# Authentic Execution in Smart Farming

Second Thesis presentation

Student: Gianluca Scopelliti
Promotor: Frank Piessens
Supervisors: Jan Tobias Mühlberg, Fritz Alder

# Recap

# Recap: what is this thesis about?

- **Goal**: provide a secure implementation of a distributed, event-driven application

- In practice: extend the concept of «Authentic Execution» with new features
  - Support for Intel SGX
  - Integration with Sancus

- Application: Smart Farming

# Authentic Execution between SGX enclaves

# Introduction

- **Goal**: keep the same structure and philosophy of the Sancus implementation
  - The developer defines
    - The main logic of the modules (with some **annotations** in the code)
    - A description of the system to be built (**descriptor file**)
  - All the rest is added at compile time
    - Authentic Execution, Enclaved Execution

# Platform

- The applications are written in *Rust*
  - Modern, efficient
  - **Safe**

- *Fortanix EDP* is the framework used to write SGX applications
  - Full abstraction over the SGX layer
  - Allows to write a SGX module as a normal, native application

# Input: from the developer to the framework

- The developer passes as input a folder, containing:
  - Description of the system
    - Specified in a **configuration file**
  - Description of the single modules
    - Each module is a separate **Rust project**

```
example/
├── input.json
├── sm1
│   ├── Cargo.toml
│   └── src
│       └── lib.rs
└── sm2
    ├── Cargo.toml
    └── src
        └── lib.rs
```

# Defining the system

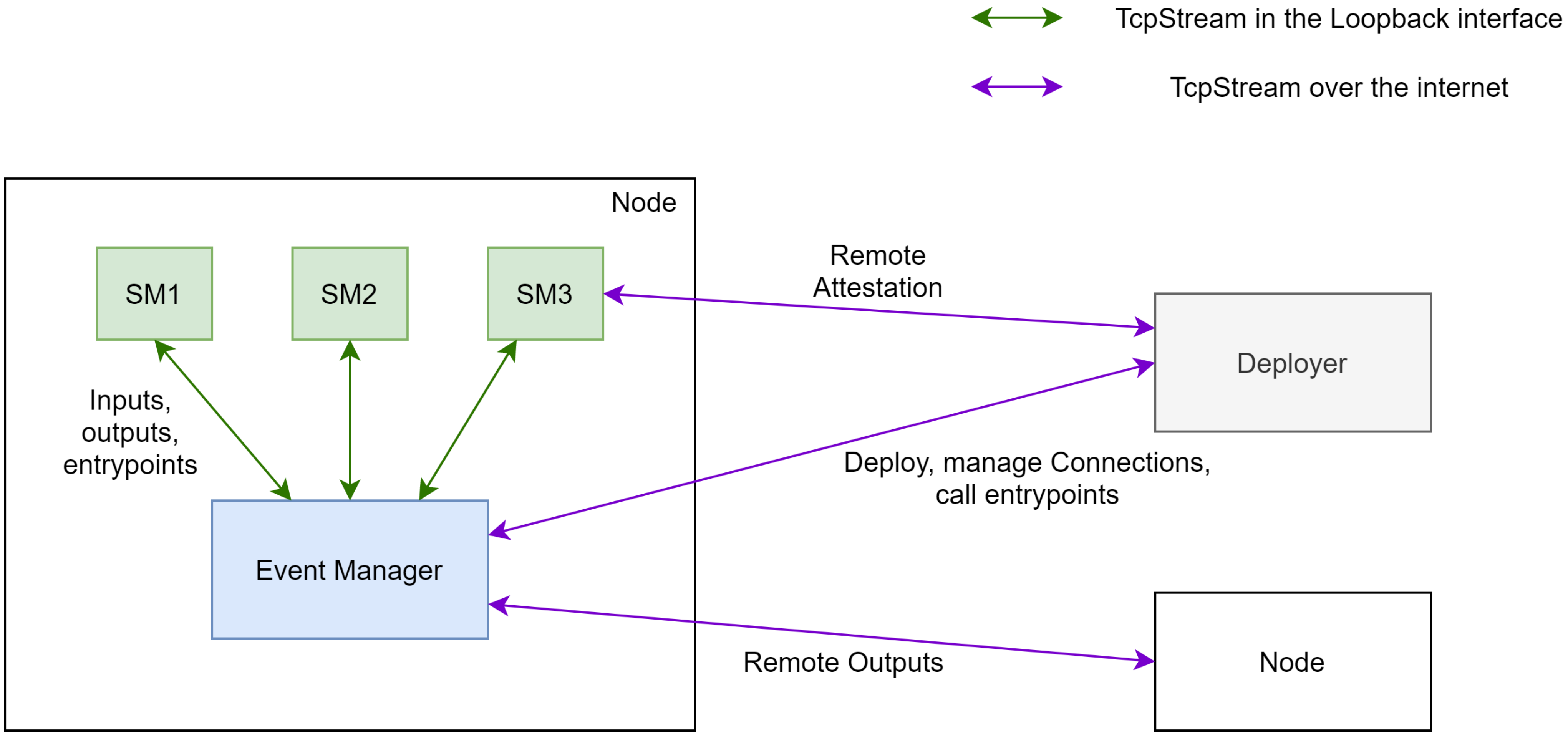
- The input JSON file contains a full description of the system
- **Nodes**
  - Subsystems
  - Each node has an Event Manager
- **Modules**
  - Each module belongs to a node
  - Except for Remote Attestation, a module directly communicates **only** with the EM
- **Connections**
  - *Trusted path* between the output of a module and the input of another

```
{
  "nodes" : [
    {
      "name" : "node1",
      "ip" : "127.0.0.1",
      "em_port" : 5000
    },
    {
      "name" : "node2",
      "ip" : "127.0.0.1",
      "em_port" : 6000
    }
  ],

  "modules" : [
    {
      "name" : "button",
      "node" : "node1"
    },
    {
      "name" : "lcd",
      "node" : "node2"
    }
  ],

  "connections": [
    {
      "from_module": "button",
      "from_output": "button_pressed",
      "to_module": "lcd",
      "to_input": "show_value"
    }
  ]
}
```

