Security in Embedded Systems

The Wireless Sensor Network case

Walter Tiberti

University of L'Aquila (Italy)
Centre of Excellence DEWS
DISIM dept.

23 April 2019

UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Agenda

1. Introduction
2. Security
3. Embedded Systems Security
4. Security in WSN
   - Cryptography
5. Our Works
   - TAKS
   - Intrusion Detection
6. Securing IEEE 802.15.4e

Agenda

Who am i

- **Walter Tiberti**
- MoS in Computer and System Engineering
- PhD student @ University of L'Aquila (supervisor: L. Pomante)
- Github/Gitlab: `wtiberti`
  - Embedded Systems
  - Low-level software (e.g. firmware, drivers, OSes etc.)
  - Reverse-Engineering, Malware Analysis, Penetration Testing
  - Cryptography, Intrusion Detection and Countermeasures
  - Digital Electronics Design and Implementation

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

L'Aquila (Italy)

Spanish Fort, Fontana Luminosa

UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

Basilica di Collemaggio, P. del Duomo

L'Aquila: University

UNIVERSITÀ
DEGLI STUDI
DE L'AQUILA

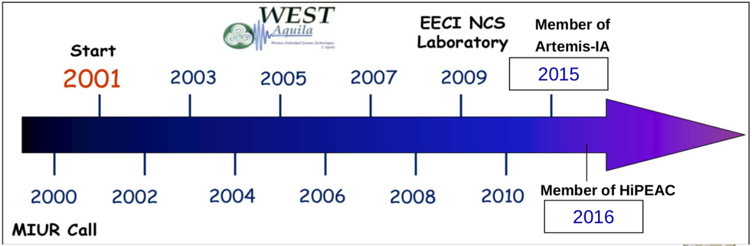
L'Aquila: view from Monteluco di Roio

DEWS

- **Center of Excellence DEWS**
  - **D**esign methodologies for
    Embedded controllers
    Wireless interconnect and
    System-on-chip

CENTER OF EXCELLENCE

http://dews.univaq.it



Embedded Systems Workgroup: Core members

- Coordinator: **Luigi Pomante**
- Senior Researchers:
  - **Tania Di Mascio** (Human-Machine interf., coordination)
  - **Marco Santic** (WSN, Localization)
- PhD Researchers:
  - **Giacomo Valente** (Digital Design, Monitoability)
  - **Paolo di Gianmatteo** (Machine Learning)
- PhD Students:
  - **Vittoriano Muttillo** (HW/SW Co-design)
  - **Walter Tiberti**
  - **Gabriella D'Andrea** (Reconf. Platforms)
  - **Federica Caruso** (Human-Machine interfac.)

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Agenda

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Un-security

- No system is **secure**

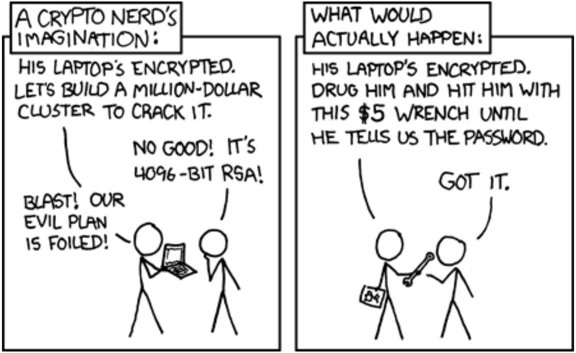


Figure: xkcd.com

Un-security

- No system is **secure**

Un-security: assumptions to make

- Basic assumption: no system is **secure**
- Basic assumption: attackers have **infinite** resources (e.g. money, time, tries etc.)

