, e sono strutturate in tre parti:

1. **Header**, dove è presente una informazione di controllo (ctr);
2. I **dati** veri e propri (x);
3. Riferimento al **blocco precedente**.



In generale, questa struttura viene chiamata **intestazione del blocco** e il dato dipende dalla blockchain:

- In Bitcoin memorizza i dati finanziari (basati su "UTXO");
- In Ethereum memorizza i dati del contratto (basato sull'account);
- In Namecoin memorizza i dati del nome;



Per inviare un bene senza la collaborazione di un'entità di terze parti si può definire un **protocollo** per eseguire transazioni affidabili e sicure in un ambiente inaffidabile e insicuro. Tale soluzione presenta diverse problematiche legate a:

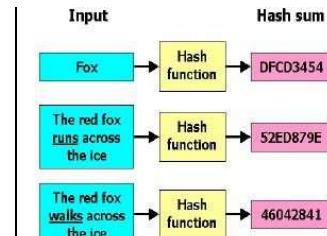
- Verifica dell'**integrità** del messaggio ricevuto, ovvero il contenuto non deve essere manomesso durante il tragitto;
- Verifica dell'**identità** dei nodi partecipanti;
- Verifica della **validità** delle transazioni;

## HASH E PARADOSSO DEL COMPLEANNO:

Le **funzioni hash** restituiscono una stringa di poche centinaia di bit (definita hash) a partire da una sequenza di bit di lunghezza arbitraria.

Le funzioni hash crittografiche cercano di rendere **computazionalmente inammissibile** trovare due sequenze che producono la stessa impronta.

La possibilità che due sequenze di bit sottomesse alla stessa funzione di hash diano origine allo stesso valore (collisione) è definito «paradosso del compleanno»: la probabilità che almeno due persone in un gruppo compiano gli anni lo stesso giorno è largamente superiore a quanto potrebbe dire l'intuito.



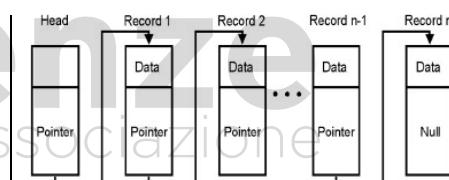
## CATENA DI BLOCCHI:

L'hash viene utilizzata per l'integrità dei dati, ma viene anche utilizzata per proteggere i blocchi quando entrano nella catena.

Un blocco è collegato al precedente mediante un hash del blocco precedente. La modifica di un blocco della catena da parte di un nodo comporterebbe la generazione di un valore di hash differente.

Gli altri nodi possono verificare l'**integrità dei blocchi**, andando a verificare se è presente il valore di hash atteso altrimenti il blocco è stato modificato, e rifiutare modifiche di transazioni già validate.

Il contenuto del libro mastro è reso così **immutable**.



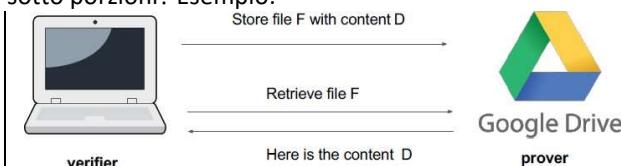
## AUTENTICAZIONE CON ALBERI HASH:

Quando si gestiscono le funzioni hash, generalmente si ha un problema di proteggere informazioni e assicurare integrità.

Un blocco è formato da tanti pezzi e quando si fa l'hash di un blocco lo si fa sull'intero blocco. Questo lo si può trasporre con un problema dei file, dove

un file ha più sotto porzioni e bisogna gestire l'integrità dell'intero file o delle sotto porzioni? Esempio:

1. L'utente desidera archiviare un file su un server;
2. Il file ha un nome F e un contenuto D;
3. Gli utenti desiderano recuperare il file F in un secondo momento e si ottiene il contenuto D.



Nel recuperare il file in un secondo momento, si vuole che il D che si recupera è uguale al D che si è memorizzato in precedenza (**integrità** da garantire). Ma se il provider fosse malizioso, potrebbe restituire un  $D' \neq D$ , per accorgersi che il contenuto è diverso, bisognerebbe conservare D. Quindi, si conserva il file F, lo si archivia sul server e dopo di che quando si chiede la restituzione del file F e restituisce  $D'$  anziché D, si confronta la copia e si prendono provvedimenti.

Una **soluzione banale** è che l'utente non elimina D, ma una tecnologia di questo tipo serve per archiviare e non avrebbe senso se si deve comunque conservare i file.

Una **soluzione con Hash** è:

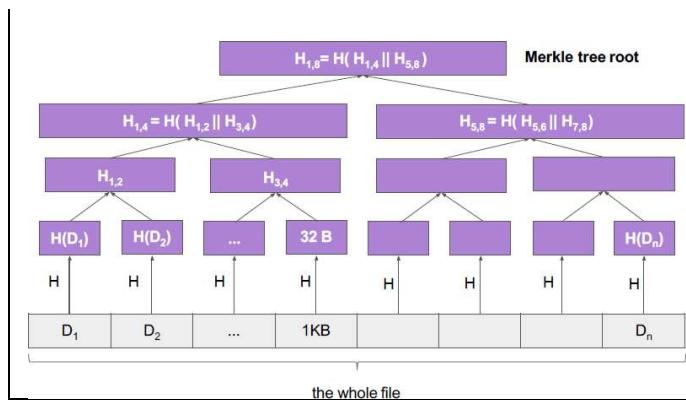
- L'utente invia il file F con i dati D al server;
- Il server archivia (F, D);
- L'utente memorizza H (D), elimina D;
- L'utente richiede F dal server;
- Il server restituisce  $D'$ ;
- L'utente confronta  $H (D') = H (D)$ ?

Se l'utente è interessato al recupero solo di una parte del file, per controllare l'integrità di quella sotto-parte si ha comunque necessità di scaricare l'intero file (in quanto l'hash è applicato sull'intero file), e questo sulla rete può essere un problema di latenza, in quanto bisogna scaricare tutto il file, calcolare l'hash e poi effettuare il controllo. Per evitare tutta questa latenza si usa il **Merkle Three**, il suo funzionamento è il seguente:

Si calcola l'hash di tutte le sotto porzioni (chunck) del file, dove ogni blocco utilizza una funzione hash crittografica.

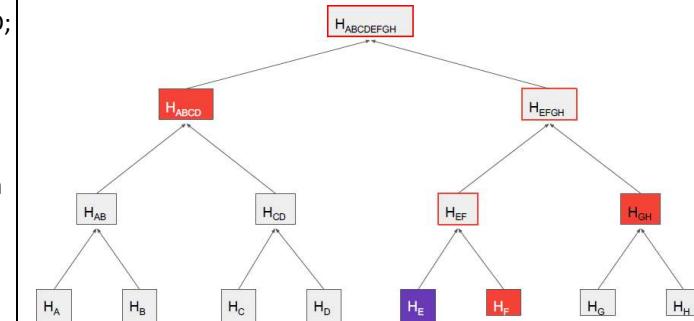
Dopo di che si combinano gli hash a coppie per creare un albero binario, e ogni nodo memorizza l'hash del concat dei suoi figli.

Si itera il processo fintanto che non si ha un unico hash, che sarebbe la radice del Merkle Three.



Quindi si ha una nuova soluzione utilizzando i **Merkle Three**:

- L'utente crea l'MTR radice del Merkle Tree dai dati del file iniziale D;
- L'utente invia i dati del file D al server;
- L'utente elimina i dati D, ma memorizza MTR (32 byte);
- L'utente richiede il blocco x dal server;
- L'utente restituisce il blocco x e una breve prova di inclusione  $\pi$ ;
- L'utente controlla che il blocco x sia incluso in MTR usando la prova  $\pi$ .



La **prova di inclusione** funziona nel modo seguente:

- Il Prover invia un chunck (es. HE);
- Il Prover invia i vicini lungo il percorso che collega la foglia a MTR (es. HF, HGH, HABCD);
- Il Verifier calcola gli hash lungo il percorso che collega la foglia a MTR;
- Il Verifier controlla che radice calcolata corrisponda a MTR (es. HABCDEFGH).

La prova di inclusione richiede  $|\pi| \in \Theta(\log |D|)$ . Se l'avversario può presentare la prova di inclusione per un nodo foglia errato, allora possiamo compromettere la funzione hash.

### IDENTITÀ DEI NODI:

Sebbene applicate a problemi in cui si vuole fare a meno di **trusted entities** (terze parti), le tecnologie blockchain utilizzano anche la firma digitale (e le **trusted authorities** di certificazione) per problemi di sicurezza tipici dei sistemi distribuiti:

- **Authentication**: autenticazione dei nodi della rete;
- **Integrity**: verifica di violazioni dell'integrità dei messaggi scambiati;
- **Non-repudiation**: Non ripudiabilità da parte dei nodi dei messaggi inviati.

### VALIDITÀ DELLE TRANSAZIONI:

Si consideri un nodo che desidera "vendere" un bene attraverso una transazione. Il nodo predispone da sé la transazione, che deve sottoporre agli altri nodi affinché sia approvata (validata) da una maggioranza. È possibile che il nodo provi a "spendere" concorrentemente due volte lo stesso bene (double spending):

- In maniera **consapevole**: byzantine behaviour;
- In maniera **inconsapevole**: malfunzionamenti di rete (duplicazione di pacchetti), replay attack.

Un primo passo per risolvere il **problema del double spending** è determinare se un messaggio è "fresco" o è una copia ritrasmessa. È necessario poter identificare univocamente un messaggio contenente un blocco di transazioni eseguite.

Un **nonce** è definito come "un valore variabile nel tempo che ha al massimo una possibilità trascurabile di ripetersi, ad esempio un valore casuale che viene generato di nuovo per ogni utilizzo, un timestamp, un numero di sequenza o una combinazione di questi".

Un nonce può quindi essere ottenuto attraverso un "**timestamp**", e per garantire che ogni blocco sia caratterizzato da un valore differente è necessario che tale valore sia generato da una fonte "fidata".

### CERTIFICAZIONE DEL TEMPO:

La certificazione del tempo serve a verificare quando un documento è stato creato, oppure è stata apportata l'ultima modifica in maniera affidabile e non falsificabile. Si impiega un **Trusted Time Server (TTS)** ed esistono due soluzioni:

1. L'utente invia al TTS e una copia del documento, e poi il TTS aggiunge data e ora. Conserva il documento nella cassetta di deposito;
2. Si impiega una soluzione crittografica per proteggere la privacy del documento, e ridurre i costi di trasmissione e memorizzazione, dove l'utente invia al TTS l'hash del documento e il TTS aggiunge data, ora e firma, per poi inviare questo certificato all'utente.

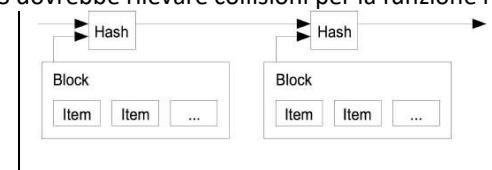
Della seconda soluzione esiste lo standard **ANSI ASC X9.95** che indica che un timestamp attendibile viene emesso da un'**autorità di timestamp** (TSA). Un hash viene calcolato e inviato alla TSA. La TSA concatena un timestamp all'hash e calcola un hash, che viene firmato digitalmente con la chiave privata della TSA. Questo hash firmato e il timestamp vengono restituiti. Chiunque si fidi del timestamper può quindi verificare che i dati non siano stati creati dopo la data che il timestamper garantisce.

Il valore NONCE, essendo gestito da un unico punto della rete, si ha la garanzia di verificare se due documenti sono lo stesso.

Indipendentemente della correttezza del tempo certificato, si vuole che due documenti sottoposti sequenzialmente al TTS abbiano un

nonce che esprima questa sequenzialità:

- Ogni certificato rilasciato contiene l'hash del documento, data, ora, firma del TSS e l'hash del certificato precedente (o di parte di esso);
  - Complessivamente i certificato formano una catena di blocchi;
  - È inammissibile inserire successivamente un certificato all'interno della catena, il TTS dovrebbe rilevare collisioni per la funzione hash.
- Questa soluzione comporta una serie di vantaggi:
- Aggiungere il timestamp rende l'hash più resistente rispetto alle collisioni;
  - Risolve il problema del double spending assegnando un ordine temporale alle transazioni e pubblicandole.



#### PROOF-OF-WORK:

Per svincolarsi dall'utilizzo di un timestamp server (*approccio centralizzato*), alcune soluzioni blockchain (*approccio decentralizzato*) sfruttano un meccanismo differente per la generazione dei nonce. Tale meccanismo è definito **Proof-of-Work (PoW)** (da notare che il PoW non realizza il consenso ma è un meccanismo per proteggere il consenso), che funziona come segue:

Per proporre un nuovo blocco da aggiungere al ledger, un nodo deve dimostrare di aver utilizzato abbastanza risorse di calcolo per risolvere un enigma matematico, il cui risultato è inserito nel blocco stesso.

La Proof-of-Work implica *la computazione di un valore di hash con un determinato numero di bit iniziali pari a zero*,  $H(ctr \parallel x \parallel s) \leq T$ . Ogni nodo (**miner**) cerca di trovare un **nonce** tale da soddisfare il vincolo sul numero di zero prodotti dalla funzione di hash.

Il lavoro medio richiesto è esponenziale rispetto al numero di bit zero richiesti e può essere verificato eseguendo un singolo hash. Tale operazione è molto onerosa dal punto di vista computazionale e richiede la collaborazione di diverse CPU per poter essere realizzata. L'attacco del 51% diventa molto oneroso.

#### TRANSAZIONI CON CONSENSO DISTRIBUITO:

Tutti i nodi della rete sono a conoscenza di tutte le transazioni, dato che ognuno ha la copia del ledger, e partecipano attivamente alla validazione dei blocchi attraverso il raggiungimento del consenso. Il modello di sicurezza deve contemplare anche il cosiddetto "51% Attack", un gruppo di nodi maliziosi possiede più del 50% della potenza computazionale della rete.

"Nessun algoritmo può garantire il raggiungimento del consenso in un sistema asincrono nel caso di anche un unico fallimento per crash di un processo" (**Teorema di Impossibilità**).

Per raggiungere il consenso in sistemi asincroni in presenza di fallimenti si può:

- **Rilassare i vincoli di consenso** (ciò che fa BitCoin): È consentito che in alcuni casi non si raggiunga il consenso tra tutti i nodi della rete, ovvero solo un sottoinsieme di nodi raggiungono il consenso, e quindi "51% Attack" possibile.
- **Rendere meno "asincrono" il sistema**, sfruttare i periodi di sincronia (ciò che fa Hyperledger Fabric).

#### BLOCKCHAIN FORKING:

Il rilassamento dei vincoli di consenso può dare origine a **biforazioni** della blockchain.

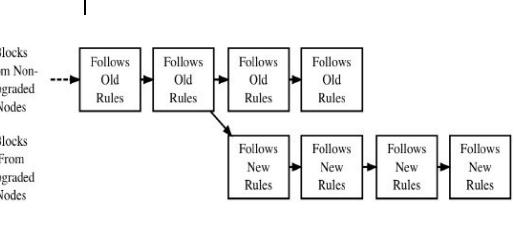
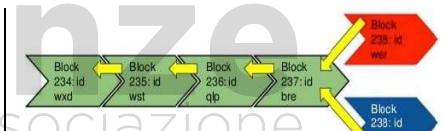
Nel caso di produzione "simultanea" di due blocchi alcuni blocchi aggiungeranno alla propria blockchain il blocco rosso ed altri il blocco blu, fino ad arrivare alla realizzazione di una "forked blockchain".

All'atto della produzione del blocco successivo, esso farà riferimento ad uno solo dei due blocchi precedentemente creati. Tale operazione comporta la rimozione del ramo della blockchain composto dal nodo blu.

In un certo istante di tempo la comunità può ritenere necessario cambiare le regole della blockchain in modo che determinate transazioni ritenute "non valide" fino a tale istante siano considerate " valide" a partire da tale istante e nel futuro.

Tale condizione genera un **hard fork**, che comporta la generazione "voluta" di un nuovo ramo della blockchain per distinguere i nodi che seguono il nuovo regolamento rispetto quelli fedeli al vecchio.

In qualsiasi momento i nodi possono decidere di abbandonare un ramo per seguire l'altro.

