REMOTE ATTESTATION OF HETEROGENEOUS CYBER PHYSICAL SYSTEMS
(THE AUTOMOTIVE USE CASE)

KARIM ELDEFRAWY (HRL LABORATORIES)
GAVIN HOLLAND (HRL LABORATORIES)
KMELEFRAWY@HRL.COM, GHOLLAND@HRL.COM

GENE TSUDIK (UC IRVINE)
GTS@ICS.UCI.EDU

This talk is largely based on previous work by the authors and others in the following papers:
Overview of HRL

HRL Laboratories, LLC

- Formerly Hughes Research Laboratories (est. 1948)
- Formed as a Limited Liability Company (LLC), 1997
- R&D for The Boeing Company and General Motors
- Government and commercial contracts
- AS9100 accredited / DoD Trusted Foundry
- 250,000 square feet of lab space
- 10,000-square-foot Class 10 clean room
- Located on 72 acres in Malibu, CA

General Motors

- General Motors Corp. est. in 1908
- #2 in sales globally (7.5M vehicles in 2009)
- 200,000+ employees worldwide, 200+ facilities
- GM R&D: world’s first automotive research ctr.
- Milford Proving Grounds: industry’s first dedicated automobile testing facility
- Long history in new technologies and breakthrough innovations, dating back to 1920
- 1,123 US patents in 2011 alone
- 1st place team in DARPA Urban Challenge

Outline

• Introduction and Motivation

• Prelims for Remote Attestation

• Secure and Minimal Architecture for (Dynamic) Root of Trust (SMART)

• Future Directions
Widening Range of Specialized/Embedded Devices

- Smartphones and Watches
- SmartCards
- RFIDs
- Sensors and Actuators
- Connected Devices
- Automotive Systems
- Industrial Systems
- Smartphones and Watches
- SmartCards
- RFIDs
- Sensors and Actuators
- Connected Devices
- Automotive Systems
- Industrial Systems
Already Here or Coming Soon

- Smart watches, e.g., Apple, Samsung
- Smart glasses and personal (VR) displays, e.g., Google Glass, Oculus Rift, Samsung
- Smart footwear, e.g., Nike+
- Smart clothes and garments (outer and under)
Notable Attacks on Embedded Systems

- Stuxnet [1] (also DUQU)
  - Infected controlling windows machines
  - Changed parameters of the PLC (programmable logic controller) used in centrifuges of 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 cardiac defibrillator [4]
- Most effective CPS attacks are remote infestations, i.e., not physical attacks

Specialized/Embedded Devices in the Automotive Domain

According to sensormag.com

Market report on "Automotive Sensor Market by Product (Pressure, temperature, level, speed, MEMS, oxygen, Nox), Application (powertrain, safety & control, vehicle security, alternative fuel, telematics) and Geography - Forecast & Analysis to 2013 – 2022" is expecting market to grow at a **CAGR of 8.6% from 2014 to 2022** and reach $35.78 Billion in 2022.

"Modern cars have become complex digital devices, which can contain over 70 electronic control units (ECUs) …”
https://www.escar.info/escar-usa.html
“Given the diverse use cases inside the vehicle, it is reasonable to describe a vehicle as a composite industrial control system network with one or more Internet Gateways and one or more human user interfaces.” -- TCG TPM 2.0 Automotive Thin Profile, March 16th 2015
Here we show an example of message flows for a use case of remote maintenance of firmware, where an integrity digest is used to verify an ECU firmware update or patch. Because of this focus, details related only to real-time vehicle operations performed by ECUs (brakes, lights, engine, etc.) will be ignored.

Remote vehicle maintenance could be done periodically and/or in vehicle off times (i.e., vehicle parked with ignition off). When vehicle recalls could occur based only on software implementation defects, not caused by hardware issues, the remote vehicle maintenance method could be used to solve these recall issues without a dealer or repair shop visit.

Figure 4 shows the message flow for each component (Head Unit/Gateway or ECU) for remote maintenance handled by Automotive-Rich, and -Thins.