Un-security: assumptions to make

- Basic assumption: no system is **secure**
- Basic assumption: attackers have **infinite** resources (e.g. money, time, tries etc.)
- Basic assumption: compilers can fail, hardware can fail (e.g. SPECTRE, MELTDOWN, ROWHAMMER)

Un-security: assumptions to make

- Basic assumption: no system is **secure**
- Basic assumption: attackers have **infinite** resources (e.g. money, time, tries etc.)
- Basic assumption: compilers can fail, hardware can fail (e.g. SPECTRE, MELTDOWN, ROWHAMMER)
- Basic assemption: **People** can fail too!

Un-security: assumptions to make

- Basic assumption: no system is **secure**
- Basic assumption: attackers have **infinite** resources (e.g. money, time, tries etc.)
- Basic assumption: compilers can fail, hardware can fail (e.g. SPECTRE, MELTDOWN, ROWHAMMER)
- Basic assemption: **People** can fail too!
- While we are talking, *vulnerabilities* are exploited, found, reported etc.

Date	D	A	V	Title	Type	Platform	Author
2019-04-17	⬇		✓	Oracle Java Runtime Environment - Heap Corruption During TTF font Rendering in GlyphIterator::setCurrGlyphID	DoS	Multiple	Google Security Research
2019-04-17	⬇		✓	Oracle Java Runtime Environment - Heap Corruption During TTF font Rendering in sc_FindExtrema4	DoS	Multiple	Google Security Research
2019-04-17	⬇	▣	✗	DHCP Server 2.5.2 - Denial of Service (PoC)	DoS	Windows	Victor Mondragón
2019-04-17	⬇		✗	ASUS HG100 - Denial of Service	DoS	Hardware	YinT Wang
2019-04-17	⬇	▣	✗	MailCarrier 2.51 - POP3 'RETR' SEH Buffer Overflow	Remote	Windows	Dino Covotsos
2019-04-16	⬇		✓	Microsoft Windows 10 1809 - LUAFV PostLuafvPostReadWrite SECTION_OBJECT_POINTERS Race Condition Privilege Escalation	Local	Windows	Google Security Research

Example: ExploitDB website

`https://www.exploit-db.com/`

Cryptography?

- Cryptography **not necessarily** means Security
  - A system using cryptography can be still vulnerable to other (often subtle) attack vectors
  - Once the keys are known, cryptography is useful no more
  - While cryptographic schemes, on paper, are super-secure, **very** often their implementation isn't so.
    - Examples: PODDLE, CRIME, Heartbleed
    - `https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/`
  - Nonces generated by weak PRNG can be learned and replicated
  - Each cryptographic scheme has its issues (e.g. small exponents, backdoor curves)
- **Rule Thumb:** *"Don't roll your own cryptography"*

How to approach security?

- *The attack is the best defence & The defence is the best attack*
- Assess your system by *attacking it* (**Penetration Testing**)

How to approach security?

- *The attack is the best defence & The defence is the best attack*
- Assess your system by *attacking it* (**Penetration Testing**)

Learning Platforms

- OverTheWire: `http://overthewire.org/wargames/`
- HackTheBox: `https://www.hackthebox.eu/`
- ExploitEducation: `https://exploit.education/`

Agenda

Security in Embedded Systems

- Let's now focus on **Embedded Systems** (in general)

Security in Embedded Systems

- Let's now focus on **Embedded Systems** (in general)
- First: too often, the security aspects are left aside during the design phase.

Security in Embedded Systems

- Let's now focus on **Embedded Systems** (in general)
- First: too often, the security aspects are left aside during the design phase.
- Embedded Systems are even **more vulnerable**, since they are often designed to work in constrained application-specific contexts.
  - Simpler protocols

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Security in Embedded Systems

- Let's now focus on **Embedded Systems** (in general)
- First: too often, the security aspects are left aside during the design phase.
- Embedded Systems are even **more vulnerable**, since they are often designed to work in constrained application-specific contexts.
  - Simpler protocols
  - Simpler HW designs