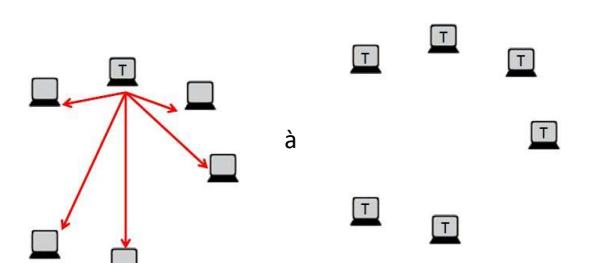


#### CONSENSO NAKAMOTO (BITCOIN):

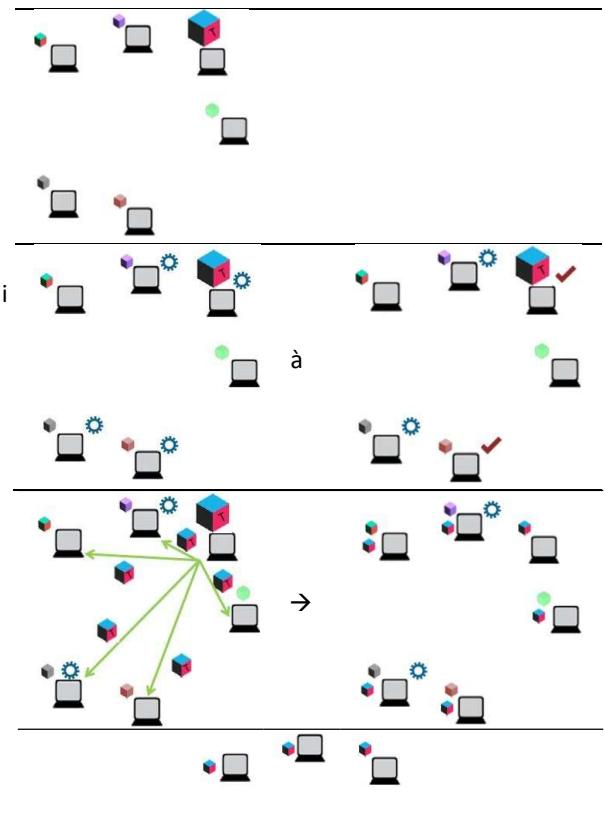
L'algoritmo di consenso che ha adottato il BitCoin prende il nome di **consenso di Nakamoto** e che adotta la Proof-of-Work. Funziona come segue:

1. Ogni nuova transazione viene inviata in broadcast a tutti i nodi.

Un nodo genera una nuova transazione T e ne fa il broadcast a tutti i nodi. Gli altri nodi ricevono T.

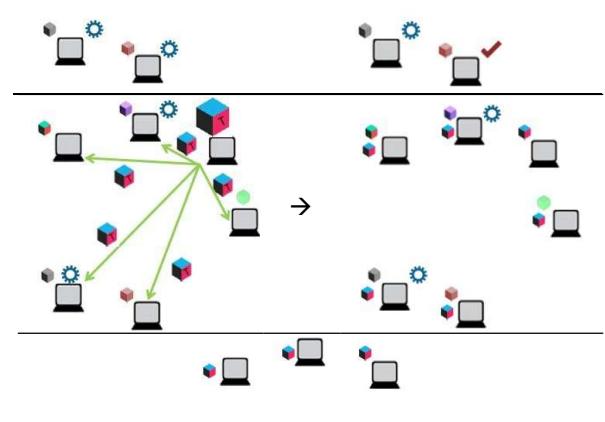


2. Ogni nodo colleziona le nuove transazioni in un blocco (da notare che il consenso è sul blocco e non sulla transazione).



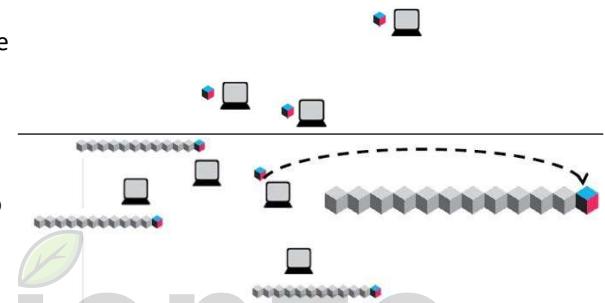
3. Ogni nodo cerca una PoW "difficile" (cioè molte computazioni) per il proprio blocco. La PoW è il calcolo di un valore (anch'esso un nonce) tale che un hash di blocco + nonce inizi con uno stabilito numero di zeri.

I nodi cercano una Proof-of-Work per il proprio blocco (l'ingranaggio rappresenta il calcolo della PoW). Dopo di che alcuni nodi completano la PoW.



4. Quando un nodo trova una PoW, invia in broadcast a tutti i nodi il blocco, contenente le transazioni e la PoW.

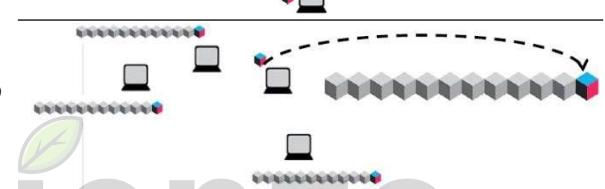
Successivamente i nodi ricevono il blocco.



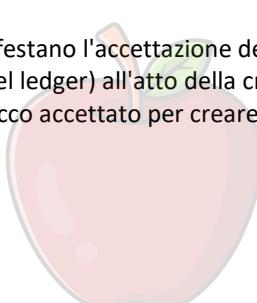
5. Un nodo accetta il blocco solo se:

- Tutte le transazioni ( contenute nel blocco ) sono valide;
- Le transazioni non sono relative ad un bene già speso ( double spending );
- La PoW è valida. A tale scopo il nodo calcola a sua volta l'hash del blocco, e verifica che abbia lo stesso numero di zeri iniziali.

Tutti i nodi accettano il blocco ( la blockchain è ancora inalterata ).



6. I nodi manifestano l'accettazione del blocco aggiungendolo alla blockchain ( contenuto nel ledger ) all'atto della creazione del blocco successivo, utilizzando l'hash del blocco accettato per creare il prossimo blocco.



# CoScienze

Associazione

Il tutto avviene in maniera distribuita e decentralizzata, quindi due nodi possono fare concorrentemente il broadcast di blocchi differenti (passo 4). Nodi diversi possono ricevere i due broadcast in ordine diverso, e portare all'accettazione di alcuni blocchi prima di altri blocchi. Ciascun nodo lavora sul primo blocco che riceve, ma conserva l'altro ramo (soft fork). Per ritornare ad uno stato di consistenza, l'indecisione si risolve quando un ramo diventa più lungo, cioè i nodi che lavorano sul ramo meno lungo lo abbandonano e proseguono a lavorare sulla catena più lunga.

È opportuno incoraggiare la cooperazione dei nodi alla creazione del consenso (validazione delle transazioni "corrette"), affinché un blocco sia validato ci deve essere una maggioranza di nodi corretti e che spendono risorse (costruendo la PoW) poiché la PoW è onerosa anche i nodi corretti potrebbero non essere ben disposti a cooperare.

È prevista una politica di incentivi, incoraggiare processi corretti a collaborare nonostante l'onere computazionale, mettendo in minoranza i processi maliziosi. I nodi maliziosi dovranno avere ancora più potenza di calcolo per "contrastare" molti nodi corretti che validano i blocchi. La prima transazione di ogni blocco assegna un compenso al nodo creatore del blocco.

## BLOCKCHAIN:

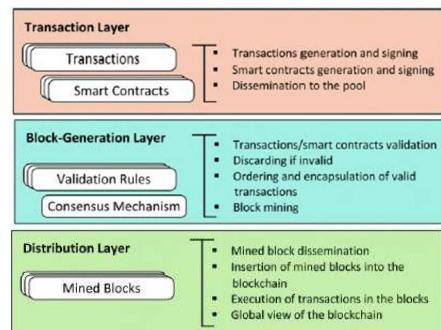
Una **blockchain** è un libro mastro pubblico distribuito (distributed public **ledger**) di transazioni o eventi digitali eseguiti e condivisi tra i partecipanti. Ogni transazione riportata nella blockchain è validata tramite il raggiungimento del **consenso** tra i nodi del sistema. Una volta memorizzate, le informazioni non possono essere cancellate né modificate. Ogni blocco contiene uno o più record (un record per transazione). Come in un libro mastro, la storia delle registrazioni è immutabile e può essere verificata a partire dal primo blocco (*genesis block*). Le transazioni di Bitcoin sono in chiaro e non cifrate, usa un meccanismo di pseudo anonimizzazione per non collegare l'identità di un utente alle sue attività su Bitcoin. L'attività non viene crittografata perché risulta difficile condividere le chiavi, in quanto, nel consenso Nakamoto, oltre alle transazioni bisognerebbe condividere anche la chiave pubblica in broadcast, ma questa operazione comporta l'identificazione dei nodi, cosa non concessa in Bitcoin.

Le proprietà che una Blockchain fornisce sono:

- **Pubblica Verificabilità**: ogni transazione può essere verificata da ogni partecipante, non esiste una confidenzialità ma un anonimato;
- **Trasparenza**: ogni partecipante ha accesso ad un sottoinsieme di informazioni;
- **Privacy**: l'identità di chi esegue una determinata transazione deve essere tutelata;
- **Integrità**: le informazioni non vengono modificate da fonti non autorizzate;
- **Ridondanza**: dati ripetuti per ogni partecipante del sistema;
- **Assenza di una "Trust Anchor"**.

Sono classificabili in base all'ambito:

- **Public**: tutti i blocchi sono visibili a tutti i nodi e ogni nodo può partecipare al consenso (es Bitcoin e Ethereum);
- **Private**: una specifica organizzazione decide quali nodi possono leggere i blocchi e partecipare al consenso (throughput delle transazioni alto).
- **Consortium**: pochi nodi predeterminati possono leggere i blocchi e partecipare al consenso. La stratificazione tecnologica di blockchain può essere la seguente:
  - Il Transaction Layer definisce il linguaggio di codifica e i criteri utilizzati durante la generazione di transazioni e smart contract. Entrambi devono essere firmati prima della loro diffusione per garantire il non ripudio e consentire il controllo dell'accesso e l'autenticazione del loro contenuto;
  - I processi di convalida delle transazioni e mining dei blocchi risiedono nel livello block-generation. Tutte le transazioni sono inserite in un blocco candidato, secondo regole di convalida. L'algoritmo di consenso viene adottato per garantire che tutti i nodi della rete abbiano una vista consistente dello stato della blockchain;
  - Il processo di distribuzione fa parte del livello di distribuzione, così come l'inserimento del mined block nella blockchain. Tale inserimento riesce solo se l'hash del blocco estratto è corretto, altrimenti viene scartato. Al momento dell'inserimento di un blocco, lo stato globale della blockchain cambia e la vista globale della blockchain viene aggiornata.



## 13.2. CONSENSO NELLE BLOCKCHAIN

In una rete blockchain, ogni partecipante può validate transazioni e proporre nuovi blocchi. L'obiettivo del **protocollo di consenso nelle blockchain** è di garantire che tutti i nodi partecipanti concordino sulla storia comune delle transazioni nella rete. Le proprietà del consenso della blockchain sono:

- **Termination**: Da ogni nodo onesto, una nuova transazione è sia scartata o accettata nella blockchain, all'interno del contenuto di un blocco;
- **Agreement**: Ogni nuova transazione e il suo blocco che la contiene deve essere sia scartata o accettata da tutti i nodi onesti. Un blocco accettato deve avere lo stesso numero di sequenza per ogni nodo onesto;
- **Validity**: Se ogni nodo riceve uno stesso blocco/transazione valida, esso deve essere accettato nella blockchain;
- **Integrity**: Per ogni nodo onesto, tutte le transazioni accettate devono essere consistenti tra loro (senza double spending). Tutti i blocchi accettati devono essere correttamente generati e collegati con hash in ordine cronologico.

Termination e validity rappresentano la **liveness** del sistema, ovvero se si arriva a terminazione del consenso alcune cose sono accettate.

Agreement e

integrity rappresentano la **safety** del sistema, il consenso che si arriva è coerente e consistente con quanto accettato precedentemente. L'*agreement classico* significa arrivare ad una decisione, nell'*agreement* delle blockchain si arriva a decisione con lo stesso numero di sequenza nei blocchi e quindi si impone il total ordering (*agreement* aumentato col **total ordering**), ovvero i nodi partecipanti hanno lo stesso ordine di accettazione e non di invio, così da garantire la consistenza di sequenzialità. Esempio se si accettano 10 bitcoin e successivamente spendo 5, il contrario potrebbe non consistere.

L'**integrity** impone la correttezza dell'origine di transazioni e blocchi, consentendo una protezione nel confronto del double-spending e favorendo la tamper-proofing nella blockchain.

Il teorema CAP viene applicato anche alle blockchain. Per i meccanismi di consenso, liveness e safety hanno una diretta correlazione col teorema CAP:

- La **liveness** garantisce che il processo di consenso completa sempre i suoi round. Anche se non si arriva a consenso, il meccanismo non attende indefinitamente, garantendo la sua availability;
- La **safety** garantisce che i suoi partecipanti sono nello stesso stato dopo un round, garantendo la consistenza nella rete.

Nelle reali implementazioni di soluzioni blockchain, non è mai possibile ottenere sia Consistenza che Availability, perché devono affrontare la **tolleranza alle partizioni**. Il sistema restituirà un errore o un timeout se non è possibile garantire che particolari informazioni siano aggiornate a causa del partizionamento di rete (nodi in errore). Ogni richiesta alla rete riceve una risposta, anche se la rete non può garantire che sia aggiornata a causa del partizionamento di rete (nodi in errore).

Tra le due opzioni rimanenti, i sistemi di tipo **permissionless** (come Bitcoin) in un gruppo aperto di nodi scelgono di privilegiare di garantire

l'Availability anziché la Consistency, per poter essere in grado di utilizzare/inviare/ricevere criptovalute. I casi di fork che rappresentano i momenti di inconsistenza sono risolti nel tempo, così da garantire l'Eventual Consistency.

Tra le due opzioni rimanenti, i sistemi di tipo **permissioned** in un gruppo chiuso di nodi scelgono di privilegiare di garantire la Consistency anziché

l'Availability, perché non sono focalizzate su criptovalute ma la gestione di dati in ambito distribuito. Il protocollo di consenso per blockchain si compone di 5 elementi:

1. La **proposta di un blocco**: generazione di blocchi e associazione di prove;
2. La **propagazione di informazioni**: disseminazione di blocchi e transazioni nella rete;
3. La **validazione di blocchi**: controllo dei blocchi per la generazione di prove e verifica della validità delle transazioni;
4. La **finalizzazione dei blocchi**: raggiungimento del consenso sull'accettazione di blocchi validati;
5. Il **meccanismo di incentivazione**: promozione di partecipanti onesti e creazione di token di rete.

## STATE MACHINE REPLICATION:

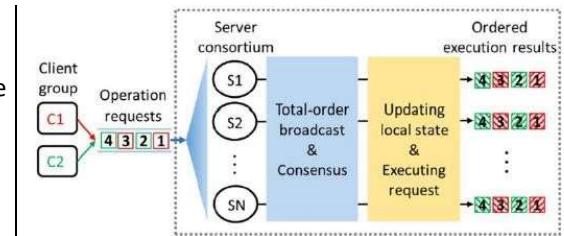
Il consenso su blockchain si ispira al meccanismo di **State Machine Replication (SMR)**. SMR stabilisce i seguenti requisiti:

1. Tutti i server iniziano con lo stesso stato iniziale;

2. Total-order broadcast: tutti i server ricevono la stessa sequenza di richieste secondo l'ordine di generazione dai client;
3. Tutti i server che ricevono la stessa richiesta emetteranno gli stessi risultati di esecuzione e termineranno nello stesso stato.

Un consorzio di N server accetta le richieste di operazioni dai client.

I server confermano il proprio stato prima di raggiungere il consenso ed eseguire le richieste. SMR è spesso realizzato in maniera leader-based, con un server primario (esempio S1) che riceve le richieste dai client e inizia la procedura di broadcast, mentre gli altri ricevono le stesse richieste e aggiornano il proprio stato locale in modo che corrisponda a quello del leader.



Sussiste una corrispondenza tra gli elementi del consenso nelle blockchain e quelli in SMR:

1. La proposta di blocchi corrisponde alle richieste di operazioni da parte dei client in SMR e il leader che inizia il consenso;
2. La propagazione delle informazioni corrisponde al reliable broadcast delle richieste di operazioni;
3. La validazione dei blocchi corrisponde alla verifica delle firme e l'esecuzione delle operazioni richieste;
4. La finalizzazione dei blocchi corrisponde al raggiungimento del consenso da parte dei server sullo stato corrente;
5. Il meccanismo di incentivo non trova una corrispondenza.

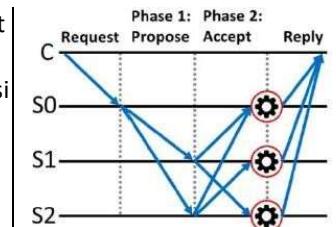
Questo perché SMR presuppone un gruppo ben definito di partecipanti che si presuppongono onesti.

L'**algoritmo di consenso di Paxos** è uno schema SMR progettato per garantire il consenso tollerante a guasti di tipo crash. Un proposer suggerisce un valore all'inizio e lo scopo del sistema è di far sì che gli acceptor concordano su un singolo valore e i learners apprendano tale valore dagli acceptor.

Il client è un learner e il leader tra i server è un proposer, mentre le repliche sono degli acceptors. Il client richiede il consenso su un singolo valore.

Il proposer propaga la richiesta agli acceptors, che si scambiano informazioni sui propri stati. Dopo essersi aggiornati allo stesso stato, tutti i server eseguono la richiesta, che la inviano al client.

Il client riceve i risultati dagli acceptor e formula l'esito a maggioranza. Quando il leader è indisponibile per crash, le repliche ne eleggono uno nuovo.



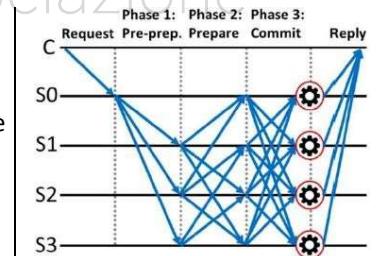
L'algoritmo tollera  $f$  crash dei server il cui numero  $N$  è maggiore o uguale di  $2f + 1$ , ma non tollera guasti bizantini perché i nodi ricevono un solo valore, ma se il mittente è bizantino non c'è modo di verificarlo, a meno che non si replica il valore nel tempo per vedere se i valori sono diversi.