# The complete scheme



# Security concerns

- Only SMs and Deployer are considered **trusted**
  - Event managers, nodes and network **are not.**

- The same principles described in the «Authentic Execution» paper have been implemented
  - **Remote Attestation**: ensures that a module is correctly loaded into a node
    - A **Master Key** is established during the process
  - Each connection between modules is protected with a **Connection Key**

# Defining a Module

- The developer creates a Rust Cargo library
  - Logic of the module
  - Inputs, Outputs, Entrypoints

- Automatic generation of the missing code
  - *main* function
  - Code for Authentic Execution
  - Code for Remote Attestation
  - Dependencies for Enclaved Execution

```rust
/* --- user-defined constants, imports, etc.. --- */

static VALUE : u32 = 42;

/* --- Inputs, Outputs, Entrypoints --- */

//@ sm_output(set_tap)

//@ sm_entry
pub fn say_hello(_data : &[u8]) -> Result<Vec<u8>, Vec<u8>> {
    authentic_execution::debug("ENTRYPOINT: say_hello");

    println!("Hello from {}!", *authentic_execution::MODULE_NAME);

    authentic_execution::success("Ok")
}

//@ sm_input
pub fn sensor_data_received(data : &[u8]) {
  authentic_execution::debug("INPUT: sensor_data_received");

  let enable = analyze_data(data);

  set_tap(&enable);
}


/* --- User-defined functions --- */

fn analyze_data(data : &[u8]) -> Vec<u8> {
  // do computation..
 }
```

# Details

- After performing Remote Attestation, the module will listen for messages (events) sent by the Event Manager

- Message: [<Entrypoint ID> <data>]
    - EID 0: *set_key*
    - EID 1: *handle_input*
    - The other entrypoints are defined by the developer

# Future extensions

- Store the Master Key using *SGX* ***data sealing***
  - Perform RA only the first time the module is executed
  - Only the module can retrieve the key