Security in Embedded Systems

- Let's now focus on **Embedded Systems** (in general)
- First: too often, the security aspects are left aside during the design phase.
- Embedded Systems are even **more vulnerable**, since they are often designed to work in constrained application-specific contexts.
  - Simpler protocols
  - Simpler HW designs
  - Lighter SW: simpler or no OS, applications deal directly with the HW (and malicious code can access HW easilly)

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Security in Embedded Systems

- Let's now focus on **Embedded Systems** (in general)
- First: too often, the security aspects are left aside during the design phase.
- Embedded Systems are even **more vulnerable**, since they are often designed to work in constrained application-specific contexts.
  - Simpler protocols
  - Simpler HW designs
  - Lighter SW: simpler or no OS, applications deal directly with the HW (and malicious code can access HW easilly)
  - Light or no **memory management** (e.g. no virtual memory, no access controls etc.): memory access is often unrestricted

Embedded Systems: Security failures (1)

Embedded Systems: Security failures (1)

- Philips Hue lightbulbs
- `https://youtu.be/Ed1OjAuRARU`

Embedded System Security: Future

- **More powerful**?

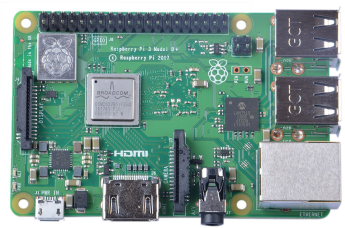
Embedded System Security: Future

- **More powerful**? Embedded Systems will start performing like modern bigger platforms, with all the common security issues

Embedded System Security: Future

- **More powerful**? Embedded Systems will start performing like modern bigger platforms, with all the common security issues

Example: Raspberry PI



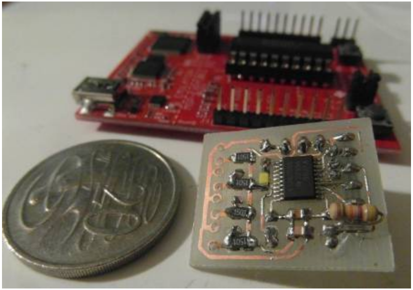
- Quad-core ARM CPU, GBs of RAM
- BT, 802.11, HDMI, Ethernet
- Can run normal OS

Embedded System Security: Future

- **More powerful**? Embedded Systems will start performing like modern bigger platforms, with all the common security issues
- **Smaller**?

Embedded System Security: Future

- **More powerful**? Embedded Systems will start performing like modern bigger platforms, with all the common security issues
- **Smaller**? Less memory, less performance, smaller CPU/MCU, harder to implement security-related functionalities

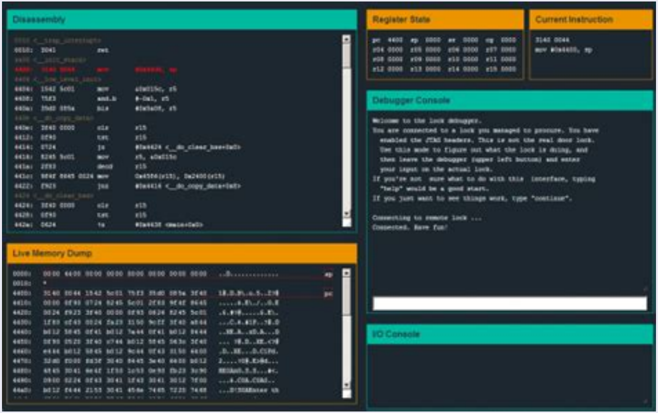


- MSP430-based board (16 bit)
- 10 KB or RAM, 48 KB of Flash storage

Learning Platforms

MSP430 security

Microcorruption: `https://microcorruption.com`



Agenda

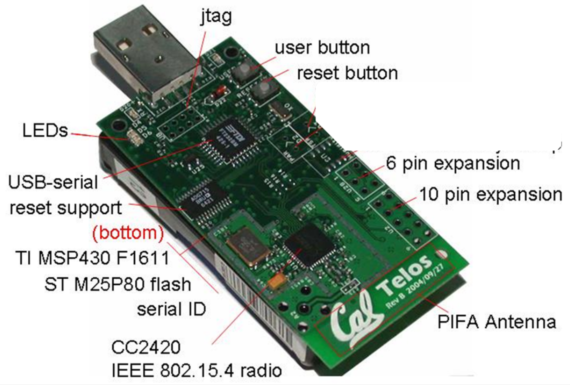
UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Wireless Sensor Networks