### PBFT (PRACTICAL BYZANTINE FAULT TOLERANCE):

L'algoritmo di **Practical Byzantine Fault Tolerance (PBFT)** è uno schema SMR che tollera i guasti bizantini ed è diventato sinonimo di tolleranza ai guasti bizantini (BFT) nel contesto delle blockchain. Rispetto a Paxos, nel PBFT si ha una fase in più, nella prima fase (pre-prepare) viene inviato il valore proposto a tutti i nodi, poi gli acceptor hanno una fase di prepare dove scambiano le informazioni ricevute, successivamente una fase di commit in cui scambiano le loro decisioni. La seconda e terza fase consentono di tollerare i guasti bizantini.

Si compone di tre fasi:

1. Tutti i risultati inviati al client devono essere uguali, altrimenti il client decide a maggioranza. Il client invia una richiesta al nodo primario (leader), che la trasmette a tutti i nodi secondari (backup) assegnando un numero di sequenza.
2. I nodi secondari e il leader si accordano sull'ordinamento delle richieste, quando l'ordinamento è stato approvato, la richiesta viene eseguita e un risultato restituito al client.
3. Se il client riceve  $f + 1$  risposte identiche, si raggiunge consenso.



Il leader è scelto per ogni round dell'algoritmo o vista. I nodi sono ordinati in base al proprio identificativo, ed indicando il numero della vista con  $v$ , il leader è il nodo con identificativo  $i = v \bmod N$ .

Quando una replica nota un comportamento errato da parte del leader, se ne può richiedere la sostituzione con il meccanismo della view change e l'elezione di un nuovo leader. L'algoritmo consente di tollerare  $f$  guasti anche di natura bizantina, con  $N \geq 3f + 1$ , ovvero i nodi bizantini non riescono a far deviare il consenso raggiunto. La primitiva di comunicazione alla base delle implementazioni PBFT nel contesto delle blockchain è il **gossiping** (Reliable multicast), impiegato per la propagazione di messaggi di blocchi o transazioni.

A differenza della formulazione classica, nelle reti blockchain si ha la terminazione probabilistica: Per ogni nodo onesto, ogni nuovo blocco è sia scartato o accettato nella sua blockchain locale. Un blocco accettato può essere ancora scartato (fork) ma con probabilità decrescente in maniera esponenziale con la crescita della dimensione della catena.

Il principale problema con la PBFT è che richiede che i nodi verifichino la validità dei messaggi degli altri e che il numero di nodi attivi in un dato momento sia sempre noto, e ciò lo rende applicabile in contesti permissioned.

Un altro svantaggio è che i leader vengono sostituiti solo quando le view change vengono attivate dalla rete. L'opportunità di diventare un leader è quindi unfair e mancano pochi incentivi per entrare a far parte della rete. Blockchain elegge i leader in base alla difficoltà del lavoro svolto, che genera incentivi, anche se spreca potenza di calcolo.

La sicurezza di PBFT si basa sul voto in tre fasi con MAC (Message Authentication Code) per la verifica dei messaggi. Sebbene non consumi molte risorse di elaborazione, crea inevitabilmente problemi di scalabilità: in PBFT è impossibile espandersi oltre i 1000 nodi.

PBFT garantisce formalmente il requisito di Safety, infatti un fork è quasi impossibile e viene garantita la terminazione immediata.

Al contrario, la blockchain Nakamoto è più focalizzata sulla liveliness che sulla safety, e quindi i fork si verificano abbastanza frequentemente (consenso multiplo) e affinché un blocco sia sicuro la sua catena deve essere più lunga di un certo numero di blocchi. A tale scopo si è formulato un diverso algoritmo di consenso, quello di Nakamoto. Lo scopo è di evitare il consenso in gruppi chiusi, e di non

punire i singoli nodi in un gruppo aperto per essersi comportati in modo malevolo, ma bisogna disincentivare i nodi a replicare comportamenti malevoli.

Il processo di mining dei blocchi (ovvero calcolo della PoW) è stocastico, per cui è impossibile sapere con certezza chi troverà la soluzione, anche se al crescere della difficoltà i nodi capaci di portare avanti il processo diminuiscono. Questo scoraggia tutti gli agenti non disposti ad investire risorse economiche a partecipare al gioco. Con il crescere dell'uso, e quindi del valore, la difficoltà a minare bitcoin aumenta, disincentivando ulteriormente chiunque voglia attaccare la rete. Inoltre, il crescente valore costringe gli agenti onesti ad investire di più nella sicurezza dei propri nodi e, di conseguenza, della rete in generale. Le regole di validazione dei blocchi assicurano che nessun agente onesto accetti blocchi con informazioni scorrette.

#### CONSENSO NAKAMOTO:

Rispetto al **consenso di Nakamoto** implementato in BitCoin, è possibile trovare un parallelismo con le 5 componenti del consenso nelle blockchain:

1. La **generazione di blocchi** richiede una Proof-of-Work mediante la risoluzione di un puzzle crittografico con un determinato grado di difficoltà tale da mantenere un intervallo di generazione e un grado di protezione adatto;
2. Il **gossiping** viene impiegato per la distribuzione dei blocchi (transazioni appena ricevuto o localmente generati);
3. Un **blocco o transazione** deve essere **validata** prima di essere inviata in broadcast agli altri o collegata alla coda di una catena locale. La validità si realizza evitando la double-spending o controllando la PoW allegata al blocco;
4. La catena più lunga rappresenta il **raggiungimento del consenso** in caso di disaccordo (che ha causato la fork);
5. Chi ha generato un blocco accettato con successo può ottenere un **reward** o premio. Sottomettere una nuova transazione ha un costo monetario. L'impiego di intensive PoW è necessario per evitare e tollerare attacchi Sybil, a causa della natura permissionless e pseudonima di alcune reti blockchain. Un attaccante può ottenere facilmente delle nuove identità, ma la risoluzione di una PoW implica il consumo di risorse di hash, che può essere difficilmente falsata. Le reward per i blocchi e il costo delle transazioni serve ad incentivare i nodi a partecipare onestamente.

Nelle classiche formulazioni del consenso distribuito, la caratteristica di tolleranza ai guasti è espressa in termini di numero di nodi non corretti che si possono tollerare. Nel caso del consenso di Nakamoto è caratterizzata in termini di percentuale di potenza di hashing avversaria tollerabile:

- Se la rete si sincronizza più velocemente del tasso di proposta di blocco basato su PoW, una maggioranza onesta può garantire il consenso su una parte stabile in continua crescita della blockchain.
- Fintanto che meno del 50% della potenza di hashing totale è controllata in modo malizioso, i blocchi prodotti da miners onesti vengono propagati tempestivamente, la catena principale è della maggioranza onesta che eventualmente supera qualsiasi ramo malizioso.

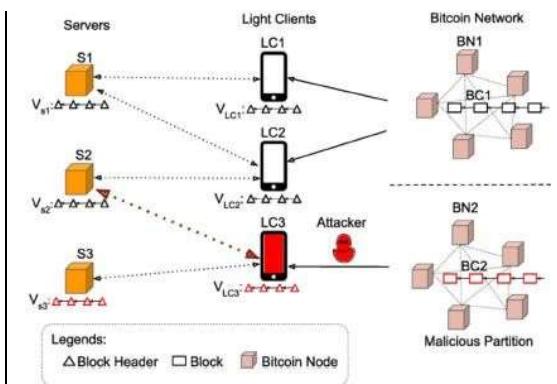
Nel consenso distribuito classico, il consenso di Nakamoto elude abilmente il vincolo fondamentale di **BFT** pari a 1/3 adottando finalità probabilistiche. Nel consenso del BFT classico se più di 1/3 della popolazione è maliziosa, i nodi onesti finiranno per decidere valori contrastanti, portando al fallimento del consenso. Nel consenso di Nakamoto, tuttavia, le decisioni contrastanti sono consentite temporaneamente sotto forma di fork della blockchain, a condizione che alla fine verranno eliminate dal continuo sforzo della maggioranza onesta.

Il consenso di Nakamoto soffre di alcune limitazioni, primo tra tutti un basso throughput delle transazioni. Si può dimostrare che l'intervallo di 10 minuti tra la generazione di blocchi garantisce che ogni nuovo blocco sia sufficientemente propagato prima che venga inserito nel sistema un nuovo blocco.

Ridurre l'intervallo tra i blocchi aumenta il throughput delle transazioni, ma lascia i nuovi blocchi non sufficientemente propagati e provoca più incidenti di fork, minando la sicurezza della catena principale. Aumentare la dimensione del blocco (attualmente 1 MB) ha lo stesso effetto, poiché dimensioni maggiori portano a ritardi di trasmissione più elevati e una propagazione insufficiente.

Inoltre, il meccanismo PoW di Nakamoto causa enorme consumo di energia, una transazione Bitcoin in media consuma 431 KWh di elettricità. Esistono possibili **attacchi al consenso di Nakamoto**:

- **Attacco Eclisse:** Se un potente attacco riesce a dominare la comunicazione in entrata/uscita tra un miner vittima e la rete principale, allora la vittima non sarà più in grado di contribuire all'estensione della catena principale. Se il potere di hashing dell'attaccante è  $\alpha$ , allora un attacco double-spending è possibile se  $\alpha + \epsilon > 50\%$ , con  $\epsilon$  percentuali di miners eclissati (limitando quindi il potere degli altri). L'attacco Eclisse è un exploit della debole connettività di una rete peer-to-peer senza autorizzazione basata su Internet. Per risolvere bisogna aumentare la connettività e la diversità geografica delle connessioni peer-to-peer.



- **Selfish Mining:** Se un gruppo di miners maligni trattiene i blocchi appena estratti e li pubblicizza strategicamente per interrompere la propagazione dei blocchi estratti da miner onesti, può parzialmente annullare il lavoro di miner onesti e amplificare il loro potere di mining.

- (1) I blocchi non sono pubblicizzati ma tenuti segreti tra i miners selfish.
- (2) Gli altri miners estendono la catena con blocchi validi. Il miner egoista continua a estendere il suo ramo segreto fino a quando la catena pubblica è un passo indietro. Quindi la pubblica. Poiché la catena segreta è più lunga, le altre parti la considerano la catena principale, quindi ora tutti stanno seguendo i blocchi del minatore egoista. I blocchi generati dagli altri minatori vengono così eliminate.
- (3) Quando forma per la prima volta la sua catena segreta, il minatore egoista corre un rischio. Se ha generato il primo blocco segreto e poi un altro miner ha generato un blocco, non può pubblicare il suo blocco segreto e avere la catena più lunga, si ha una corsa tra due rami di lunghezza uno.

Il miner egoista cercherà di estendere il proprio ramo come tutti gli altri miner. Se vince, pubblica la catena più lunga, e l'attacco riparte. Se vincono gli altri, il miner egoista è in svantaggio.

A prima vista potrebbe sembrare che l'attacco non funzioni: il miner di minoranza perderà più gare che vince. Tuttavia, un'attenta analisi mostra che non è così in generale. Un insieme di miner selfish più grande di  $1/3$  della potenza di mining aumenterebbe le sue entrate deviando dal protocollo prescritto ed eseguendo Selfish Mining.

#### PROOF-OF-STATE:

La **Proof-of-Stake (PoS)** è un'alternativa efficiente dal punto di vista energetico al PoW. Uno Stake o puntata si riferisce alle monete o token posseduti da un partecipante che possono essere investiti nel processo di consenso.

Rispetto a PoW la cui possibilità di proporre un blocco è proporzionale alla sua potenza di calcolo, la possibilità di proporre un blocco per PoS è proporzionale al valore del suo stake.

PoS non si basa sull'hashing dispendioso per generare blocchi, poiché la difficoltà dell'hashing puzzle diminuisce con il valore dello stake del minter, il numero atteso di tentativi di hashing per un minter per risolvere il puzzle può essere significativamente ridotto se il suo valore di stake è alto.

Pertanto, PoS evita la competizione di hashing bruteforce che si verificherebbe se fosse stato usato PoW ottenendo così una significativa riduzione del consumo energetico.

Questo implica l'assenza del premio per chi inserisce il blocco giusto nella blockchain, e del mining, in quanto non vengono create nuove unità di criptovaluta con la creazione di ogni blocco. I validatori sono ricompensati con una commissione per le transazioni validate.

Esiste la **Proof of Stake delegato (DPoS)**, che consente ai nodi che detengono lo stake maggiore di votare per eleggere i verificatori di blocchi. Questo fa sì che i detentori di stake concedano il diritto di creare blocchi ai delegati che sostengono invece di creare blocchi stessi, riducendo così il loro consumo di potenza computazionale a 0.

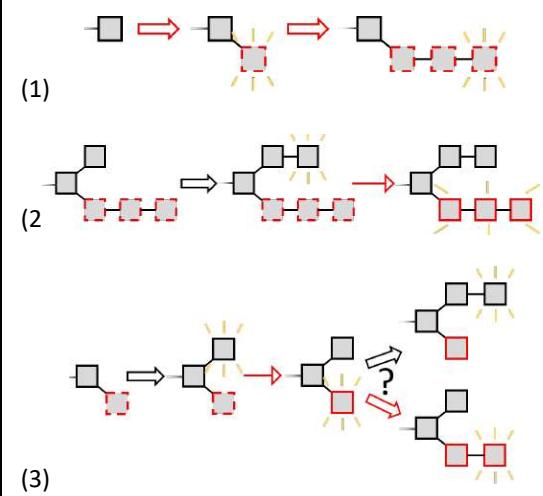
PoW avvantaggia i miners che hanno investito maggiormente in hardware, PoS dà una influenza sproporzionata a coloro che posseggono un numero importante di criptovalute, con il rischio di un accentramento di ricchezza nelle mani di pochi, seguendo il dogma secondo cui «Rich people get richer» ("Le persone ricche diventano più ricche").

Un altro problema è "nothing at stake" per il quale nel caso di una fork del network i validatori saranno incentivati ad operare su entrambe le catene, risultando eventualmente in problemi di double-spending, questo problema è meno evidente in un sistema di DPoS. PoS e DPoS sono stati applicati nel contesto di classici algoritmi BFT, come PBFT, e di consentirne l'applicazione in contesti open e permissionless.

#### TENDERMINT:

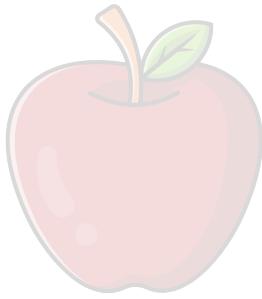
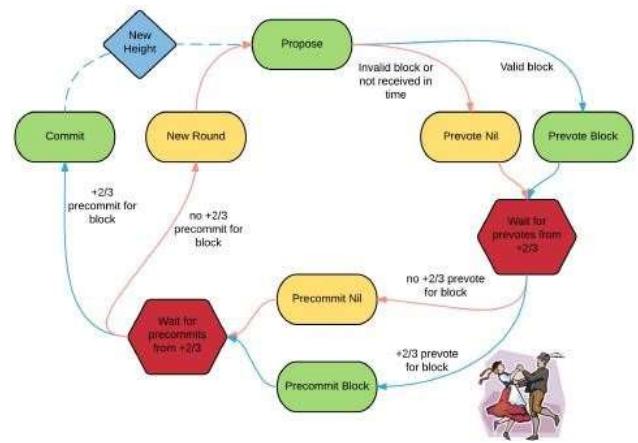
È un algoritmo di consenso ispirato a PBFT che sfrutta la PoS per il consenso in un ambiente permissioned, dove il gruppo dei validatori è un gruppo chiuso e precostituito.

Ogni nuovo blocco viene validato o meno in una iterazione dell'algoritmo, che si compone di molti round per poter giungere a consenso. Si tratta di un consenso di tipo CP, dove la consistenza viene fortemente garantita.



- **[Propose]** Inizialmente, viene scelto a caso un validatore che propone un nuovo blocco.
- **[Prevote Nil] [Prevote Block]** Il nuovo blocco viene distribuito agli altri validatori che ne verificano la validità, e ne votano l'accettazione.
- **[Precommit Nil] [Precommit Block]** Se sono arrivati i 2/3 ed oltre voti allora si manifesta l'intenzione di accettare il blocco, altrimenti di rigettarlo.
- **[Commit] [New Round]** Se sono arrivati i 2/3 dei voti ed oltre allora si conferma la decisione e si passa ad un'altra iterazione con un nuovo validatore, altrimenti si realizza un nuovo round.
- **[Wait for prevotes from +2/3]** È un algoritmo parzialmente sincrono, quindi si smette di aspettare dopo che è trascorso un timeout.

Non tutti i validatori avranno lo stesso "peso", ma la PoS viene usata per dare peso maggiore a chi ha uno stake maggiore. La condizione di 2/3 non è sul numero di votanti ma sulla quantità di criptovaluta totale nel sistema.



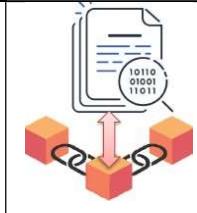
# CoScienze

Associazione

## 13.2. PROGRAMMAZIONE SMART CONTRACTS

Un contratto è un accordo legalmente vincolante che riconosce e disciplina i diritti e i doveri delle parti del contratto. Uno **smart contract** è la "trasposizione" in codice di un contratto, mediante funzioni "if/then" incorporate in software o protocolli informatici, per verificare in automatico l'avverarsi di determinate condizioni e di autoeseguire azioni quando le condizioni sono raggiunte e verificate.

È un programma deterministico che elabora le informazioni in una blockchain, a parità di input i risultati restituiti saranno identici (garantendo la consistenza). Ciò garantisce alle parti una assoluta "certezza di giudizio oggettivo" escludendo qualsiasi forma di interpretazione.