Figure source: TCG TPM 2.0 Automotive Thin Profile, March 2015
http://www.trustedcomputinggroup.org/resources/tcg_tpm_20_library_profile_for_automotivethin
Low-end Embedded Devices (Automotive-Thin in TCG Language)

- Memory: program (e.g., 128KB Flash) and data (e.g., 4KB SRAM)
- Typically built around an MCU (serving as CPU), Integrated clock
- As well as:
  - Communication interfaces (USB, CAN, Serial, Ethernet, etc.)
  - Analog to digital converters

- Examples: TI MSP430, Atmel AVR, Raspberry Pi
High-end Embedded Devices
(Automotive-Rich in TCG Language)

• Contrast with high-end processors, e.g., ARM, Intel

• Possibly built-in cryptographic support/functions, e.g., TPM, secure boot, HW-based isolation

• Notable example: ARM Trustzone

Figure sources: http://www.arm.com/products/processors/technologies/trustzone/index.php
Issues in a (Heterogeneous) CPS

• Unrealistic that every processor will be Trustzone-like, maybe (at most) 1 or 2 in the system

• Cost is a serious limitation

• “Trust Anchors” in a large CPS can be built on more powerful CPUs, they can attest other CPUs/MCUs

CPS Definition: “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”
Disclaimer: This Talk is …

- Not a final solution for securing heterogeneous CPS, more research still needed (also ongoing work to standardize it)

- A description of design and performance of an essential component for remote attestation for low-end (automotive-thin) embedded devices

- A blueprint for how the entire system can be attested

- Outline of future direction and research
Introduction and Motivation

Prelims for Remote Attestation

Secure and Minimal Architecture for (Dynamic) Root of Trust (SMART)

Future Directions
Remote Attestation Definitions

- Two party protocol between trusted verifier and untrusted prover
- Remotely verify the internal state of the prover

Internal state of prover is composed of: code, registers, data memory, i/o

Three types of attestation:
- Hardware-based: secure hardware supported (e.g., TPM)
- Software-based attestation: does not support multi-hop communication
- Hybrid: minimal hardware support and changes (this talk)
Malicious software will lie about the software state of the prover.

Need to have guarantees that the device is not lying.

Attestation protocol $P = (Setup, Attest, Verify)$:

- $k = Setup(1^k)$
  a setup procedure to generate a shared key

- $\alpha = Attest(k, s)$
  Key, Device state => Attestation token

- output = $Verify(k, s, \alpha)$
  Key, Expected state, Token => Yes/No
(1) Pick a random challenge

(2) State of prover + challenge + key

(3) Verify that the returned output corresponds to expected state

Attestation protocol may also return the exact state
Attestation of a Heterogeneous CPS

• Assume there are two types of devices:
  – High-end CPUs with TPM and hardware functionalities
  – Low-end CPUs without TPM (SMART could be a solution)
  – Otherwise: use SW-based attestation on Low-end (weak security guarantees)
Outline

• Introduction and Motivation

• Prelims for Remote Attestation

• Secure and Minimal Architecture for (Dynamic) Root of Trust (SMART)

• Future Directions
Systematic Analysis of Required Features

Remote Attestation

Prover Authentication

Authenticated Integrity of Prover’s Internal State

Verifier Challenge

Prover Secret Key

MAC function + helper code

Restricted Access Secret Key Storage

Atomic Execution

Non-malleable Code = ROM

Exclusive Access
Building Blocks

1. Secure Key Storage (as little as 180 bits)
   - Required for remote Prover
   - Enables Prover authentication

2. Trusted ROM code memory region
   - Read-only means integrity: computes response
   - Accesses/uses key (exclusively)

3. MCU access control
   - Grants access to key from within ROM code only

4. Atomicity of ROM code execution
   - Disable/enable interrupts
   - No invocation other than from the start
**Scope of Modifications to MCUs**

**AVR:** Dark gray boxes represent logic added to the processor. Core control signals provide information about internal processor status to memory bus controls.