- Wireless Sensor Network (WSN)
- Quick recap
  1. **Small**, **energy-constrained** (battery-powered) nodes (called *motes*)
  2. Motes are equipped with a **radio transceiver** for wireless interconnection
  3. A set of **sensors** are embedded in the mote (directly or by means of *sensorboards*
  4. Small storage, small performance, small costs
  5. Flexible platforms: motes can be densely distributed and/or organized to cover big areas

IEEE 802.15.4

- Nowadays, the term WSN is becoming more *vague*
- In this presentation, we will discuss about **IEEE 802.15.4**-based WSN



Security Overview

In general, in WSNs:

- Mote hardware side: often no security (i.e. attackers can **access the IO pins** with no effort.
- Some mote MCU have **hardware primitives** for symmetric/public-key cryptography (e.g. the TI CC3825)
- The radio channels used are (mostly) in the**unlicensed ISM 2.4GHz bands** (i.e. everybody can listen and transmit over the channel)
- The SoA OS for WSN (TinyOS, ContikiOS, RIOT) have only few security features. External software can provide other features with limitations (e.g. TinyECC)
- iIt is easy to perform different **jamming attacks** (from constant jamming to dynamic self-adapting jamming)
- Some of the 2.4GHz channels of IEEE 802.15.4 **collides**

WSN: Using Symmetric Ciphers?

- Using *Symmetric Cryptography schemes* (e.g. block ciphers such as AES or stream ciphers e.g. Salsa/Chacha20) are generally fast and lightweight, but they requires the secret key to be **available** (e.g. preprogrammed) before any transmission
- $\rightarrow$ *Key Distribution Problem*

WSN: Using Public-Key Cryptography? (1)

- Using *Public-Key Cryptography schemes* solves the key distribution problem (e.g. Diffie-Helman) but is expensive in terms of performance and storage.

WSN: Using Public-Key Cryptography? (2)

Example: RSA

With a maximum of ~ 100 bytes available as MAC payload and using a SoA key size e.g. 2048 bit = 256 bytes, every transmission which requires a key exchange has to be *splitted*.

Example: ECC

Even though ECC keys require less space (e.g. 192bits) the computation involved (e.g. Point Addition/Multiplication) are possible but they can be very expensive.

From the IEEE 802.15.4 standard

- Section 9 "Security"
- AES 128bit CCM as (authenticated) symmetric encryption
- Not enough information to implement a complete solution:
  - 128bit AES is not super-secure
  - CCM mode (CTR+HMAC) have **problems** is the nonces are not carefully "choosen"
  - **No mechanism for key distribution or storage**

Key storage?

Consider the following C source code

```c
#include <stdint.h>
#define KEYLENGTH 128

uint8_t supersecretkey[KEYLENGTH/8] = {
  0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
  0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
};

int main()
{
        // .. application code ...
        while(1);

        return 0;
}
```

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

Key storage?