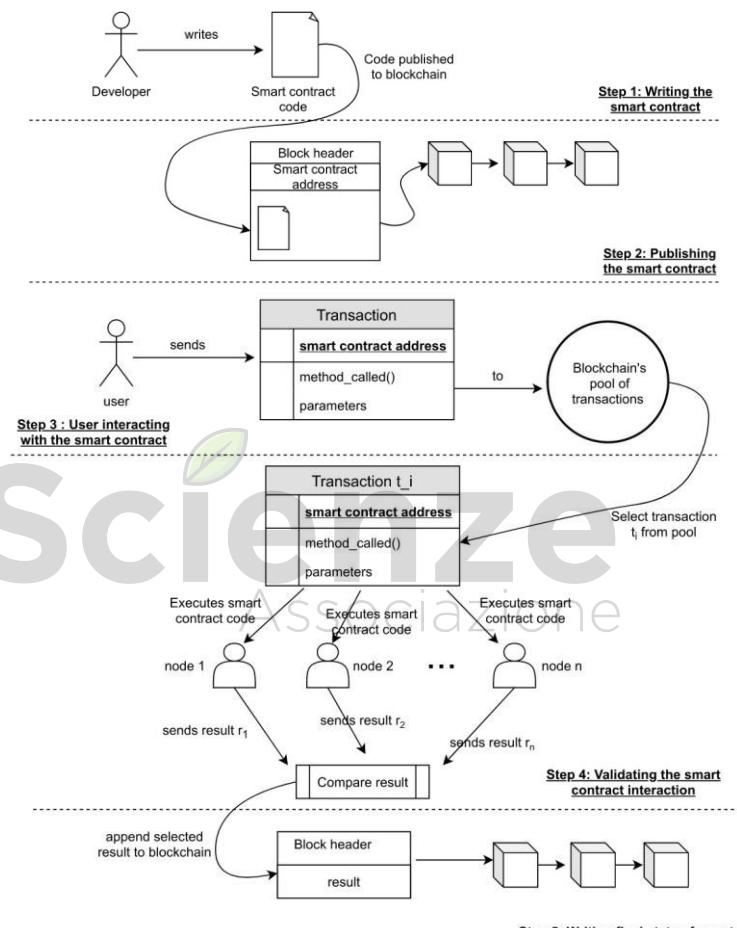
Uno smart contract per blockchain deve soddisfare i seguenti obiettivi:

- **Osservabilità**, l'output deve essere legato sempre all'input, ovvero non bisogna avere dati nascosti e che i risultati vengano tutti restituiti come output (nessuna variabile globale o statica);
- **Verificabilità**, si deve avere la possibilità di testare il software;
- **Riservatezza**, non si devono esporre dati riservati;
- **Applicabilità**, uno smart contract deve essere progettato in maniera tale che è applicabile alla blockchain.

Su una blockchain, uno smart contract non può essere modificato, in quanto è visto come un blocco della blockchain e quindi non mutabile, ma può essere facilmente osservato, verificato, auto-applicato e, a seconda della modalità di accesso della blockchain, è possibile ottenere la privacy.

Ecco un processo graduale di vita di uno smart contract:

1. Gli sviluppatori scrivono la logica per lo smart contract in un linguaggio di programmazione supportato dalla piattaforma blockchain che desiderano utilizzare. Utilizzando un compilatore specifico (di solito fornito dalla piattaforma blockchain), compilano il codice sorgente del loro smart contract e ottengono una sua rappresentazione in bytecode;
2. Dopo aver ottenuto il codice byte, lo smart contract è pubblicato sulla piattaforma blockchain ed archiviato. Una volta pubblicato lo smart contract, sarà di sola lettura o modificabile. Nel caso in cui sia di sola lettura, per fornire un aggiornamento, gli sviluppatori dovranno pubblicare una nuova versione dello smart contract e reindirizzare gli utenti ad essa. Una volta caricato, lo smart contract è al suo stato iniziale, pari ai valori iniziali delle variabili interne;
3. L'accesso a un programma di smart contract pubblicato dipende dalla piattaforma blockchain, come un indirizzo restituito quando lo smart contract è caricato nella piattaforma. Tale indirizzo può essere utilizzato per interagire con lo smart contract, inviando le transazioni contenenti la funzione che desiderano utilizzare e gli argomenti della funzione. Se è necessaria una quantità di valuta della piattaforma per avviare l'esecuzione della funzione, tale importo sarà trasferito insieme alla transazione. La transazione verrà archiviata nel pool di transazioni della piattaforma blockchain che attendono di essere eseguite e convalidate;
4. La piattaforma blockchain selezionerà le transazioni da eseguire e convalidare. Durante l'esecuzione, le funzioni nella transazione verranno eseguite dai nodi. Durante la validazione, i nodi confronteranno i propri risultati e selezioneranno quello da mantenere secondo un protocollo di consenso;
5. Una volta selezionato il risultato valido, verrà inserito in un blocco da aggiungere alla blockchain. Se una transazione validata ha alterato le variabili interne di uno smart contract, i nuovi valori saranno considerati come valori iniziali da transazioni future sullo smart contract.



L'implementazione di smart contract non è standardizzata e ogni piattaforma blockchain propone una propria soluzione.

Le funzioni di uno smart contract possono essere chiamate direttamente dai client o indirettamente da altri smart contracts. Per garantire che le chiamate terminino, al client viene addebitata una fee ad ogni chiamata e sua durata. Se l'addebito supera quello che il cliente è disposto a pagare, il calcolo viene interrotto e le operazioni annullate.

L'attuale approccio di sviluppo di smart contract limita il throughput perché non ammettono concorrenza, infatti:

- Quando un miner crea un blocco, assembla una sequenza di transazioni e calcola un nuovo stato provvisorio eseguendo gli smart contract di tali transazioni in serie, nell'ordine in cui si verificano nel blocco.
- Un miner non può eseguire gli smart contracts in parallelo, perché potrebbero sussistere degli accessi in conflitto a dati condivisi e un interleaving arbitrario potrebbe produrre uno stato finale non coerente. Spesso per gli smart contract, non è possibile dire in anticipo se le esecuzioni di contratti siano in conflitto.

### 13.3. PRINCIPALI PIATTAFORME

La blockchain della criptovaluta **BitCoin** contiene blocchi di transazioni codificate secondo il modello UTXO (Unspent Transaction Output).

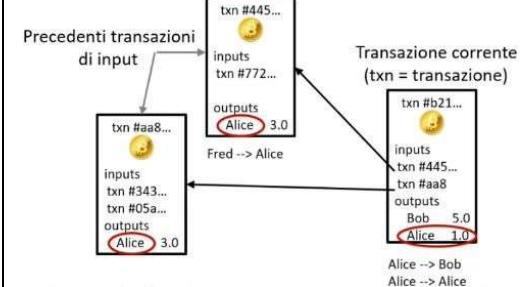


Ogni transazione ha dei Bitcoin in ingresso e in uscita. In ingresso, si ha l'indirizzo delle transazioni mai usate in precedenti transazioni e la cui somma degli output equivale l'output della transazione che le contiene. I destinatari di una transazione sono identificati da un **indirizzo Bitcoin**, un identificatore di 26-35 caratteri. Gli indirizzi possono essere generati gratuitamente da qualsiasi utente.

Spesso è difficile avere transazioni che sommate risultano esattamente pari alla quantità da trasferire. L'eccedenza deve essere trasferita all'autore della transazione.

Ogni transazione contenuta nel wallet ha dei Bitcoin in input e in output. In input si ha l'indirizzo delle transazioni mai usate in precedenza e la cui somma degli output equivale all'output della transazione che lo contiene.

Ad esempio, se si hanno due transazioni aventi un identificativo (Alice) ed un output (3.0 o 2.0), se Alice deve inviare soldi a Bob gli invia transazioni che hanno come destinatario Alice (ovvero le precedenti transazioni in input) e la cui somma è maggiore o uguale ai soldi richiesti. Se rimane della crypto-valuta, nel caso che la somma da inviare è maggiore, Bob riceve la quota richiesta e il rimanente Alice viene segnato come secondo destinatario, ovvero Alice da a sé stessa 1.0.



Il modello UTXO non permette di ritrovare sulla blockchain il saldo corrente di un determinato utente, che deve essere ricostruito raccogliendo tutte le transazioni che presentano un determinato utente in output e quelle con tali transazioni in input. UTXO consente transazioni parallele perché non esiste alcun account, inoltre consente l'anonimato dal momento che un utente può disporre di multipli indirizzi BitCoin. Essendo **stateless** rende complesso la realizzazione di applicazioni come smart contracts, in quanto, ad esempio il saldo di un utente non può essere salvato.

La **natura stateless** della rete consente di ottenere resilienza ed elasticità, oltre che la possibilità per qualsiasi istanza di eseguire qualsiasi attività. Gli utenti hanno molti indirizzi Bitcoin, che fungono da pseudonimi per garantire la privacy, e si vuole impiegare un indirizzo per ogni transazione:

- Un modo ingenuo per accettare pagamenti in Bitcoin è dire ai propri clienti di inviare denaro a un determinato indirizzo (mancanza di identità);
- Essendo le transazioni Bitcoin pubbliche, se un cliente Alice invia Bitcoin, un agente dannoso Bob potrebbe vedere tale transazione e inviare un'email affermando che è stato l'autore del pagamento;

- Il destinatario del pagamento non è in grado di distinguere se il trasferimento è stato effettivamente svolto da Bob o Alice;
- Per questo motivo si usa indicare un indirizzo per ogni cliente che deve effettuare un pagamento.

È possibile impiegare una transazione conoscendo il suo identificativo, che è pubblico. Sorge il problema di evitare che utenti maliziosi impieghino transazioni che non sono proprie, e per formulare dei meccanismi di controllo di autenticazione della legittimità di utilizzo, è stato realizzato un sistema di scripting su BitCoin, precursore degli smart contract. **Script** è un programma imperativo, basato su stack ed elaborato da sinistra a destra:

- Uno script è essenzialmente un elenco di istruzioni registrate con ogni transazione che descrivono come la prossima persona che desidera spendere i Bitcoin trasferiti può accedervi;
- Una transazione è valida se nulla genera un errore e l'elemento in cima allo stack è True (diverso da zero) alla fine dello script;
- Gli stack contengono vettori di byte.

Il linguaggio di scripting ha due tipi di istruzioni:

- Le **istruzioni di dati** contengono semplicemente un valore e sono racchiuse tra parentesi angolari (cioè <data>);
- Gli **OP\_CODE** sono operazioni specifiche appartenenti al linguaggio Bitcoin Scripting che agiscono sul valore in cima allo stack e pone il loro risultato anche in cima allo stack.

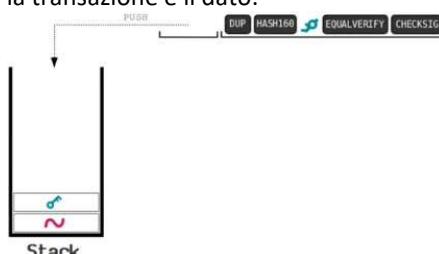
Lo script si compone di due parti, una parte iniziale, che contiene l'ID della transazione (è una chiave)

e la parte finale dove ci sono altre informazioni.

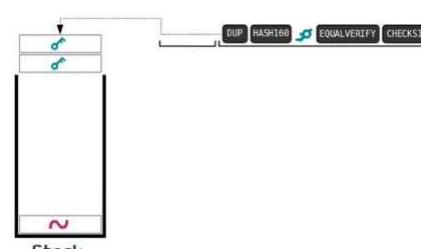
Il funzionamento dello script per la transazione precedente è il seguente:

1. Mette nello stack le prime due parti, ovvero

la transazione e il dato:

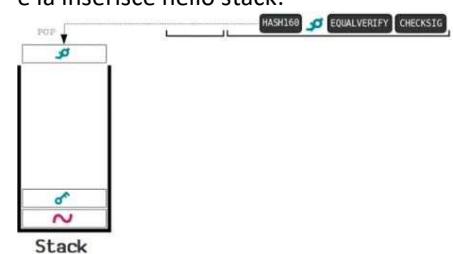


2. Effettua il duplicato (DUP) della chiave:

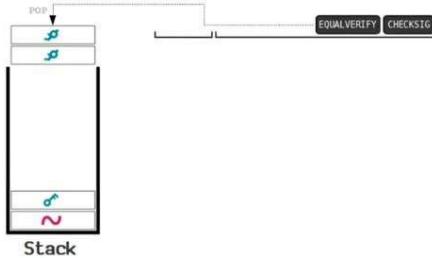


3. Esegue l'HASH160 della chiave che prende

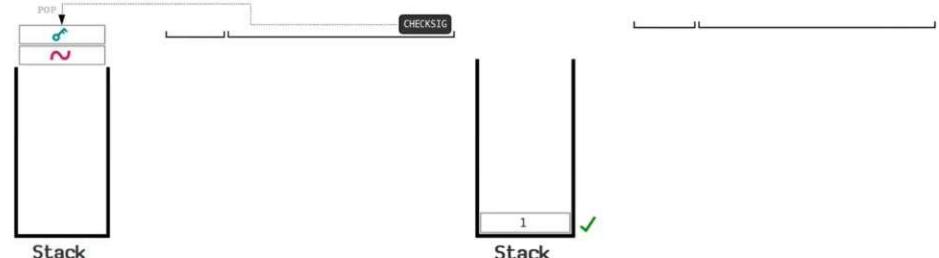
e la inserisce nello stack:



#### 4. Effettua il confronto tra le due chiavi:



#### 5. Esegue il controllo della firma risolvendo così l'accesso:



Altri esempi sono i seguenti:

Per spendere questo output, è necessario fornire due numeri la cui somma è 8.

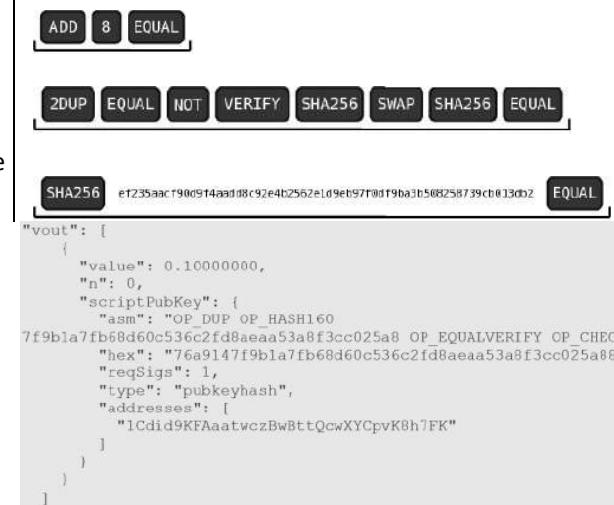
Per spendere questo output, vanno fornite due diverse stringhe di dati che producono lo stesso risultato hash (trovare una collisione).

Per spendere questo output, va dato qualcosa che ha lo stesso hash di ciò che è all'interno dello script di blocco.

Ogni codice operativo ha una corrispondente rappresentazione esadecimale, che viene usata per fornire la sua codifica.

Lo script ha un campo "scriptPubKey" che contiene la parte iniziale dello scripting, ovvero le operazioni da effettuare.

Uno **locking script** viene posizionato su ogni output creato in una transazione per determinare come la transazione deve essere usata. Mentre è necessario fornire uno **unlocking script** per ogni input che si desidera spendere in una transazione.



Quindi nella *transazione precedente* nel campo "out" e in "scriptPubKey" (la parte finale della transazione, ovvero Locking) è presente lo script di unlocking.

Nella *transazione corrente* che intende usare la precedente, nel campo "scriptSig" (la prima parte della transazione, ovvero Unlocking) si mettono i dati su cui lavora lo script che sta in output.

La parte Locking + Unlocking viene eseguito dal validatore per determinare che la transazione può essere utilizzata.

Anche se l'unlocking script è fornito dopo del locking script iniziale, in realtà viene messo per primo quando si eseguono entrambi gli script.

Il linguaggio non è Turing-completo, in quanto non ci sono istruzioni condizionali e cicli, perché si vuole predicitività dei tempi di esecuzione, che dipende deterministicamente dal numero di istruzioni in esse contenute.

Inoltre, è senza stato, non hanno informazione della valuta sbloccata, o non accede alle informazioni del blocco che li contiene.



Transazione precedente

```

txn #3be...
"in": [
    {
        "prev_out": [
            "hash": "...",
            "n": 0
        ],
        "scriptSig": "..."
    },
    ...
],
"out": [
    {
        "prev_out": [
            "value": "10.12287097",
            "scriptPubKey": "OP_DUP..."
        ],
        ...
    }
]

```

Transazione corrente

```

txn #b21...
"in": [
    {
        "prev_out": [
            "hash": "3be...",
            "n": 0
        ],
        "scriptSig": "3044..."
    },
    ...
],
"out": [
    ...
]

```

### 13.3.1. ETHEREUM

**Ethereum** è la più grande piattaforma software decentralizzata ed open-source che consente lo sviluppo di smart contracts e applicazioni implementate al di sopra di una Blockchain permissionless. La blockchain crea un **world state**, ovvero uno stato distribuito tra tutti i nodi. Ethereum è una rete blockchain (con PoW è in transizione verso PoS) ed ha una criptovaluta come BitCoin, chiamata **Ether**. Usato per pagare **gas**, un'unità di calcolo utilizzata nelle transazioni e in altre transizioni di stato. Pertanto, ogni operazione che viene fatta sulla blockchain ha un costo, questo perché l'algoritmo di consenso basato sulla PoW deve dare delle reward, quando si eseguono smart contract la blockchain cresce e il **gas** decresce.

