Secure and Minimal Architectures for Establishing Dynamic Root of Trust in (remote) Embedded Devices

Joint work with:
Karim El Defrawy (HRL), Aurelien Francillon (Eurecom)
Daniele Perito (Square), Kasper Rasmussen (UCI)

Roadmap

• Low-end embedded devices and CPS
• Remote device attestation: motivation
• POSE: proof-of-secure-erasure
• SMART: blueprint and goals
• Building blocks
• Implementation details
• Summary + future work

Low-End Embedded Devices

- Low-cost, low-power devices
  - e.g., actuators, sensors, controllers
- Typically built around a Micro-controller Unit (MCU)
- Limited facilities:
  - Memory, e.g.:
    - 4 KB Data Memory (SRAM)
    - 128KB Program Memory (Flash)
  - Power
  - Computation capabilities
- For example
  - TI MSP430
  - Atmel AVR
  - Raspberry Pi

Low-End Embedded Devices (contd.)

- Memory: program and data
- CPU
- Integrated clock
- As well as:
  - Communication interfaces (USB, CAN, Serial, Ethernet, etc.)
  - Analog to digital converters
  - ...
CPS: Cyber-Physical Systems

“A cyber-physical system (CPS) is a system where there is tight coordination of the system’s computational and physical elements, though sensors and actuators”

Why Security?

- CPS are built to be reliable, fault-tolerant
- Security is/was treated as an afterthought
- Ease of management and cost motivates wireless connectivity, e.g.:
  - Implanted medical devices can be accessed via home readers through an RF channel
  - Automotive component systems are starting to get connected via wireless
  - Same goes for building automation systems (BAS)
- Indirect connectivity to the Internet?

Recent Attacks

- Stuxnet [1] (also DUQU)
  - Infected controlling windows machines
  - Changed parameters of the PLC (programmable logic controller) of centrifuges in Iranian nuclear reactors
- Attacks against automotive controllers [2]
  - Internal controller-area network (CAN)
  - Exploitation of one subsystem (e.g., bluetooth) allows access to critical subsystems (e.g., braking)
- Medical devices
  - Insulin pumps hack [3]
  - Implantable cardioverter defibrillator [4]
- Most effective CPS attacks are remote infestations, i.e., not physical or hands-on...

What can we do?

- Prevention?
- Detection?
- Disinfection?

Remote Attestation

Definitions:
- 2-party protocol between trusted verifier and untrusted prover
  - A service that allows the former to verify internal state of the latter

Where:
- Prover – untrusted embedded device
- Verifier – trusted reader/controller/base station
  - Internal state is composed of:
  - Code
  - Registers
  - Data Memory
  - I/O

Prior work – two types of attestation:
- Secure Hardware-based (e.g., TPM) – uses OTS components
- Software-based – uses custom checksum code
  - Does not support multi-hop communication (i.e., prover not truly remote)

Attestation + related services

- **Attestation** → Does prover run intended code and nothing else?
- **Code Update** → How to replace old code on prover with new code such that nothing else remains?
- **Secure Erasure** → How to make sure that prover really erased all stored data?

Note: one way to “force” attestation is by:
1. secure erasure, followed by 2. code update

Remote Attestation

- Malicious software will lie about the software state of prover
- Need to have guarantees that prover is not lying

Verifier

Generate nonce

Challenge (send nonce)
and (maybe) code

Prover

Verify checksum

Response (Send Checksum)

Compute Memory Checksum with nonce

SW-based Attestation

- Prover has no architectural support for attestation
  - Commodity/legacy device
  - Peripheral, e.g., adapter, camera, keyboard, mouse
- Verifier sends customized (random-seeded) checksum routine which covers memory in a unique pattern
- Prover runs checksum over specific memory range, returns result
- Verifier uses precise timing to determine presence/absence of malware
  - Main idea: malware has nowhere to hide, no place to go... Even if it does, delay will be noticed

For this to work:
- Verifier/Prover r/t must be either negligible or stable-and-precise
- Checksum code must be minimal in both speed and space
  - How can one prove that?
  - If not the case, even non-remote sw-attestation not secure
- Prover must be unable to get outside help
  - No extraneous communication!
**SW-based Attestation**

Some discouraging results:

- “Code injection attacks on Harvard-architecture devices”, ACM CCS’08

**HW-based Attestation**

- Prover has architectural support for attestation (e.g. a TPM), including a unique secret
- Verifier sends a challenge (nonce) and memory range
- Prover computes authenticated checksum (e.g., HMAC), returns result
- Verifier checks result

For this to work:

- Verifier/Prover r/t must be bounded but not fixed/stable
- No code minimality/speed assumptions
- Prover can get outside help

However:

- TPM is a heavy-weight approach, not suitable for low-end devices
  - Cost, size, etc.
- Not clear what features are really needed for attestation

**Proofs of secure erasure (PoSE)**

- Clean-slate approach
- Addresses attestation, code update and erasure
- Design goal:
  - verify all prover memory/storage with provable security
- Assumptions
  - Common assumptions (previous slide) PLUS

**PoSE**

- Before PoSE
  - Original Code
  - Malicious Code
- During the protocol (proof of possession of randomness)
  - Fresh Randomness
- End of PoSE
  - Updated Code

**Prover**
ROM?

Mask ROM is generally cheaper than flash memory.
Flash has supplanted ROM on embedded devices.
Other ways to "emulate" ROM, e.g.:

- Lockable memory
  - ATMEGA128 [5] allows a portion (up to 4K) of its flash memory to be "locked". Unlocking requires physical actions.
- One-time Programmable (OTP) memory

Base Case Protocol

\[
\begin{align*}
[1] & \quad P \leftarrow V : R_1, \ldots, R_n \\
[2] & \quad P \rightarrow V : R_1, \ldots, R_n \\
[3] & \quad P \leftarrow V : C_1, \ldots, C_n \\
[4] & \quad P \rightarrow V : \text{ACK or } H(C_1, \ldots, C_n)
\end{align*}
\]

Things to note:
- Not cheap
- Does not detect malware, only gets rid of it

POSE Code

Two subroutines:

1. Receive-and-Write (bit, loc)
   - Obtains and writes prover-supplied bit in prover-specified location

2. Read-and-Send (loc, bit)
   - Reads and returns bit from prover-specified location

Two trivial observations

\[
\begin{align*}
[1] & \quad P \leftarrow V : R_1, \ldots, R_n \\
[2] & \quad P \rightarrow V : R_1, \ldots, R_n \\
[3] & \quad P \leftarrow V : C_1, \ldots, C_n \\
[4] & \quad P \rightarrow V : \text{ACK or } H(C_1, \ldots, C_n)
\end{align*}
\]

- Step 3 need not return entire \( n \) bits
- Step 1 does not require "true" randomness
  - Good quality pseudo-randomness will suffice
Optimized protocol

Note: need MAC in ROM!

\[ K = R_{n-k+1} \ldots R_n \]

1. \( P \leftarrow V : R_1, \ldots, R_n \)
2. \( P \rightarrow V : MAC_K(R_1, \ldots, R_{n-k}) \)

Parameters

MAC constructions on MicaZ.

(a) Energy consumption and time

<table>
<thead>
<tr>
<th>MAC</th>
<th>Time (sec)</th>
<th>Energy (µJ/byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC-MD5</td>
<td>0.45</td>
<td>1</td>
</tr>
<tr>
<td>HMAC-SHA1</td>
<td>0.95</td>
<td>3.5</td>
</tr>
<tr>
<td>Skipjack CBC-MAC</td>
<td>1.12</td>
<td>1.5</td>
</tr>
</tbody>
</table>

(b) Code and working memory required

<table>
<thead>
<tr>
<th>MAC</th>
<th>ROM (bytes)</th>
<th>RAM (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMAC-MD5</td>
<td>9,728</td>
<td>110</td>
</tr>
<tr>
<td>HMAC-SHA1</td>
<td>4,646</td>
<td>124</td>
</tr>
<tr>
<td>Skipjack CBC-MAC</td>
<td>2,990</td>
<td>100</td>
</tr>
</tbody>
</table>

What’s next?

- POSE is a provably secure approach for a specialized (non-remote) case and even that under some limiting assumptions
- What about remote attestation?
- Clearly, ROM is a fundamental requirement
- What else?

SMART: Secure & Minimal Architecture for Remote Trust

Motivation:
- **Hardware-based** techniques (TPM-based) too expensive for low-end embedded devices
- **Software-based** attestation is insecure in remote settings
  - NOTE: might be okay for one-hop settings assuming either
    - (1) adversarial silence during attestation, or
    - (2) verifier controls all verifier’s communication
- What is the minimal set of architectural features needed to achieve remote attestation?