Consider the following C source code

```c
#include <stdint.h>
#define KEYLENGTH 128

uint8_t supersecretkey[KEYLENGTH/8] = {
   0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
   0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
};

int main()
{
        // .. application code ...
        while(1);

        return 0;
}
```

Compiling into IHEX format for mote programming:

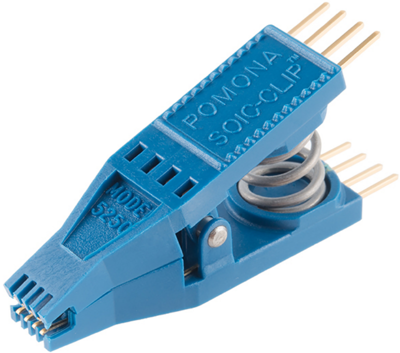
```
:10010000214601360121470136007EFE09D2190140
:100110002146017E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

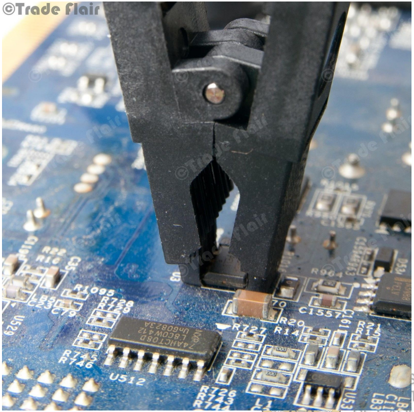
Start code | Byte count | Address | Record type | Data | Checksum