L'algoritmo che viene utilizzato per la PoW è leggermente diverso da quello canonico e viene chiamato **Ethash**, che utilizza un algoritmo di hash appartenente alla famiglia Keccak, la stessa delle funzioni hash SHA-3. Utilizza un set di dati di grandi dimensioni che viene periodicamente rigenerato e cresce lentamente nel tempo. Svolge una serie di intensive letture ad accesso casuale in memoria al set di dati di 2 GB strutturato come un grafo aciclico diretto (DAG). La risoluzione si realizza con 64 cicli a partire dall'hash di informazioni di partenza con pagine estratte dal DAG. Il digest viene confrontato con una soglia target. Se inferiore o uguale, il calcolo è considerato riuscito. In caso contrario, l'algoritmo viene rieseguito con un nonce diverso (incrementando quello corrente o scegliendone uno nuovo a caso).

A differenza di Bitcoin dove si aveva una anonimicità e le transazioni erano da un indirizzo all'altro, in Ethereum si ha il concetto di **accounts**, che sono delle identità sulla blockchain dei suoi utilizzatori che interagiscono tra loro tramite transazioni (messaggi) e sono caratterizzati da un indirizzo a 20 byte che li identifica e viene memorizzato come stato all'interno della blockchain.

Ethereum gestisce due diversi tipi di account:

- **Externally owned accounts (EOA)** (*utenti umani*), sono in grado di inviare e ricevere Ether, e inviare transazioni agli Smart Contract;
- **Contract accounts** (*Smart Contract veri e propri*), che oltre alle funzionalità degli EOAs hanno del codice associato che implementano delle azioni "triggerate" da EOA o altri Contract accounts.

L'esecuzione del codice modifica le informazioni contenute nel proprio spazio di archiviazione, consentendo di utilizzare la Blockchain per scopi diversi dalla criptovalute.



- **Address:** Gli indirizzi Ethereum contengono 40 cifre esadecimali. Gli indirizzi EOA incominciano con il prefisso "0x" seguito dai 20 byte più a destra dell'hash della chiave pubblica. Gli indirizzi di contratto sono nello stesso formato, ma sono determinati dal mittente e dal nonce pari al numero di transazione inviato dal mittente.
- **Balance:** Ethereum non si basa sugli output di transazione non spesi (UTXO), ma gli account hanno uno stato che indica il saldo corrente. Lo stato non è memorizzato sulla blockchain, ma in un albero separato di Merkle Patricia. Un wallet di criptovaluta memorizza le chiavi pubbliche e private (un utente può avere più indirizzi), che possono essere utilizzati per ricevere o spendere ether. I wallet sono come applicazioni che consentono di interagire con un account Ethereum, analogamente alle app di e-banking e un conto bancario;
- **Nonce:** Numero di transazioni totali

I contratti hanno generalmente 4 scopi:

1. Gestire un "data store" che contiene informazioni utili (variabili o dati) per altri contratti o per il mondo esterno;
2. Comportarsi come un EOA con politiche di accesso più avanzate (una sorta di filtro che consente l'inoltro di "messaggi" a determinate EOA solo se determinate condizioni vengono soddisfatte);
3. Gestire un contratto o una relazione in corso tra più utenti (un esempio è un contratto che paga automaticamente chi invia una soluzione valida ad alcuni problemi matematici, o dimostra che sta fornendo una risorsa computazionale);
4. Fornire funzionalità ad altri contratti (una sorta di libreria).

Il codice viene interpretato dalla **Ethereum Virtual Machine (EVM)**, è rappresentato in un EVM bytecode.

Gli **account dei contratti** sono diversi:

	Account Personale	Account di Contratto
address	H(pub_key)	H(addr + nonce del creatore)
code	∅	Codice
storage	∅	Dati
balance	Saldo ETH (in Wei)	Saldo ETH (in Wei)
nonce	# transazioni inviate	# transazioni inviate

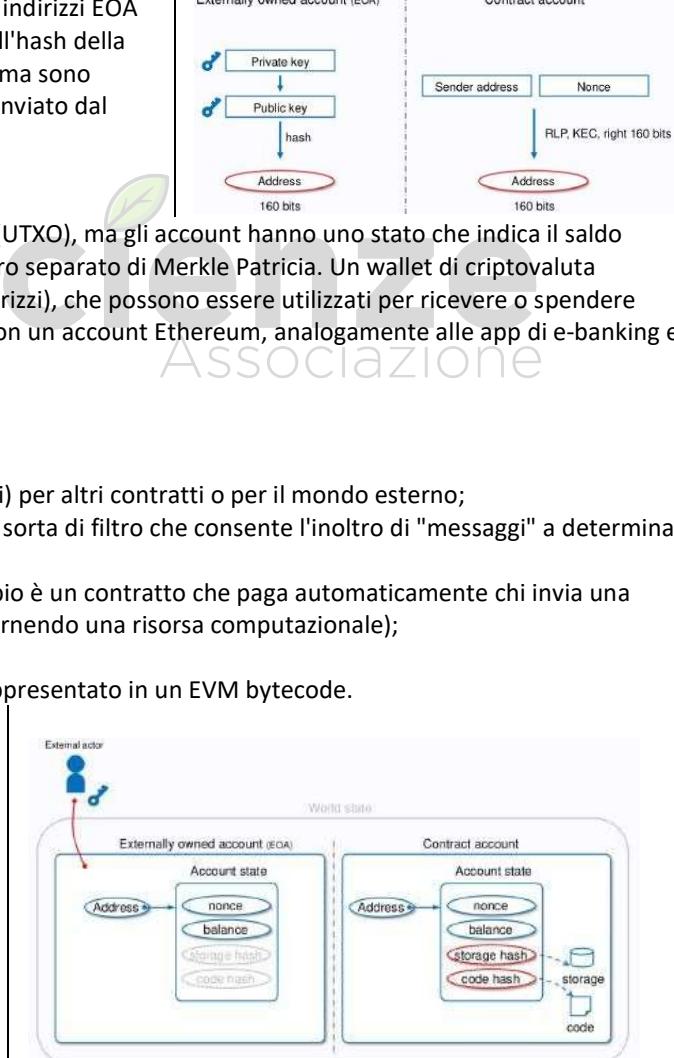
\*H è la funzione hash;

\*Wei è la più piccola unità di ETH, e sono necessari 1018 wei per un ETH. 109 wei compongono un gwei.

Invece, una **transazione** ha la seguente struttura dati:



- **From**, indirizzo dell'utente di origine della transazione (mittente);
- **Signature**, firma della nuova transazione usando la chiave privata dell'utente creatore;
- **To**, indirizzo dell'utente di destinazione della transazione;
- **Amount**, quantità di ETH trasferita.



Terminale Ethereum mediante cui chiamare funzioni della classe principale della libreria web3.js.

- Web3.fromWei converte in Ether un valore espresso in Wei;
- Web3.toWei opera l'inverso;
- La transazione ha richiesto una **commissione** (in gas), per cui il valore associato all'account[0] è inferiore a 90 Ether.

L'avvenuta transazione genera un codice di hash come ricevuta.

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> web3.fromWei(eth.getBalance(eth.accounts[1]))
100
> eth.sendTransaction({
..... from: eth.accounts[0],
..... to: eth.accounts[1],
..... value: web3.toWei(10)
..... })
"0x497913c178f65613035b22340fcf5bc59c7ed474bfa3c1e798c6dffbeda9da5b"
>
> web3.fromWei(eth.getBalance(eth.accounts[0]))
89.99958
> web3.fromWei(eth.getBalance(eth.accounts[1]))
110
```

Nel caso di **transazioni per smart contracts** la struttura cambia per contenere anche i dati per il contratto:



- **Data**, serve per encapsulare i parametri di chiamata della funzionalità dello smart contract e anche a capire qual è la funzionalità invocata.

#### CICLO DI VITA DI SMART CONTRACT IN ETHEREUM:

Il **ciclo di vita degli smart contract** si compone di 3 stati:

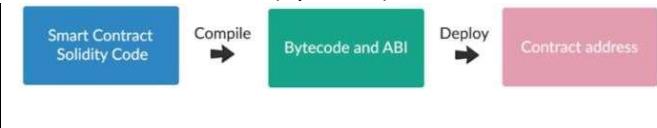


#### -FASE DI CREATE:

Nella fase di **create**, un nuovo contratto viene scritto e caricato sulla rete da parte di un utente. In questa fase, la transazione ha il parametro "to"

vuoto in quanto lo smart contract non esiste ancora, mentre il parametro "data" contiene il codice (Bytecode) del contratto.

Gli smart contracts sono scritti in linguaggi di programmazione di alto livello (Solidity), tale codice viene compilato dalla EVM e deployato nella Blockchain sottoforma di EVM Bytecode.



Successivamente, una volta che il contratto viene scritto viene compilato in un Bytecode (contenuto nel campo "data"):

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

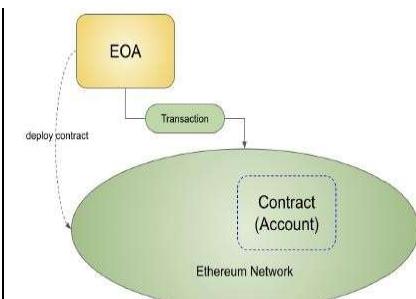
    function set(uint x) public {
        storedData = x;
    }

    function get() constant public returns (uint retVal) {
        return storedData;
    }
}
```



```
"608060405234801561001057600080fd5b5060df8061001f6000396000f30060806
04052600436106049576000357c010000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
078575b600080fd5b348015605957600080fd5b50607660048036038101908080359
060200190929190505060a0565b005348015608357600080fd5b50608a60aa565
b6040518082815260200191505060405180910390f35b8060008190555050565b600
080549050905600a165627a7a7230582080122bb351e6e2c021f1c56c0c5933087e7
62ea6e7a3360b902b39cb5a38f10029"
```

Quando viene effettuata la transazione di deploy, viene creato un nuovo account di tipo smart contract che avrà un suo indirizzo dato dall'hash dell'indirizzo + nonce di chi ha creato.



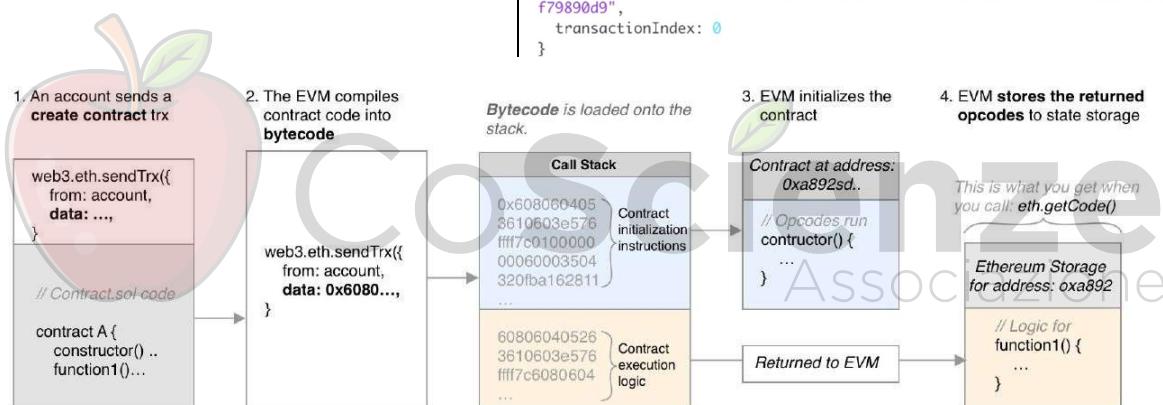
L'EOA crea un contratto con una transazione contenente il relativo bytecode e somministrando una determinata quantità di gas.

L'esecuzione della transazione restituisce un codice di hash.

```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> var bytecode = "608060405234801561001057600080fd5b5060df8061001f6000396000f30060806
0006080604052600436106049576000357c010000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
fd5b348015605957600080fd5b50607660048036038101908080359
a0565b005b348015608357600080fd5b50608a60aa565b60405180910390f35b8060008190555050565b600
80910390f35b8060008190555050565b60008054905090560016!e6e2c021f1c56c0c5933087e762ea6e7a3360b902b39cb5a38f:undefined
>
> eth.sendTransaction({
..... from: eth.accounts[0],
..... data: bytecode,
..... gas: 200000
..... })
"0xc14c38a447fd59ab6ae4df47bd7c15f3125446596675f9ea8;"
```

Eseguendo ***getTransaction(hash\_value)*** si ottengono le informazioni riguardo la transazione eseguita. Il report ottenuto ci informa anche sul blocco creato all'interno della Blockchain.

Eseguendo `getTransactionReceipt(hash_value)` si ottiene una "ricevuta", con l'indirizzo del contratto e l'informazione sul gas speso. In ingresso alla transazione è stato inserito un valore di gas pari a 200000, ma poiché il "gaslimit" è pari a 90000, la parte non spesa viene restituita all'EOA.



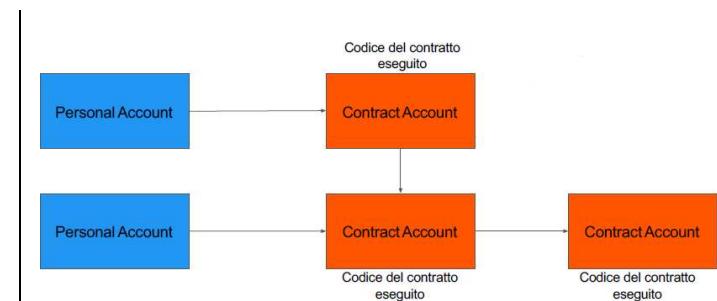
1. Un EOA o un contratto invia una transazione contenente dati, ma nessun indirizzo di destinatario. Questo formato indica all'EVM che è una creazione di contratto, non una normale transazione di invio / chiamata;
  2. EVM compila il codice del contratto in Solidity ottenendo il bytecode, che traduce direttamente in opcode il codice del contratto, che vengono eseguiti in un singolo stack di chiamate;
  3. Durante la creazione del contratto, EVM esegue solo il codice di inizializzazione fino a quando non raggiunge la prima istruzione STOP o RETURN nello stack, il contenuto oltre tale punto è ritornato alla EVM e rappresenta le funzioni che si possono invocare. Durante questa fase, viene assegnato un indirizzo al contratto;
  4. I codici operativi delle altre funzioni vengono copiati in memoria.

#### -FASE DI INTERACT:

Nella fase di ***interact***, un contratto esistente e viene attivato, esegue il suo codice, legge/scrive nella memoria interna ed invia transazioni o chiama altri contratti. Un contratto non può avviare nuove transazioni se non in risposta a delle transazioni ricevute.

In questa fase, nella transazione il parametro “to” contiene l’indirizzo del contratto da invocare, mentre il parametro “data” contiene il metodo del contratto da chiamare ed eventuali argomenti.

Possono sussistere delle relazioni tra contratti, con transazioni inviati tra contratti come effetto dell'esecuzione di codice.



Per poter accedere alle funzioni implementate nel precedente contratto è necessario ottenere la codifica che le è stata assegnata nel bytecode.

Tale codifica è realizzata applicando una funzione di hash alla firma del metodo. Per ottenere tali valori è necessario considerare i primi 4 bytes dei valori ottenuti dall'hash delle funzioni:

```
> web3.sha3('get()')
```

```
"0x6d4ce63caa65600744ac797760560da39ebd16e8240936b51f53368ef9e0e01f"
> web3.sha3('set(uint256)').
```

Effettuando una call si verifica che il valore contenuto nella variabile storedData è 0.

Viene caricato un nuovo valore tramite la sendTransaction.

Bieffettuando la call si ottiene il valore aggiornato.

Le call sono transazioni eseguite direttamente dalla VM e non riportate nella Blockchain.

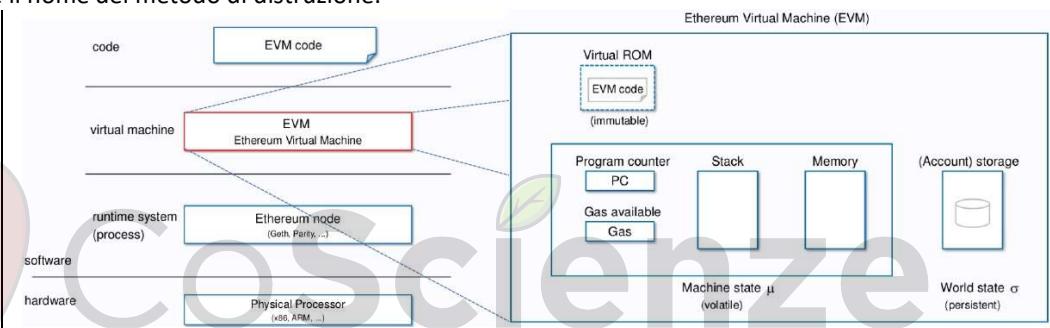
## Bytecode del contratto

### **-FASE DI DESTROY:**

L'ultima fase è la ***destroy***, dove un contratto esistente viene rimosso (in realtà viene invalidato dato che le informazioni in un a blockchain non possono essere modificate) dalla rete da parte di un utente. In questa fase, nella transazione il parametro "to" contiene l'indirizzo del contratto da invocare.

mentre il parametro “data” contiene il nome del metodo di distruzione.