Desired properties:
- Minimal modifications to current platforms
  - Lowest # of additional gates
- Security under a strong attacker model
- Portable in terms of multiple commodity platforms
  - We used AVR and MSP430
Security Goals

Establish a **dynamic root of trust** on the prover
- “ Guarantee un-tampered execution of a target piece of code, even in the presence of a corrupted platform”

In particular:
- Prover authentication
  - Are we talking with the right prover?
- External verification
  - Do we know the internal state of the prover?
- Guaranteed execution
  - Do we know execution state?

NOTE: No tamper resistance & no hardware attacks assumed

How do we achieve that?

With minimal cost =?= smallest set of features

Building Blocks

1. **Secure Key Storage** (as little as 180 bits)
   - Required for multi-hop (remote) prover settings
   - Enables prover authentication
2. **Trusted ROM code** memory region
   - Read-only means integrity: computes auth. token
   - Accesses/uses key (exclusively!)
3. **MCU access controls**
   - Grants access to key to ROM code
4. **Atomicity of ROM code execution**
   - Disable/enable interrupts
   - No invocation other than from the start

Key storage

- Facilitates remote prover authentication
- Key cannot be stored in “regular” memory
  - Malware would steal it
- Need to restrict key access

Our approach

- Only trusted ROM code region can access the key
Low-end embedded devices do not have support for rings to restrict access to memory.

Adding those would require significant complexity.

**Our approach:**
- Restrict key access to read-only trusted code region
- Access control can be implemented easily

Only SMART ROM code have access to key
- Control program counter value

Trusted ROM code and malware share the same resources
- Malware can set up the execution environment to compromise trusted code and extract the key
- Interrupts can asynchronously execute while a copy of the key is in main memory
- Malware can use code gadgets in ROM to access the key
  - Return-Oriented Programming (ROP)
- ROM code might leave traces of the key in memory after execution
Countermeasures

- Atomic ROM code execution
  - Enforced in hardware
  - Enter at first instruction
  - Exit at last instruction
- ROM code instrumented to check for memory safety
  - We used DEPUTEE
  - Upon detecting error reboot and clean memory
- Interrupts disabled immediately upon ROM entry
  - Before key usage
  - DINT instruction must itself be atomic
- Erase key-related material before end of execution

Stepping Back...

- SMART is a seemingly ad hoc set of features that seems to work...
- Can we justify it better?

Access Control Schematics

Costs of ROM and Access Control

- Implemented on two commodity low-end MCU platforms
  - AVR
  - MSP430

<table>
<thead>
<tr>
<th>Component</th>
<th>Original Size in kGE</th>
<th>Changed Size in kGE</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVR MCU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>103</td>
<td>113</td>
<td>10%</td>
</tr>
<tr>
<td>Sram</td>
<td>11.3</td>
<td>11.6</td>
<td>2.6%</td>
</tr>
<tr>
<td>Flash</td>
<td>26.6</td>
<td>26.6</td>
<td>0%</td>
</tr>
<tr>
<td>ROM</td>
<td>32 kB</td>
<td>65</td>
<td>0%</td>
</tr>
<tr>
<td>MSP430 MCU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>128</td>
<td>141</td>
<td>10%</td>
</tr>
<tr>
<td>Sram</td>
<td>7.6</td>
<td>8.3</td>
<td>9.2%</td>
</tr>
<tr>
<td>Flash</td>
<td>55.4</td>
<td>55.4</td>
<td>0%</td>
</tr>
<tr>
<td>ROM</td>
<td>32 kB</td>
<td>12.7</td>
<td></td>
</tr>
</tbody>
</table>
SMART Summary

For further information:


Future Work/Directions:

- Verifier Authentication
- Asymmetric vs Symmetric cryptography?
- Formal proof of security + Minimality proof
- SMART as a platform for more sophisticated/specialized services, e.g., code update, secure boot.
- More experiments/implementation

Summary (contd.)

- Architectural support – a must for provably secure attestation
- POSE: a minimalist approach to one-hop (non-remote) attestation
- SMART: efficient low-cost (minimal-feature) hardware attestation for remote embedded devices
- Low # of additional gates
- Virtually no run-time cost

Questions?