```
:10400000554220135D0085A8245101131400039FF
:104010003F4010000F93082492421011120012F837B
:104020009F4F50400011F8233F4000000F9307249A
:10403000924210112001 1F83CF431011F923044134
:104040002453FF3F32D0F000FD3F30404E4000137C
:1040500011223344556677881122334455667788 98
:10FFE0004A404A404A404A404A404A404A404A40C1
:10FFF0004A404A404A404A404A404A404A400040FB
:040000030000400 0B9
```

Key storage...failure

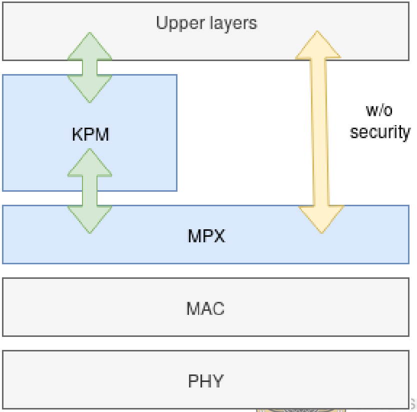
Dumping data from Flash/EEPROM chips:





IEEE 802.15.9 standard

- IEEE 802.15.9
- Recent standard, it addresses the *Key Transport* mechanisms
- Overview:
  - Additional layer on top of MAC (**MPX**)
  - Fragmentation and Multiplexing
  - Additional (vertical) layer for Key management (**KMP**)



Agenda

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

TAKS

- Ad-hoc cryptographic cryptography techniques
- *Topology Authenticated-key Scheme* (**TAKS**)
- **Hybrid Cryptogrphy**: TAKS use both public-key and symmetric cryptography features
- **Lightweight**: designed to run on top of WSN platforms

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

TAKS

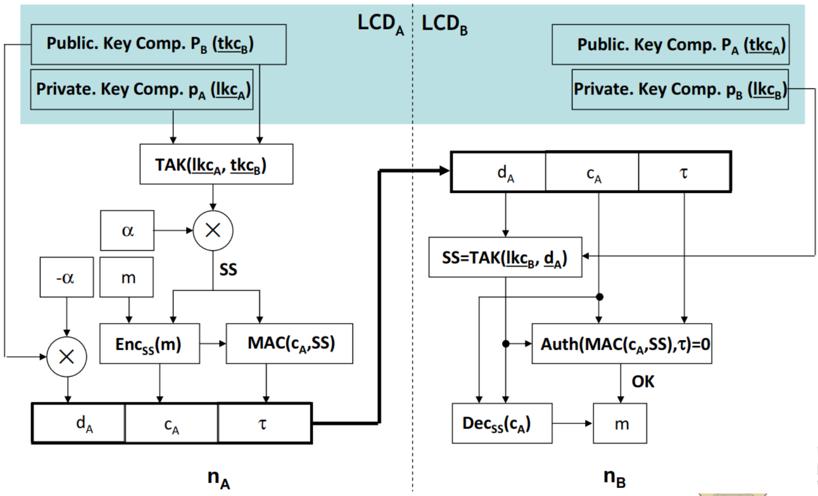
Features:

- **Objective**: creating a *shared secret* with public-key like mechanisms to be used as symmetric key for encrypting/decrypting and to generate a *signature* for authentication.
- **How?** Vector algebra over finite fields (usually $GF(2^n)$)
- Two mode of operation: **Point-to-Point** (star topologies) and **Cluster-wise** (for cluster-wise topologies)

TAKS: idea

- A WSN topology is planned (Planned Network Topology - **PNT**)
- A set of *Key components* is generated offline for each node in the PNT (Local Configuration Data - **LCD**)
    - a node-local, private component (Local Key Component, **LKC**) is pre-programmed in the mote
    - a set of public components (Transmit Key Component - **TKC**)
    - a set of restricted components (Topology Vector - **TV**) known only to the motes enabled to communicate with target node

TAKS: block diagram



TAKS: block diagram

- The **TAK()** function combines the nonce and two components into the *shared secret SS*
- Multiple definitions of **TAK()** are possible. An example is[1]:

$$TAK_{i \to j} = \alpha * LKC \times TV_{i \to j} = KRI \times TKC_{i \to j}$$

$$= \alpha * \left\| \begin{matrix} \hat{i} & \hat{j} & \hat{k} \\ lkc_1 & lkc_2 & lkc_3 \\ tv_1 & tv_2 & tv_3 \end{matrix} \right\| = \left\| \begin{matrix} \hat{i} & \hat{j} & \hat{k} \\ kri_1 & kri_2 & kri_3 \\ tkc_1 & tkc_2 & tkc_3 \end{matrix} \right\|$$

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