La EVM è un ambiente di esecuzione isolata (sandbox) dei contratti con uno stack, un parallelismo interno a 32- byte, e codici operativi che formano una macchina quasi Turing-completa.



## SOLIDITY (ETHEREUM):

**Solidity** è un linguaggio di programmazione con tipizzazione statica ed orientato agli oggetti per la scrittura di applicazioni che implementano la logica di business incorporata in smart contract su Ethereum e altre piattaforme blockchain. È progettato attorno alla sintassi ECMAScript ma si differenzia per una tipizzazione statica e tipi di ritorno variadici. Supporta inoltre tipi complessi definiti dall'utente, ad esempio struct ed enumerazioni, che consentono di raggruppare i tipi di dati correlati. I contratti supportano l'ereditarietà, inclusa l'ereditarietà multipla con linearizzazione C3. È stata inoltre introdotta un'interfaccia binaria dell'applicazione (ABI) che facilita più funzioni type-safe all'interno di un singolo contratto.

Per **tipizzazione statica** si intende che il controllo del tipo delle strutture dati membro di contratti avviene in fase di compilazione, non in fase di esecuzione come per i linguaggi tipizzati in modo dinamico.

**ECMAScript** è la specifica tecnica di un linguaggio di scripting, destinato a garantire l'interoperabilità delle pagine Web tra diversi browser Web. L'implementazione più conosciuta di questo linguaggio è JavaScript.

Un **parametro di ritorno variadico** accetta zero o più valori di un tipo specificato, e la funzione restituisce una quantità variabile di valori. La **linearizzazione C3** è un algoritmo utilizzato per ottenere l'ordine in cui i metodi devono essere ereditati in presenza di ereditarietà multipla. La linearizzazione è la somma della classe più un merge delle linearizzazioni dei suoi genitori e un elenco dei genitori stessi. Il merge viene eseguito selezionando la prima intestazione delle liste che non appare nella coda di nessun'altra. L'elemento selezionato viene rimosso e aggiunto all'elenco di output. Se non è possibile selezionare un'intestazione valida, allora l'unione è impossibile da calcolare a causa di ordinamenti incoerenti delle dipendenze e nessuna linearizzazione è possibile.

Un **ABI** è un'interfaccia tra due moduli di programma binario, e definisce come si accede alle strutture dati o alle routine nel codice macchina. È espresso in un formato di basso livello, dipendente dall'hardware. Al contrario, un'API definisce questo accesso nel codice sorgente, in formato di livello relativamente alto, indipendente dall'hardware.

È possibile scrivere applicazioni in Solidity mediante l'IDE Remix.

Uno smart contract scritto in Solidity inizia sempre con “*pragma*” che specifica la versione di Solidity. Il contratto inizia con la parola chiave “*contract*” più il nome del contratto (proprio come una classe), per poi definire al suo interno la logica del contratto. In Solidity i qualificatori di accesso vanno messi dopo il nome della funzione.

Quando una funzione deve ritornare qualcosa, nel nome della chiamata va specificato cosa ritorna col tipo di dato. La variabile “amount” è una variabile interna al contratto, per tanto non va nella blockchain. La parola chiave “view” significa che la funzione non modificherà lo stato del contratto, rende visibile il dato e quindi ha un ritorno.

```
testContract.sol x
1 pragma solidity ^0.5.1;
2
3 contract testContract {
4
5     uint amount=13;
6
7     function set(uint _n) public {
8         amount = _n;
9     }
10
11    function get() view public returns (uint)
12        return amount;
13 }
```

Per deploys lo smart contract:

The diagram illustrates the workflow for deploying a Solidity contract:

- Solidity Compiler:** Set up compiler version (0.5.11+commit.c082d0b), language (Solidity), and EVM version (compiler default). Click "Compile testContract.sol".
- Deploy & Run Transactions - Step 1:** Set environment to JavaScript VM, account to 0xCA3...a733c (100 eth), gas limit to 3000000, and value to 0 wei. Click "Deploy".
- Deploy & Run Transactions - Step 2:** Click "At Address" to load the deployed contract at address 0xbBF...732dB.
- Final View:** The deployed contract is shown with methods "set" and "get".

Secondo esempio di smart contract:

In questo contratto è presente la variabile di tipo *"address payable issuer"*, che memorizza un indirizzo che può ricevere delle transazioni di trasferimento di criptovaluta.

Successivamente è presente un costruttore, che in fase di creazione inizializza lo smart contract e la parte funzionale viene memorizzata.

In tal caso prende l'indirizzo di chi ha inviato la transazione di creazione, memorizzato in `"msg.sender"`. Solidity creerà una variabile di sistema che è il messaggio di transazione.

Nella funzione `getBalance()` è presente “`address`” che è una funzione di sistema che conosce l’indirizzo di qualcosa. Quindi, “`address(this)`” darà l’indirizzo dello smart contract, col `.balance` restituisce il saldo dello smart contract.

La funzione `withdrawFunds()` conferma chi ha creato lo smart contract e chi ha invocato la funzione, se corrispondono al possessore dello smart contract viene fatto il trasferimento di criptovaluta con `"issuer.transfer(funds)"` (ricordando che `issuer` è di tipo `payable`).

La funzione `receiveFunds()` è anche essa payable, significa che quando si invoca questa funzione, lo smart contract riceve dei fondi, non ha logica perché serve solo a

In totale, questo smart contract funge da salvadanaio, ovvero il creatore invia dei fondi che si accumulano nello smart contract, quando poi sono necessari all'issuer, invoca la funzione withdrawFunds() per prelevare saldo.

### 13.3.2 HYPERLEDGER FABRIC

**Hyperledger** è una famiglia di progetti di blockchain open source avviato nel 2015 dalla Fondazione Linux, per supportare lo sviluppo collaborativo di soluzioni DLT (Distributed Ledger Technology).

(Distributed Ledger Technology). Si suddivide in framework per la realizzazione di piattaforme blockchain, e strumenti (tools) a supporto di applicazioni e smart contracts.



**Hyperledger Fabric** è un progetto di IBM per imprese, e consiste in una piattaforma di DLT **permissioned**, con alcune differenze chiave rispetto ad altri popolari soluzioni:

- ha un'architettura altamente modulare e configurabile;
  - supporta smart contracts (detto *Chaincode*) scritti in linguaggi di programmazione comuni, piuttosto che utilizzare linguaggi specifici del dominio;
  - i partecipanti sono noti, invece che essere anonimi, adottando un modello di governance costruito sulla fiducia che c'è tra i partecipanti;

- supporto per *protocolli di consenso* innestabili, da scegliere sulla base delle esigenze, e senza una criptovaluta nativa per incentivare costosi mining o per alimentare l'esecuzione di contratti intelligenti.

Le **organizzazioni** che prendono parte alla costruzione della rete Hyperledger Fabric sono chiamate «**membri**», ognuna responsabile di impostare i propri peer per la partecipazione alla rete. Questi **peer** devono essere configurati con materiali crittografici appropriati per autenticare la propria *identità* o proteggere i canali di comunicazione (con SSL/TLS).

- I peer ricevono richieste di transazioni dai client mediante un SDK o servizi Web REST per interagire con la rete Hyperledger Fabric, e sono connessi tra loro da canali che consentono l'isolamento dei dati e la riservatezza;
- Tutti i peer mantengono il loro registro unico per canale a cui sono iscritti, cioè solo se il peer è registrato a quel canale riceve i blocchi pubblicati su quel canale e ogni canale ha un suo registro (ledger). A differenza di Ethereum, i peer hanno ruoli diversi:
  - Endorser peer:** ricevuta la richiesta da un client, convalida la transazione ed (eventualmente) esegue il Chaincode simulando l'esito della transazione ma senza aggiornare la blockchain, cioè solo il nodo Endorser esegue il Chaincode per vedere cosa succederebbe se eseguito, quindi non è necessario installarlo in ogni nodo della rete. Al termine, l'Endorser può approvare o disapprovare la transazione;
  - Anchor peer:** riceve gli aggiornamenti e trasmette gli aggiornamenti agli altri peer nell'organizzazione. Quindi, quando bisogna aggiornare la blockchain viene avvisato prima l'anchor peer di una organizzazione e questo propaga le informazioni agli altri peer. Tutto ciò viene fatto per ridurre i partecipanti all'algoritmo di consenso. È possibile configurare canali segreti tra i peer e le transazioni tra i peer di quel canale sono visibili solo ai partecipanti;
  - Orderer peer:** rappresenta l'elemento centrale di un canale tra peer ed è responsabile dello stato del registro coerente in tutta la rete ordinando le transazioni e collocandole in nuovi blocchi. Sono attualmente disponibili due impostazioni:
    - Solo:* un singolo orderer da usare per lo sviluppo, non in produzione, perché poco scalabile e resiliente.
    - Kafka:* soluzione di streaming processing di Apache che garantisce la ricezione di messaggi nell'ordine di invio.

Si possono avere due tipi di **transazioni**:

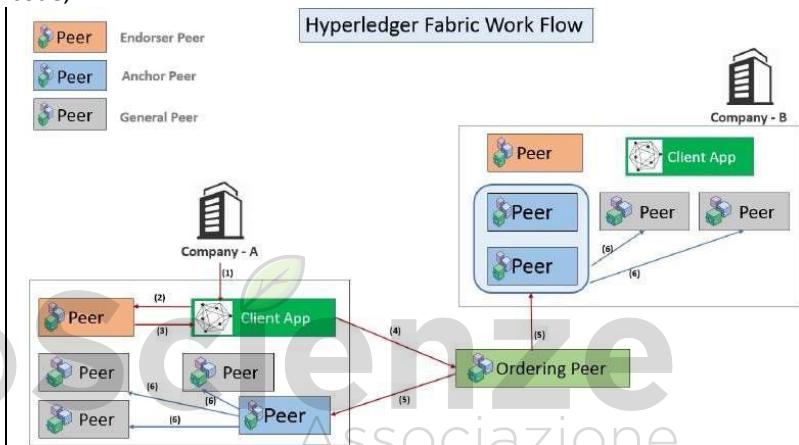
- Deploy transactions** per creare un nuovo chaincode ed installarlo sulla Blockchain (solo sull'Endorser peer);
- Invoke transactions** per richiamare la funzioni di un chaincode;

#### Esempio:

Una infrastruttura su presenta nel modo seguente:

Ci sono (in questo caso) due organizzazioni (Company-A / -B) e un Ordering Peer, in tal caso è centralizzato, se fosse stato con Kafka gli ordering peer erano di più.

Ogni organizzazione ha un Endorser peer che interagisce col client, un Anchor peer che comunica direttamente con l'ordering peer, ed infine i general peer che mantengono la copia del registro, questi non sono direttamente collegati all'ordering peer ma ricevono le informazioni tramite Anchor peer.



Immaginiamo che ogni organizzazione ha una Client App, applicazione con cui dialoga l'utente con la blockchain per mezzo di un SDK:

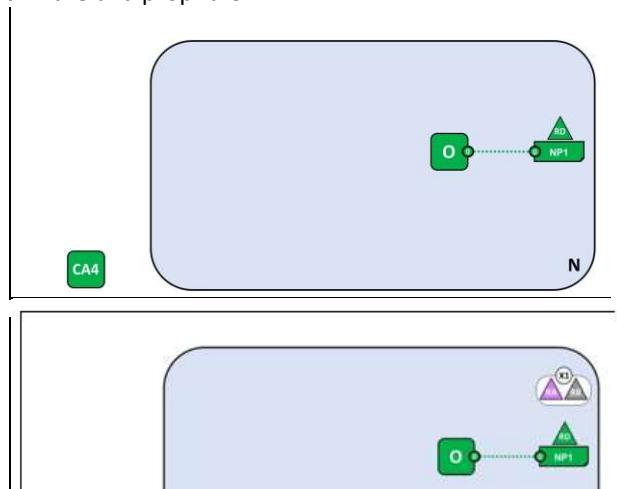
- Un partecipante nell'organizzazione membro effettua una richiesta di transazione tramite l'applicazione client;
- L'applicazione client trasmette la richiesta al Endorser peer, che controlla i dettagli del certificato e altri dettagli per convalidare la transazione;
- Eventualmente esegue il Chaincode e restituisce le risposte di Endorsement, accettando o rifiutando la transazione;
- Il client invia la transazione approvata all'Orderer peer, affinché questa venga ordinata correttamente rispetto ad altre transazioni;
- Il nodo Orderer include la transazione in un blocco, e inoltra il blocco ai nodi Anchor di diverse Organizzazioni membri della rete, che eseguono un consenso distribuito PBFT;
- Al raggiungimento del consenso, gli Anchor peer trasmettono il blocco agli altri peer all'interno della propria organizzazione, così che questi aggiornano il proprio registro locale con l'ultimo blocco. Così tutta la rete ottiene il registro sincronizzato.

#### CERTIFICATE AUTHORITY (CA):

Una **Certificate Authority (CA)** emette i certificati per consentire alle organizzazioni di autenticarsi sulla rete, alle applicazioni client di autenticare le proposte di transazione e ai peer di approvare le proposte e caricare le transazioni nel libro mastro se valide.

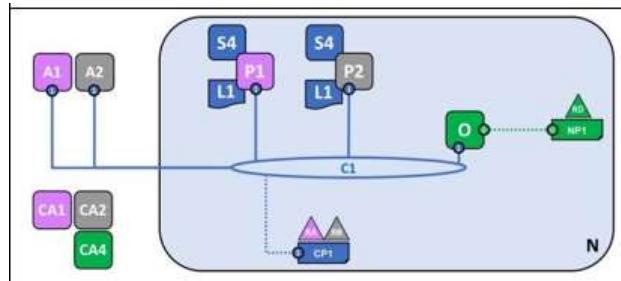
Possono esserci una o più CA sulla rete e le organizzazioni possono scegliere di utilizzare una propria CA.

La rete viene creata dalla definizione del servizio di ordinazione (O) con la configurazione per i canali all'interno della rete (NP1), includendo le politiche di accesso e le informazioni sull'appartenenza (come certificati radice X509) per ogni membro del canale (CA4).



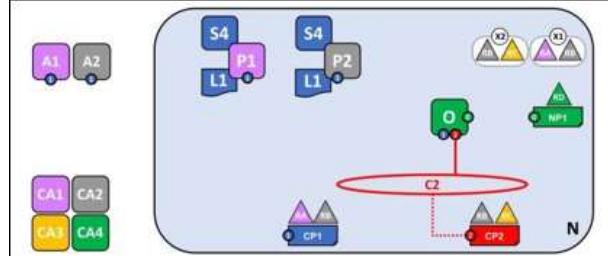
Il consorzio (X1) alla base della rete viene istaurato definendo i certificati delle organizzazioni membro (RD, RA e RB) e le relative CA.

Non vi siano limiti teorici alla dimensione di una rete, e il protocollo gossip è usato per accogliere un gran numero di nodi peer sulla rete e scambiare informazioni sulla rete. Sul canale sono collegate anche le applicazioni client (A1 e A2).



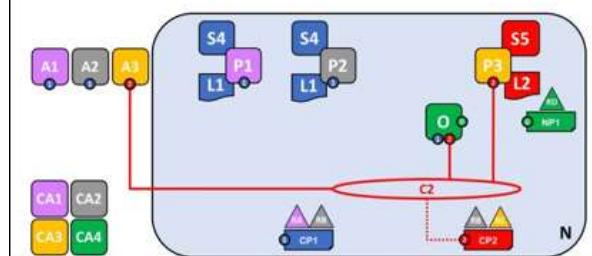
È possibile definire un altro consorzio, dove una terza organizzazione (RC) interagisce con la seconda RB.

Un secondo canale (C2) nell'ambito del secondo consorzio è definibile, con proprie politiche di accesso.



Un peer (P3) per la terza organizzazione può essere istanziato e collegato al secondo canale, dispiegando il chaincode dello smart contract S5.

Nota che le informazioni del canale 1 e le informazioni del canale 2 sono indipendenti.

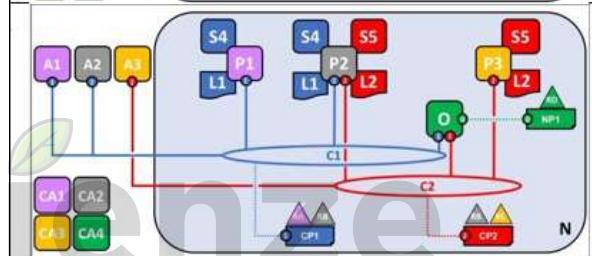


Ora l'organizzazione RB si trova sia in C1 che in C2, il suo peer P2 ha sia il ledger L1 che L2, questo è il meccanismo di isolamento delle porzioni della blockchain.



# CoScienze

Associazione



Nel complesso l'architettura di Hyperledger è composta da tre macro-aree (o layer):

1. Una per la gestione delle **identità**, dell'appartenenza di peer ad associazioni e l'iscrizione del peer ai canali;
2. Un **libro mastro** con lo stato corrente dei dati e uno storico delle transazioni;
3. L'ambiente di esecuzione dei **chaincode** per gli smart contract.

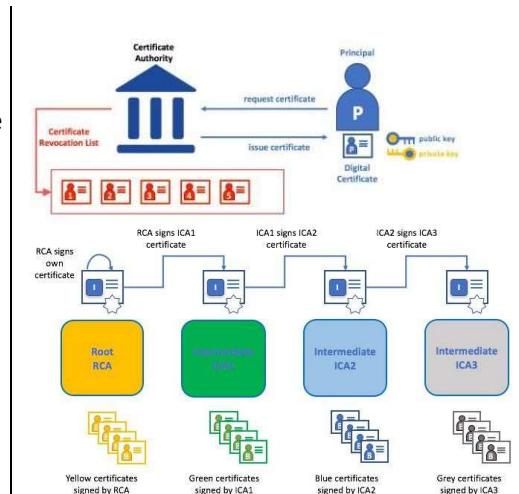
Trasversalmente sono disponibili delle primitive crittografiche a supporto di questi tre strati.