- N:N relationships between inputs and outputs
  - Currently only 1:1 relationships allowed (for simplicity)
  - Multiple connections would be useful
    - e.g. a sensor output connected to both a «database» and a «computation» enclave

# Conclusions

- The framework provides a very easy way for developers to write distributed SGX applications

- **Trusted paths** between modules, in terms of *confidentiality*, *integrity* and *authenticity* of the data
  - Availability is out-of-scope (e.g. EM might not deliver messages)

- The source of the path cannot be trusted
  - No *secure I/O* in SGX
  - Need to use SGX enclaves in combination to other ones (-> Sancus)

# Next steps

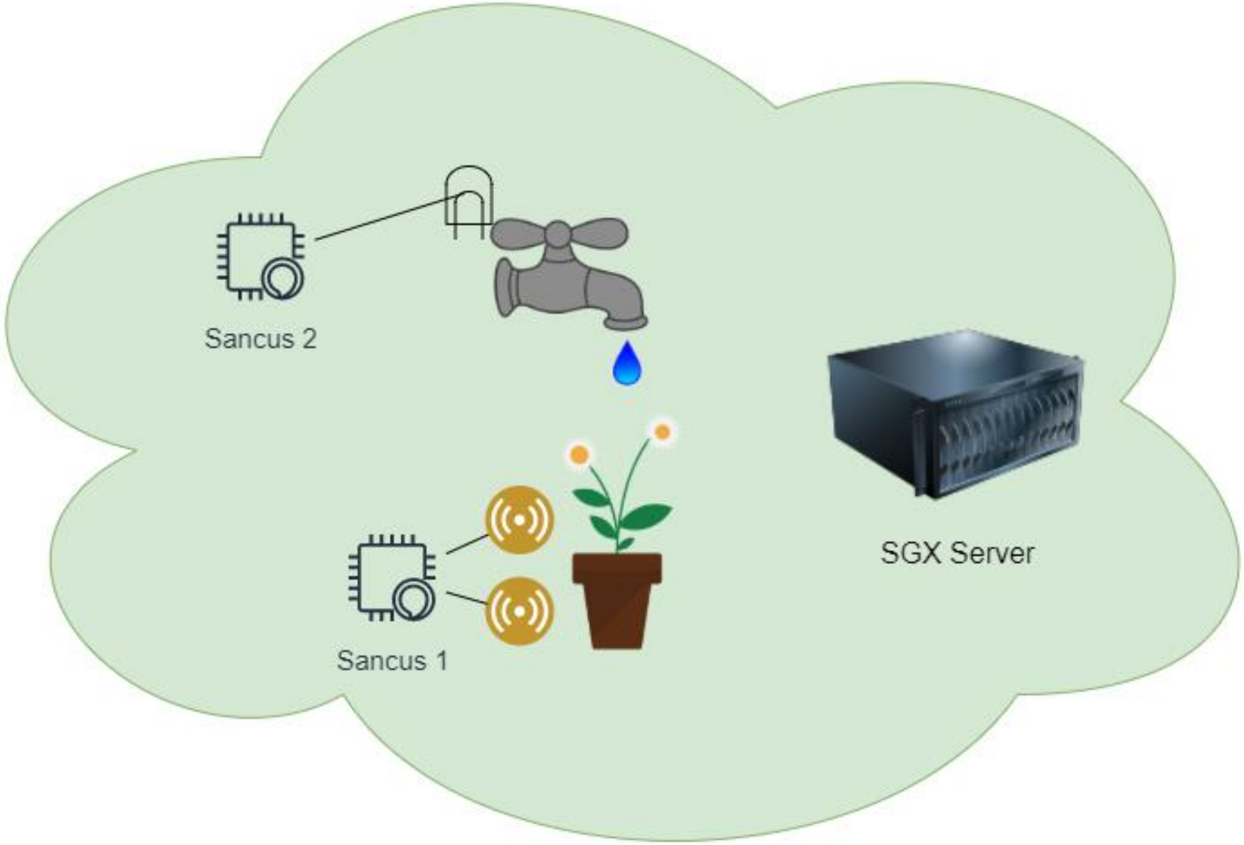
# Integration with Sancus

- Sancus: Trusted Execution Environment for embedded devices
  - Important feature: **Secure I/O**
- Combination of Sancus and SGX enclaves: **full trusted paths**
  - From an input (e.g. sensor) to an output (e.g. actuator)
- Authentic Execution for Sancus devices already implemented
  - Goal: «merge» the two frameworks into a single one