[1] In this case, LKC, TV and TKC are 3-dimensional vectors over $GF(2^n)$

TAKS: Sending..

- Alice wants to transmit to Bob a message
- Alice has LKC_a and Bob's TKC_b. She uses the $TAK()$ function to combine the two key components
- Alice generate a **nonce** (α) and multiplies it with the result of the component combination to obtain a **shared secret** SS
- She uses SS **as symmetric key** to encrypt the data and to **generate a message authentication code** from the cipher text
- Alice creates also the **key reconstruction information** d_a
- Finally, Alice sends $ciphertext|d_a|MAC$ to Bob

TAKS: ..Receiving

- Bob receives $ciphertext|d_a|MAC$ from Alice
- Due the construction of $TAK()$, $TAK = LKC_a \cdot TKC_b = -LKC_b \cdot TKC_a$, Bob uses $d_a \cdot LKC_b = (-\alpha \cdot TKC_a) \cdot LKC_b = SS$ to derive the SS
- Bob checks the MAC and, if valid, decrypts the message with SS

Practical Example

Demo

ECTAKS: Prologue

- Elliptic Curve Cryptography (**ECC**)
- Old concept, but recently re-discovered
- Public-Key scheme (ECIES, ECDSA, ECDH, ECQV)
- Curve, Point, PointAdd, PointMul, ECDLP
- PubKey = Point, PrivKey = Scalar
- Standard Curves

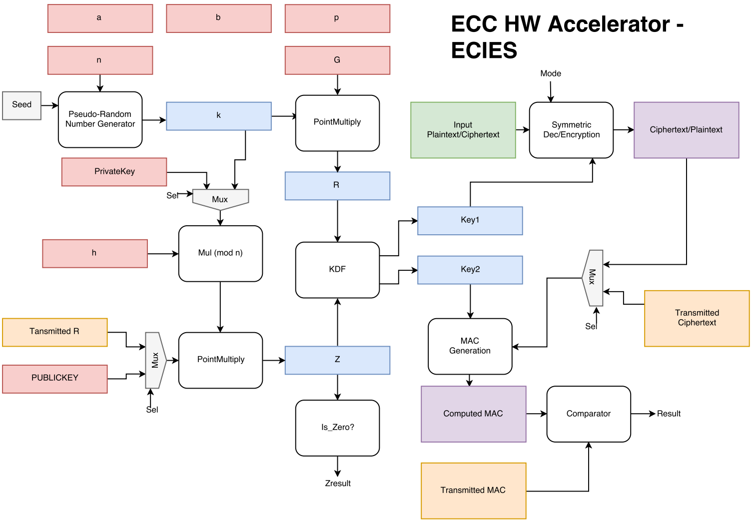
ECTAKS

- **Work-in-Progress**
- Elliptic Curve-based TAKS (**ECTAKS-ECIES**, **ECTAKS-ECDSA**)
- Idea: add TAKS mechanism to EC Points: **TAKS Key Components = Points**
- TAK function combines points into a **private** (scalar) key, which can be used to generate the ECC Public Key
- Target platforms: *TinyOS, Contiki-OS, RIOT-OS*

ECHARP-WSN

- **Work-in-Progress**
- **ECC HW Accelerator** on Reconfigurable Platform for WSN motes
- Idea: *Faster processing* $\rightarrow$ *Less Time* $\rightarrow$ *Less Energy*
- We are developing a ECC HW Accelerator to be embedded and controlled by WSN motes and activated **on-demand** (default: *power off*)
- Full **RTL** design to improve area occupation, performances and energy impact
- **Flexible**: no assumption is taken, e.g. any curve can be used and changed at runtime
- **Very good** initial results (PointAdd, PointDouble, PointMul)

Example: block diagram for ECIES



WSN: Detect anomalies and attacks

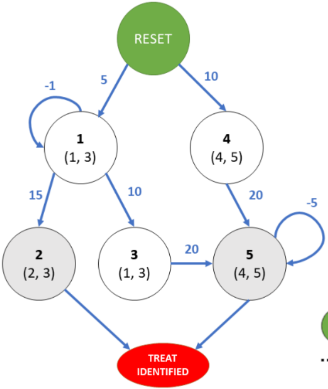
- **Intrusion Detection Systems for WSN**
- *Scenario*: an *attacker* targets the WSN and tries to change the behavior of motes remotely in order to manipulate the data exchanged or to leak information (e.g. cryptographic keys)

WIDS

- *WSN Intrusion Detection System* (**WIDS**)
- Design of a WSN IDS based on *Weak Process Models*(**WPM**