## IDENTITÀ DIGITALE:

Ogni elemento ha un'**identità digitale** encapsulata in un certificato digitale X.509, esprimendo anche le autorizzazioni necessarie per le risorse e l'accesso alle funzionalità nella rete blockchain.

Affinché un'identità sia verificabile, deve provenire da un'autorità fidata, come la CA con un modello gerarchico PKI.

Il certificato digitale contiene molte informazioni in merito ad un'identità in SUBJECT, ma anche la chiave pubblica legata all'identità, mentre la sua chiave di firma non lo è e viene mantenuta privata. Il certificato è controfirmato dalla CA che lo emette e lo rende impossibile da modificare.



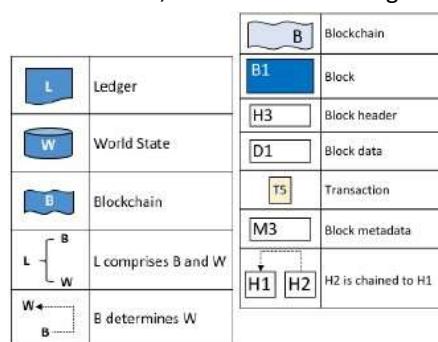
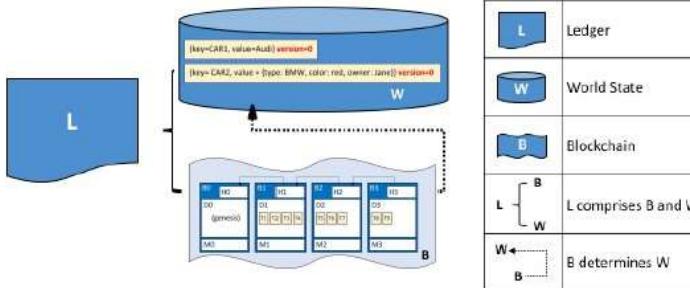
Esiste una gerarchia di CA, con una Root CA come radice e una serie di CA intermediari i cui certificati sono firmati dalle CA di livello superiore. Fabric offre una propria CA radice.

## LEDGER:

Il **libro mastro** blockchain è costituito da due parti distinte:

- Uno **stato globale (world state)** in un database che contiene i valori correnti di un insieme di dati indirizzabili per mezzo di una chiave;
- Uno **storico delle transazioni (ledger)** come blockchain, con tutti i cambiamenti dello stato globale.

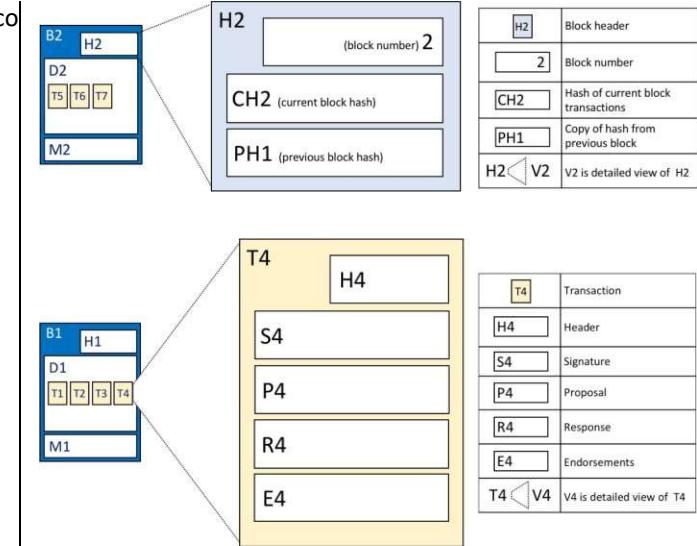
Il numero di versione viene incrementato ogni volta che lo stato cambia, e anche verificato ogni volta che lo stato viene aggiornato.



W	Ledger world state
(key=K, value = V ) version=0	A ledger state with key=K. It contains a set of facts expressed as a simple value, V. The state is at version 0.
(key=K, value = (KV) ) version=0	A ledger state with key=K. It contains a set of facts expressed as a set of key-value pairs (KV). The state is at version 0.

## BLOCCHI E TRANSAZIONI:

Un **blocco** ha un header con un numero di sequenza, l'hash del blocco e del blocco precedente, il corpo con l'insieme ordinato delle transazioni e dei metadati con l'ora di creazione del blocco, il certificato, la chiave pubblica e la firma del creatore del blocco.



Le **transazioni** hanno un header come il nome del chaincode pertinente e la sua versione, la firma del creatore, la proposta e la risposta con l'endorsement.

## CHAINCODE:

**Chaincode** è un programma scritto in Go, node.js o Java che implementa un'apposita interfaccia e viene eseguito in un contenitore Docker isolato dal processo dell'Endorser peer. Lo stato creato da un chaincode è limitato esclusivamente a quel chaincode e non è possibile accedervi direttamente da un altro chaincode. Tuttavia, un chaincode può invocare un altro chaincode per accedere al suo stato.

L'interfaccia Chaincode specifica le funzioni da implementare:

- **Init()** viene chiamato quando un chaincode riceve un'istanza o una transazione di aggiornamento in modo da eseguire qualsiasi inizializzazione necessaria, inclusa l'inizializzazione dello stato dell'applicazione;
- **Invoke()** viene chiamato in risposta alla ricezione di una transazione invoke per elaborare le proposte di transazione.

L'altra interfaccia è ChaincodeStubInterface, che viene utilizzato per accedere e modificare la blockchain e per effettuare invocazioni tra i chaincode. Dato che il supporto Java è stato introdotto di recente, le API Node.js e Go lang sono a un livello più maturo. JavaScript non si presta bene nel caso di calcoli numerici.

## LINGUAGGIO GO:

Il linguaggio più usato per realizzazione chaincode è Go. Questo linguaggio non presenta il concetto di classe, ma Go offre struct, una versione leggera delle classi, a cui è possibile aggiungere metodi.

Realizziamo uno smart contract in Go per la gestione delle auto e dei loro proprietari. Implementiamo le struct per questi dati.

```

type Car struct{
    modelName string
    color string
    serialNo string
    manufacturer string
    owner Owner //composition
}

type Owner struct{
    name string
    nationalIdentity string
    gender string
    address string
}

//attached by reference ==> called as pointer receiver
func (c *Car) changeOwner(newOwner Owner) {
    c.owner = newOwner
}

/** attached by value ==> called as value receiver
func (c Car) changeOwner(newOwner Owner) {
    c.owner = newOwner
}
*/

```

Il metodo è definito esternamente, indicando nel primo caso se le modifiche nel corpo del metodo si riflettono sul chiamante.

L'interfaccia da implementare è la seguente con le due principali funzioni delle 36 attualmente specificate -->

Non sussiste una sintassi di derivazione, ma basta solo implementare i metodi di interesse.

#### Inseriamo la logica di inizializzazione:

```
func (t *CarChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {

//Declare owners from Owner struct
tom := Owner{name: "Tom H", nationalIdentity: "ABCD33457", gender:
"M", address: "1, Tumbbad"}
bob := Owner{name: "Bob M", nationalIdentity: "QWER33457", gender:
"M", address: "2, Tumbbad"}

//Declarea carfrom Car struct
car := Car{modelName: "Land Rover", color: "white", serialNo:
"334712531234", manufacturer: "Tata Motors", owner: tom}

// convert tom Owner to []byte
tomAsJSONBytes, _ := json.Marshal(tom)
//Add Tom to ledger
err := stub.PutState(tom.nationalIdentity, tomAsJSONBytes)
if err != nil {
    return shim.Error("Failed to create asset " + tom.name)
}

//Add Bob to ledger
bobAsJSONBytes, _ := json.Marshal(bob)
err = stub.PutState(bob.nationalIdentity, bobAsJSONBytes)
if err != nil {
    return shim.Error("Failed to create asset " + bob.name)
}

//Add car to ledger
carAsJSONBytes, _ := json.Marshal(car)
err = stub.PutState(car.serialNo, carAsJSONBytes)
if err != nil {
    return shim.Error("Failed to create asset " + car.serialNo)
}

return shim.Success([]byte("Assets created successfully."))
}
```

```
type Chaincode interface {
// Init method accepts stub of type ChaincodeStubInterface as
// argument and returns peer.Response type object
Init(stub ChaincodeStubInterface) peer.Response
}
```

```
type CarChaincode struct{
}

//Init implemented by CarChaincode
func (t *CarChaincode) Init(stub shim.ChaincodeStubInterface)
pb.Response {

}

//Invoke implemented by CarChaincode
func (t *CarChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {

}
```

#### Inseriamo la logica di invoke:

```
func (c *CarChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {

    // Read args from the transaction proposal.
    // fc=> method to invoke
    fc, args := stub.GetFunctionAndParameters()
    if fc == "TransferOwnership" {
        return c.TransferOwnership(stub, args)
    }
    return shim.Error("Called function is'n't defined in the chaincode
")

func (c *CarChaincode) TransferOwnership(stub
shim.ChaincodeStubInterface, args []string) pb.Response {
    // args[0]=> car serial no
    // args[1]=> new owner national identity
    // Read existing car asset
    carAsBytes, _ := stub.GetState(args[0])
    if carAsBytes == nil {
        return shim.Error("car asset not found")
    }

    // Construct the struct Car
    car := Car{}
    _ = json.Unmarshal(carAsBytes, &car)

    //Read newOwnerDetails
    ownerAsBytes, _ := stub.GetState(args[1])
    if ownerAsBytes == nil {
        return shim.Error("owner asset not found")
    }

    // Construct the struct Owner
    newOwner := Owner{}
    _ = json.Unmarshal(ownerAsBytes, &newOwner)

    // Update owner
    car.changeOwner(newOwner)

    carAsJSONBytes, _ := json.Marshal(car)

    // Update car ownership in the
    err := stub.PutState(car.serialNo, carAsJSONBytes)
    if err != nil {
        return shim.Error("Failed to create asset " + car.serialNo)
    }
    return shim.Success([]byte("Asset modified."))
}
```

```
package main

import (
    "encoding/json"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)
func main() {

    logger.SetLevel(shim.LogInfo)

    // Start chaincode process
    err := shim.Start(new(CarChaincode))
    if err != nil {
        logger.Error("Error starting PhantomChaincode - ", err.Error())
    }
}
```

Per completare lo smart contract è necessario specificare il preambolo:

E la funzione principale:

## 13. AGGIUNTA. BLOCKCHAIN APPLICATION IN IOT

**Blockchain** è vista come una tecnologia **disruptive** che avrà un ruolo fondamentale nella gestione, controllo e **sicurezza** dei dispositivi **IoT**.

- **Autenticazione e integrità dei dati:** in contesti dove è richiesta una **tracciabilità completa degli asset** nel loro ciclo di vita, l'**immutabilità dei dati** è una sfida importante. Questo ha spinto l'adozione di **Blockchain** nell'ambito dell'**IoT**;

- Identità dei Dispositivi (IDoT) e Governance:** Blockchain è ampiamente utilizzata per fornire **registrazione delle identità affidabile, monitoraggio della proprietà e tracciamento degli asset**;
- Autenticazione, Autorizzazione e Privacy:** i **contratti intelligenti (smart contracts)** su Blockchain consentono regole di **autenticazione decentralizzata**, sia per l'autenticazione **singola** che **multilaterale** dei dispositivi IoT. Inoltre, i contratti intelligenti semplificano notevolmente le **regole di accesso e autorizzazione**, rispetto ai tradizionali protocolli di autorizzazione;
- Comunicazione Sicura:** con l'uso di Blockchain, la gestione delle **chiavi** e la distribuzione sono eliminate, ogni dispositivo IoT avrà un proprio **GUID unico** e una coppia di **chiavi asimmetriche** una volta installato e connesso alla rete Blockchain. Questo porta a una **semplificazione significativa** dei protocolli di **sicurezza**.

Un altro aspetto da considerare nell'integrazione della **blockchain** con l'**IoT** è che non è possibile eseguire la **blockchain direttamente sui nodi IoT** a causa degli **elevati costi** in termini di **memoria** e **consumo energetico**:

- I dispositivi IoT possono comunicare tra loro**, utilizzando meccanismi di **scoperta** e **intradramento**. Solo una parte dei dati IoT viene memorizzata nella **blockchain**, mentre le interazioni tra i dispositivi IoT avvengono senza l'uso della blockchain.

- Tutte le interazioni avvengono tramite la blockchain**, consentendo una registrazione **immutable** delle interazioni e la possibilità di renderle **tracciabili**. Tuttavia, registrare tutte le interazioni nella **blockchain** comporta un aumento della **larghezza di banda** e dei **tempi di recupero dei dati**, rappresentando una delle **sfide** più conosciute della blockchain.
- Un design ibrido** in cui solo una parte delle interazioni e dei dati avviene tramite la **blockchain**, mentre il resto viene condiviso direttamente tra i dispositivi **IoT**. Una delle sfide di questo approccio è decidere quali interazioni debbano passare tramite la **blockchain** e come stabilire questa decisione in tempo reale. In questo contesto, l'**edge computing** (fog computing) e il **cloud computing** potrebbero integrare le **limitazioni** della **blockchain** e dell'**IoT**.

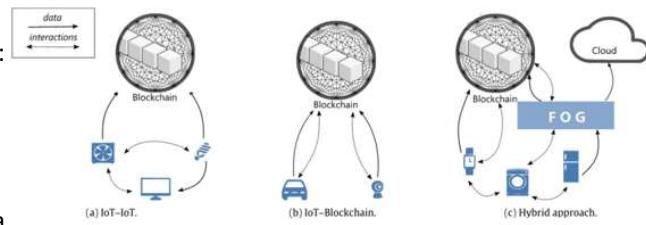
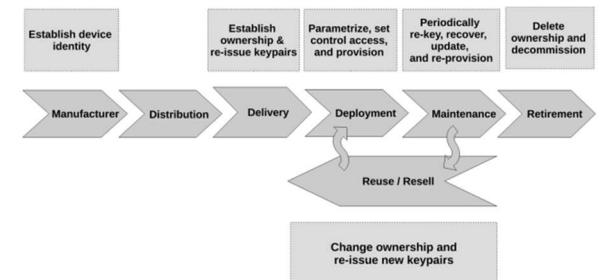
In una tipica **implementazione IoT**, i dispositivi con risorse limitate vengono utilizzati come **nodi finali** che comunicano con un **gateway**, responsabile dell'invio dei dati dei sensori ai livelli superiori (cloud o server). La funzionalità **crittografica** potrebbe essere integrata nei dispositivi **IoT** per garantire **autonomia**, ma ciò comporta l'uso di **hardware più sofisticato** e un **maggior costo computazionale**.

## EXAMPLE – BLOCKCHAIN & IOT:

Una soluzione adatta per una gestione efficiente della **fiducia** nell'**IoT** sarebbe avere una **federazione di Trust Providers** replicati, per renderla adatta a un **IoT multi-tenant** e rimuovere il **collo di bottiglia** e il **punto di guasto unico** rappresentato da un fornitore centralizzato.

Problemi:

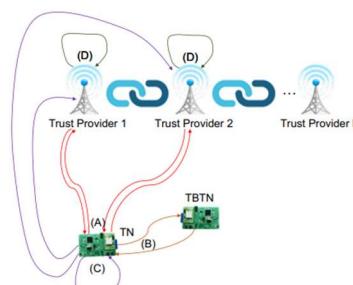
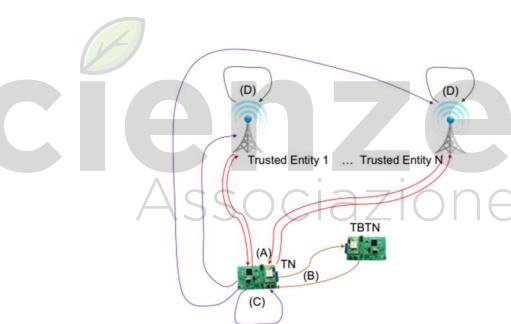
- Proteggere le **comunicazioni** tra i trust providers e il nodo richiedente.
- Evitare la rivelazione delle informazioni ospitate dai trust providers.
- Supportare la coerenza tra i vari trust providers;



Una soluzione proposta è quella di utilizzare la **blockchain** come **repository condiviso** dei valori di fiducia. I trust providers operano a livello **edge/fog** dell'infrastruttura IoT, ma non sui nodi IoT, a causa delle loro limitazioni di **risorse disponibili**.

Questa soluzione può essere facilmente applicata anche nel contesto di **autenticazione/ autorizzazione**, dove il fornitore implementa **XACML**.

- I **nodi fog/edge** e/o quelli nel **cloud** possono scrivere nuovi blocchi e aggiungerli alle catene. Ogni catena contiene i **valori di fiducia** associati a ciascuna entità nel sistema;
- Far sì che questi nodi informino periodicamente i dispositivi **IoT** di eventuali modifiche sarebbe inefficiente. Pertanto, quando è necessario un **valore di fiducia**, i dispositivi IoT contattano i nodi **raggiungibili**;
- Vengono contattati più nodi per evitare che un nodo **malevolo** possa impersonare un nodo legittimo o ricevere informazioni divergenti;
- Se l'interazione diretta con un'entità porta il sensore a possedere un valore di fiducia diverso da quello raccolto dai nodi contattati, viene richiesto un **aggiornamento**, creando un nuovo blocco.