# Prototype for Smart Farming

- Simple application, illustrated in the first presentation

- Automatic water supply of a flowerpot
  - **Sancus 1:** retrieve data using sensors
  - **SGX server:** execute some logic and make decisions
  - **Sancus 2:** enable/disable the water tap



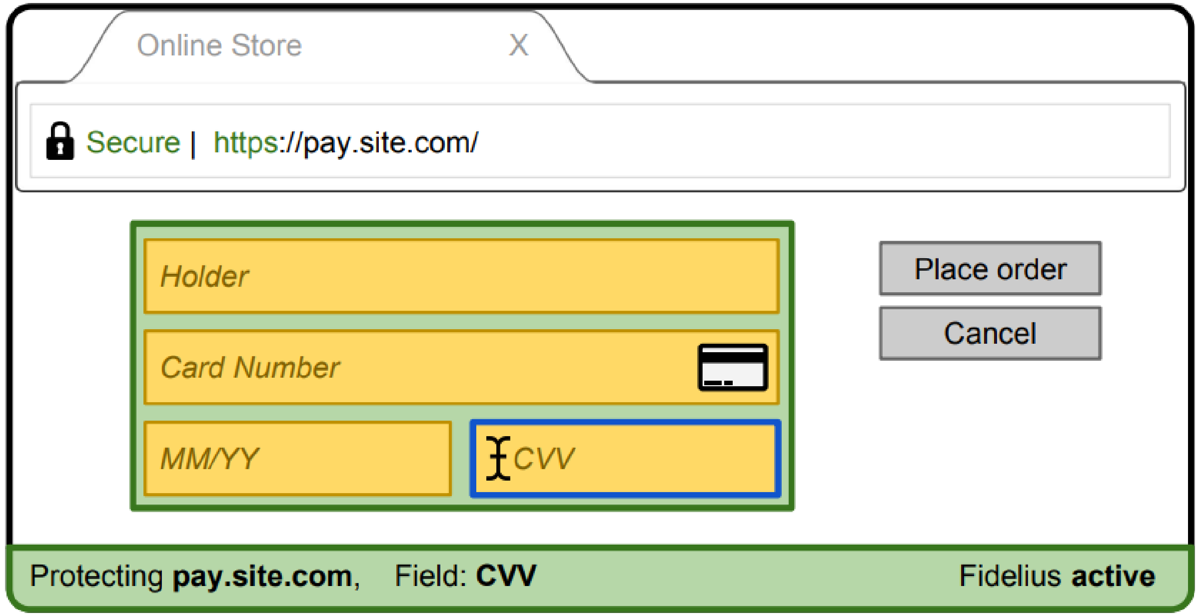
# Other ideas

- **Availability concerns:** implement some «backup» logic to be executed when availability is not guaranteed (e.g. the network goes down)
- **General network API:** Communication between Event Managers in different nodes can also be performed using different mediums
- **End-user application:** a dashboard for the end-user (i.e. the farmer) used to monitor the system and send commands

# Related work

# Fidelius: Protecting User Secrets from Compromised Browsers

- Developed by researchers from the Stanford University
- **Goal**: secure sensitive form fields of a web page
- Establishment of a *trusted path* from end-user to remote server
- Main concern is **confidentiality** of data (e.g. credit card info)
  - Different from ours (**integrity**)



# Conclusions

# Conclusions

- The SGX implementation brings the Authentic Execution framework to the next level
  - We can now define modules to perform some expensive computation..
  - ..or to store data into a central database
  - Microcontrollers alone cannot perform these operations

- The framework can be easily extended in future work
  - New features
  - Support for other architectures
  - Support for other comm. mediums

# References

- Authentic Execution
- Rust programming language
- Fortanix EDP
- Remote Attestation Rust code
- Fidelius