- Attacks are modeled into WPM represented as a graph in which the nodes are the **states** in which the mote could be at any time and the edges are the possible **transitions** from and to states
- WIDS tries to perform a state **estimation** based on events (**observables**)

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

WIDS example

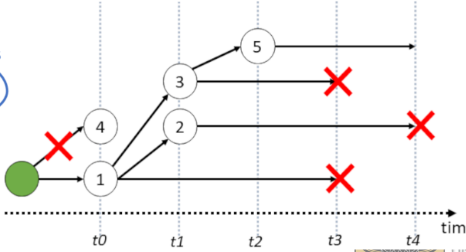


Observables:

t0 -> 1
t1 -> 3
t2 -> 4
t3 -> 2
t4 -> 5

State traces: (only last state is shown)

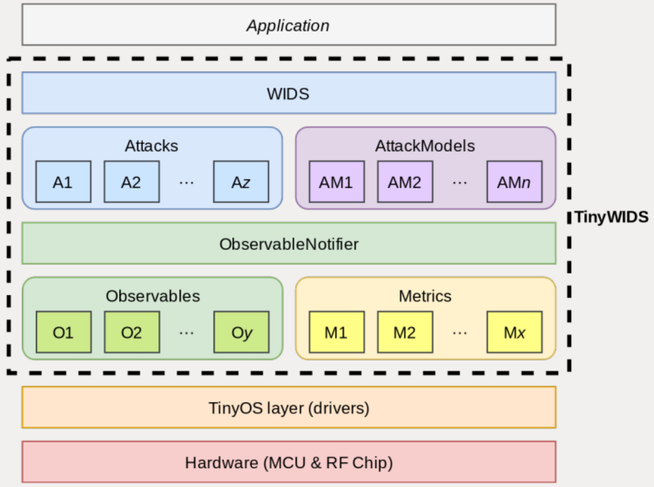
t0 -> {1}
t1 -> {1,2,3}
t2 -> {1,2,3,5,4}
t3 -> {2,5,4}
t4 -> {5}



WIDS implementation: *TinyWIDS*

- Implementation of WIDS on top of TinyOS
- Acquisition of data from the hardware and lower software-layer of the motes in the form of **Metrics**
- Metrics are checked against thresholds. If the checks fails, **observables** are created
- Observables are collected at runtime and the WPM attack models are updated
- If any attack model has an estimated state considered dangerous, a notification is sent

TinyWIDS architecture



WIDS: Countermeasures to attacks

- **Work-in-Progress**
- Question: *When an attack is detected, what can we do?*
- We are studing different techniques to both **contain the attacks** and **retrieving information** of it upon detection

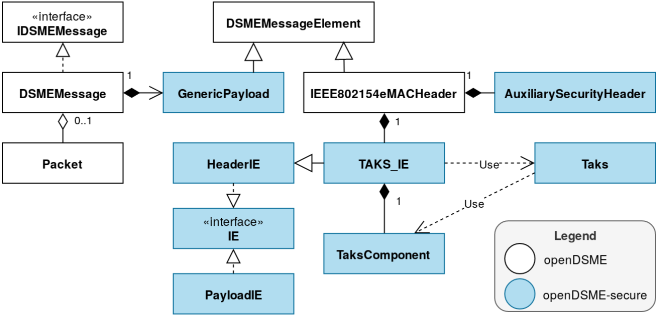
Agenda

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

TAKS in IEEE 802.15.4e

- Done here at CISTER!
- TAKS has been adapted and implemented in the Omnet++ based DSME MAC Behavior of the IEEE 802.15.4e (**OpenDSME**)
- OpenDSME has been enhanced by adding support to:
  - IEEE 802.15.4-2015 MAC layer Information Elements (**IE**)
  - Specific TAKS IE (Header IE)
  - Generic packet content (GenericPayload)
- The code is on Github: `https://github.com/wtiberti/openDSME/tree/Security`

TAKS in IEEE 802.15.4e

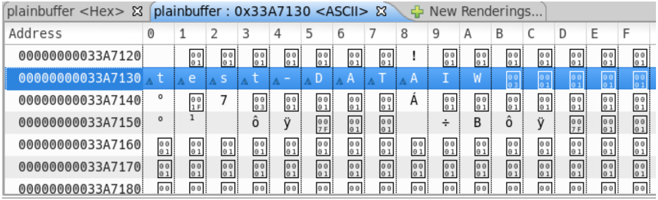


TAKS in IEEE 802.15.4e

Before:

[4] (BytesChunk) : BytesChunk, length = 10 B, bytes = {9, 54, 69, 143, 142, 209, 145, 7, 15, 150}
- mutable = false (bool)
- complete = true (bool)
- correct = true (bool)
- properlyRepresented = true (bool)
- chunkLength = 80 (inet::b)
- bits[3] (string)
- bytes[1] (string)
  - [0] 09 36 45 8F 8E D1 91 07 0F 96

After:



References

TAKS / WIDS / TinyWIDS:

- https://ieeexplore.ieee.org/document/5345623
- https://ieeexplore.ieee.org/document/6583643
- https://ieeexplore.ieee.org/document/6775056
- https://ieeexplore.ieee.org/document/5703728
- https://ieeexplore.ieee.org/document/4660137
- https://dl.acm.org/citation.cfm?doid=3178291.3178293

Thank you!