Tutti i dati memorizzati **non sono crittografati**, quindi, nonostante l'uso di canali sicuri, se un avversario conosce l'**identificativo** di un'entità valida nel sistema, potrebbe compromettere il protocollo. Per questo motivo, è necessario un **meccanismo di pseudonimizzazione** per anonimizzare le informazioni di fiducia e rendere più difficile rintracciarle.

## 14. CLOUD SECURITY

### INTRODUZIONE:

Il **cloud computing** comporta la perdita di controllo da parte di fornitori e clienti riguardo ai dati e alle applicazioni, aumentando il rischio percepito di sicurezza. Questo timore è uno dei principali ostacoli all'adozione delle soluzioni cloud.

La ricerca sulla sicurezza del cloud ha fatto progressi significativi, ma esistono ancora soluzioni di sicurezza parziali che affrontano solo piccole parti del problema.

La **garanzia della sicurezza nel cloud** è essenziale per stabilire fiducia, assicurando che le applicazioni e i dati siano protetti nonostante guasti o attacchi.

La letteratura sul tema può essere suddivisa in tre principali aree di studio:

- **Vulnerabilità e minacce:** nuove minacce e attacchi nel cloud;
- **Tecniche di sicurezza:** nuove soluzioni per proteggere dati e applicazioni nel cloud;
- **Tecniche di garanzia:** metodi per verificare, provare e garantire le proprietà di sicurezza fornite dalle tecniche implementate;

Nel cloud possiamo avere:

- Gli **attacchi di tipo SaaS** (Software-as-a-Service) mirano a compromettere i servizi e i dati a livello di applicazione;
- Gli **attacchi su PaaS e IaaS** (Platform-as-a-Service e Infrastructure-as-a-Service) colpiscono risorse, processi e dati condivisi tra diversi clienti della stessa infrastruttura, spesso sfruttando vulnerabilità nella virtualizzazione;

La **trasparenza** è un concetto cruciale, che permette ai fornitori di cloud di mostrare come ottengono e mantengono la conformità a normative e politiche di sicurezza. È importante che i fornitori non solo rispettino le normative, ma anche spieghino come raggiungono tali livelli di conformità, operando secondo il principio "comply-or-explain". La trasparenza aiuta sia nella **introspezione** (analisi interna dei processi) che nell'**outrospezione** (possibilità per i clienti di esaminare i processi del cloud).

Le problematiche di sicurezza del cloud sono complesse a causa di:

- **Eterogeneità delle tecnologie del cloud;**
- **Mancanza di requisiti di sicurezza formalizzati;**
- **Difficoltà nella categorizzazione delle tecniche di sicurezza;**
- **Bilanciamento tra sicurezza, flessibilità e alte prestazioni;**
- **Mancanza di trasparenza nelle attività e negli eventi nel back-end del cloud**

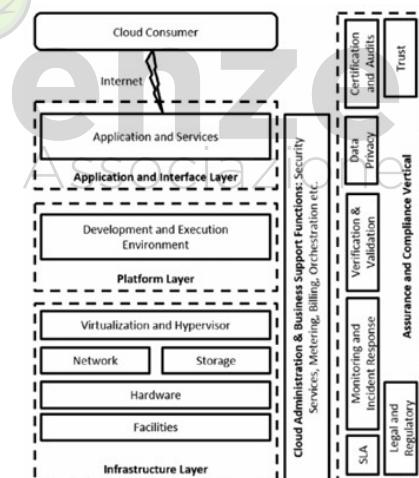
Anche se è possibile avere una buona sicurezza nel cloud, spesso la **garanzia della sicurezza** è scarsa, il che impedisce di dimostrare che i processi rispettano le leggi e le normative.

### MODELLAZIONE DELLA SICUREZZA NEL CLOUD:

La **modellazione della sicurezza** nel cloud riguarda l'identificazione dei requisiti di sicurezza, l'analisi delle minacce derivanti dalle vulnerabilità nei componenti architetturali del cloud e la definizione di contromisure per mitigare o eliminare i vettori di attacco. Questi aspetti sono fondamentali per garantire che la sicurezza e la privacy nel cloud siano rispettate.

La sicurezza nel cloud si può analizzare attraverso un **modello a strati** che descrive i vari componenti del cloud e le loro potenziali vulnerabilità:

- **SaaS (Software-as-a-Service):** Protegge l'accesso ai servizi attraverso l'autenticazione dell'utente e la crittografia dei dati;
- **PaaS (Platform-as-a-Service):** Fornisce un ambiente di sviluppo sicuro per le applicazioni, garantendo la protezione dei dati e l'integrità del software;
- **IaaS (Infrastructure-as-a-Service):** Si occupa delle risorse di infrastruttura virtualizzata, con attenzione alla gestione sicura delle risorse hardware e dei dati;
- **Hypervisor:** Software che funge da livello di astrazione per la gestione delle macchine virtuali e delle risorse fisiche, proteggendo da vulnerabilità nella virtualizzazione;
- **Comunicazione interna:** Fornisce i servizi di comunicazione tra i componenti virtualizzati, cruciali per la sicurezza del sistema complessivo;
- **Archiviazione dei dati:** Gestisce il ciclo di vita dei dati utente, con attenzione alla sicurezza della memorizzazione e al controllo degli accessi;
- **Hardware di base:** Comprende i server fisici e le risorse hardware distribuite ed eterogenee utilizzate dai fornitori di cloud;
- **Centro Dati:** Gestisce l'infrastruttura fisica e la rete, implementando politiche di accesso controllato e misure di continuità operativa.



La **ricerca sulla sicurezza nel cloud** ha identificato diverse vulnerabilità nel sistema, evidenziando la necessità di tecniche di garanzia per validare la sicurezza dei dati e delle applicazioni. Le problematiche di sicurezza sono complesse a causa della **eterogeneità dei sistemi cloud**, della **mancanza di requisiti di sicurezza formalizzati**, e della **necessità di bilanciare sicurezza, flessibilità e performance**.

Un concetto centrale nella sicurezza del cloud è la **trasparenza**, che permette ai fornitori di cloud di mostrare come ottengono e mantengono la conformità alle normative. I fornitori devono spiegare chiaramente come garantiscono la sicurezza dei dati e dei servizi, per migliorare la fiducia degli utenti.

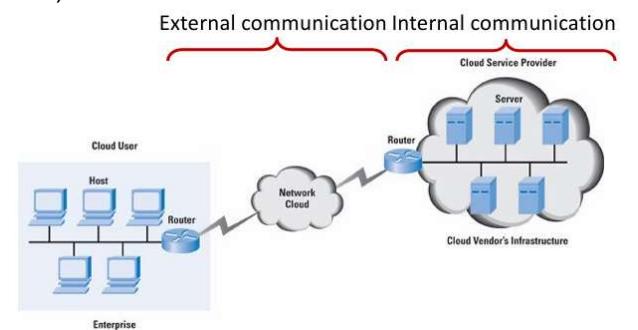
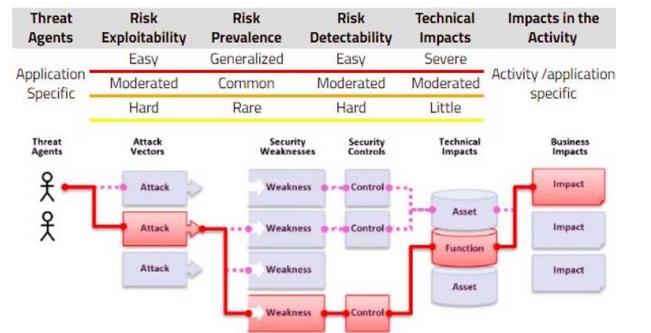
La modellazione della sicurezza nel cloud è essenziale per proteggere i dati, le applicazioni e le infrastrutture dai rischi legati alle vulnerabilità del sistema. Le tecniche di sicurezza devono essere applicate in tutti i livelli dell'architettura cloud e devono tenere conto delle minacce emergenti e delle esigenze specifiche dei diversi attori coinvolti.

## VULNERABILITÀ NEL CLOUD:

Le **vulnerabilità nel cloud** riguardano vari aspetti legati alla sicurezza e alla protezione dei dati, dovuti alle caratteristiche specifiche dell'infrastruttura cloud. I principali tipi di vulnerabilità includono problemi nel livello delle applicazioni, delle piattaforme, della virtualizzazione e della comunicazione, che potrebbero essere sfruttati per attacchi.

Le minacce e le vulnerabilità comuni nel cloud sono:

- **Vulnerabilità nelle Applicazioni Web:** Le minacce a livello di applicazione possono derivare da tecnologie web vulnerabili, che formano i vettori di attacco per i cloud provider. Le dieci minacce più critiche identificate dal **OWASP** sono applicabili anche all'ambiente cloud;
- **Livello Piattaforma:** La vulnerabilità nella piattaforma può derivare da una scarsa verifica e validazione del software in fase di sviluppo, che porta a codici vulnerabili e aumentata possibilità di sfruttamento. La mancanza di un'adeguata separazione delle risorse e il controllo inadeguato sui sistemi operativi possono causare fughe di dati;
- **Virtualizzazione e Hypervisor:** La virtualizzazione, una componente cruciale del cloud, presenta vulnerabilità, come la possibilità di *escape dalla VM*, che potrebbe consentire accessi non autorizzati ad altre macchine virtuali. La gestione della memoria e la protezione dell'isolamento delle VM sono aspetti da monitorare attentamente;
- **Sicurezza dei Dati nel Cloud:** I dati memorizzati nel cloud sono vulnerabili a manomissioni da parte di esterni o interni. Problemi come la gestione inadeguata delle chiavi di crittografia e la memorizzazione dei dati in ambienti condivisi aumentano il rischio di accesso non autorizzato;
- **Sicurezza delle Reti nel Cloud:** La comunicazione esterna tra il cloud e i consumatori è soggetta a vulnerabilità, come attacchi agli IP, vulnerabilità nei protocolli di rete e nelle tecnologie utilizzate. Inoltre, le reti interne al cloud sono vulnerabili a minacce derivanti da risorse virtualizzate condivise;
- **Infrastruttura Fisica e Data Center:** I data center fisici, che ospitano le risorse cloud, sono vulnerabili a disastri naturali, guasti infrastrutturali e minacce da parte di insider malintenzionati. La protezione fisica, la gestione dell'energia e la progettazione inadeguata possono compromettere la sicurezza del cloud;
- **Problemi di Conformità e Trasparenza:** La mancanza di trasparenza sui processi del cloud e le politiche di protezione dei dati può causare problemi di fiducia tra fornitori e clienti. La gestione delle normative legali su dati e privacy, specialmente con la presenza di più giurisdizioni, rappresenta una difficoltà significativa;
- **SLA Debole e Incidenti di Sicurezza:** Un **SLA (Service Level Agreement)** debole può portare a violazioni, compromettendo la fiducia nei servizi cloud. La debolezza nei controlli di monitoraggio delle attività può causare incidenti di sicurezza che, se non affrontati tempestivamente, potrebbero aggravare la situazione;
- **Problemi nei Test e nelle Certificazioni:** La metodologia di test tradizionale non è sufficiente per le applicazioni basate su cloud. È necessario che i fornitori di cloud siano certificati da terze parti per garantire il rispetto delle normative legali e di privacy, supportando le indagini forensi



## SIGNIFICATO DELLA SICUREZZA NEL CLOUD:

Le soluzioni di **sicurezza nel cloud** sono state introdotte per affrontare le vulnerabilità note e proteggere i dati e le applicazioni da attacchi. Di seguito sono descritte alcune delle principali tecniche di sicurezza adottate:

- **Gestione dell'Identità e Accesso (IAM):** Gestisce l'autenticazione e l'autorizzazione degli utenti in tutti i livelli dell'ambiente cloud, garantendo che solo utenti legittimi possano accedere ai dati e alle risorse;
- **Meccanismi di Crittografia:** La crittografia protegge la riservatezza e l'integrità dei dati sia in transito che a riposo. Viene utilizzata anche l'autenticazione tramite firma digitale per garantire l'autenticità dei dati scambiati;
- **Crittografia Omomorfica:** Una tecnica avanzata che permette di eseguire operazioni sui dati cifrati senza decriptarli, proteggendo ulteriormente la privacy;
- **Intrusion Detection and Prevention Systems (IDS/IPS):** Strumenti che monitorano il traffico di rete per rilevare e prevenire possibili attacchi, sia a livello di rete che su macchine virtuali (VM);
- **Protezione delle Applicazioni Web:** L'utilizzo di WS-Security per proteggere i servizi web e garantire che le applicazioni siano sicure e robuste contro le vulnerabilità dei client web;
- **Sicurezza delle Macchine Virtuali (VM):** Le VM possono essere vulnerabili a vari tipi di attacchi. Le contromisure includono la gestione sicura delle immagini delle VM, il controllo sull'accesso e la protezione dei dati sensibili;
- **Sicurezza dei Dati e Storage nel Cloud:** Il cloud storage è vulnerabile a manomissioni e accessi non autorizzati. È essenziale adottare politiche di accesso rigorose e crittografare i dati sensibili. Inoltre, la gestione della privacy e della trasparenza nelle operazioni di backup e recupero dei dati è fondamentale;
- **Misure di Sicurezza Fisica:** La protezione delle risorse fisiche del cloud è essenziale per impedire l'accesso fisico non autorizzato e garantire la continuità aziendale, specialmente durante disastri naturali;
- **Sicurezza del Backup e Gestione della Disponibilità:** La gestione del backup e la protezione dei dati archiviati è fondamentale. È necessario garantire che i dati possano essere recuperati in caso di guasti, e l'utilizzo di meccanismi di bilanciamento del carico può migliorare la disponibilità hardware.

## GARANZIA DEL CLOUD:

Le **misure di garanzia e conformità** nel cloud si riferiscono a un insieme di azioni proattive che un provider adotta per garantire che siano implementate contromisure adeguate ad affrontare le vulnerabilità conosciute nei componenti architetturali del cloud e nelle tecnologie utilizzate, assicurando che vengano rilevati eventuali usi malevoli delle risorse cloud. Queste misure includono il monitoraggio e la valutazione dell'efficacia delle soluzioni di sicurezza implementate.

## SLA (SERVICE LEVEL AGREEMENT):

Il SLA è un contratto tra l'utente del cloud e il provider che definisce:

- Il livello minimo di performance del provider;
- Le azioni che vengono intraprese in caso di violazione dello SLA;
- Le conseguenze della violazione dello SLA.

I requisiti di sicurezza devono essere esplicitamente inclusi nello SLA, insieme ai meccanismi di monitoraggio e misurazione.

## MONITORAGGIO E VERIFICA:

Il **monitoraggio** è cruciale per aumentare la trasparenza e la fiducia nell'ambiente cloud. I log di attività, eventi e traffico consentono di stabilire la responsabilità e aiutano a mettere in atto misure proattive e reattive per proteggere il cloud. La **verifica e validazione** riguardano i test statici e dinamici durante il ciclo di vita dello sviluppo del software. La sicurezza dei software è solitamente focalizzata su autenticazione utente, database di sistema e test di rete.

## CERTIFICAZIONE E RECUPERO DATI:

- **Certificazione:** Il processo di certificazione da parte di autorità accreditate garantisce la sicurezza dei servizi cloud e dei relativi meccanismi di sicurezza;
- **Auditing dei Dati:** È fondamentale garantire l'integrità dei dati utente memorizzati in remoto, specialmente quando gestiti da terzi. I metodi tradizionali di verifica dell'integrità, come le funzioni hash, non sono sufficienti per il cloud; quindi, è necessario implementare tecniche di auditing remoto dei dati (RDA).

## OTTIMIZZAZIONE DEL PROCESSO DI AUDIT:

Per ottimizzare il processo di audit, si seleziona solo una parte dei dati memorizzati per l'auditing remoto. Le tecniche utilizzate includono:

- **Replica dei dati:** I dati sono replicati su più archivi distribuiti;
- **Codifica di cancellazione:** Tecniche come il **MDS (Maximum Distance Separable)** vengono utilizzate per garantire l'affidabilità della replica;
- **Codifica di rete:** Un nuovo blocco di dati viene creato durante la riparazione tramite combinazione lineare dei dati memorizzati.



CoScienze  
Associazione

## DOMANDE CHE SONO STATE FATTE AD ALCUNI DI NOI

---

### PRIMO E SECONDO: voto 28

- Ring Oscillator Puf e come renderlo più sicuro + disegno;
- Tls, autenticazione con Puf, sram Puf;
- Differenza tra Tee e Secure Element.

### TERZO: voto 29

- Arbiter Puf con disegno;
- Jtag exploitation;
- Tee;
- Tipi di tee;
- Ring oscillator puf.

### QUARTO: voto 30

- Cpuf;
- Curve ellittiche.

### NOTIZIE DA ALTRI RAGAZZI

---

*"Di solito chiede lightweight cryptography, tutti i PUF, communication security e physical attacks. Poi non so se ha cambiato"*

*"A me chiese le curve ellittiche, wifi (intende wep, wpa e wpa2 con particolare attenzione a wpa e wpa2), bt on IoT, sistemi vari ma piacque il progetto e ci trattò bene"*

*"BLE lo chiese al mio collega ma non stavo ascoltando"*

*"Ricordo che a un altro chiese Zigbee e cos'è il BT in generale"*



CoScienze  
Associazione