**MSP430:** Memory backbone was modified to control access to ROM and key. MSP430 is based on Von Neumann architecture, concurrent access can occur to different memory parts (e.g., instruction fetch and read data). In that case, memory backbone arbitrates bus access and temporarily saves/restores data.

---

**Figure 4: Modifications to A VR and MSP430.**

<table>
<thead>
<tr>
<th>Component</th>
<th>A VR MCU</th>
<th>MSP430 MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>103</td>
<td>55.4</td>
</tr>
<tr>
<td>Flash</td>
<td>26.6</td>
<td>65</td>
</tr>
<tr>
<td>SRAM</td>
<td>11.6</td>
<td>8.3</td>
</tr>
<tr>
<td>Key ROM</td>
<td>65</td>
<td>55.4</td>
</tr>
<tr>
<td>SMART ROM</td>
<td>11.3</td>
<td>65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ratio</th>
<th>Orig.</th>
<th>SMART</th>
<th>Orig.</th>
<th>SMART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size in kGE</td>
<td>10%</td>
<td>0%</td>
<td>0%</td>
<td>10%</td>
</tr>
</tbody>
</table>

- **SMART:** The act of transforming (or compiling) the design from a high-level description language (Verilog or VHDL) into a set of wires and elementary gates that serve as building blocks of an Application-Specific Integrated Circuit (ASIC).

- **SMART:** For better performance, RAM and ROM memories were generated with a specific tool. Memory numbers were gleaned from publicly available information.

- **SMART:** Results can vary substantially depending on many parameters, such as: required maximum frequency, latency, placement and routing, and availability of better memory IP. However, our SMART framework to both A VR and MSP430 caused only a minimal increase in surface area compared to the original MCU. Synthesizing is not perform optimizations.

- **SMART:** Flash and SRAM are key components of the memory hierarchy. In that case, memory backbone arbitrates concurrent access can occur to different memory parts (e.g., instruction fetch and read data). In that case, memory backbone arbitrates bus access and temporarily saves/restores data.
The Complete SMART Protocol

Verifier

(nonce, target code)

Prover

HMAC result

(a = x, \( \mathcal{HC} \): Code to Attest)

\( \mathcal{RC} \): SMART ROM Code

Program Memory Address Space

User's Application Code

Data Memory Address Space

Registers/I0

Application Data Memory

HMAC Result

Stack Area

\( K \): Protected KEY

1. User application starts SMART
2. Code attestation is performed using the protected key
3. HMAC result is written to global memory, at a predefined location
4. C is executed (optional)

\[ \text{Data read/write Control flow} \]
Figure 3: Schematic view of access control for attestation key.

A1: Cryptographic checksum $C$ computed by $PRV$ cannot be forged. Since $C$ is a result of secure HMAC function (e.g., HMAC-SHA) we assume that, for any $ADV$ – external to $PRV$ – that observes a polynomial number of such checksums, finding HMAC collisions and/or learning bits of the attestation key is infeasible.

A2: Physical and hardware-based attacks on $PRV$ are beyond $ADV$'s capabilities.

A3: Attestation key $K$ can be accessed only from within ROM-resident SMART code. This is guaranteed by MCU-based access controls.

A4: SMART code cannot be modified since it resides in ROM.

A5: SMART code can be only invoked at its beginning. The hardware checks that, except for the very first instruction in $RC$, if the program counter is in $RC$ range, then the previous executed instruction must also be in ROM.

A6: $RC$ execution can only terminate at the very last instruction address in $RC$. The hardware checks that, except for the very last instruction in $RC$, if the program counter is not in $RC$ range, then the previous instructions must also be outside $RC$ range.

A7: Upon each invocation of SMART, all interrupts are disabled and remain so if, upon completion of SMART, control is passed to $HC$.

A8: $K$ cannot be extracted by any software-based $ADV$ internal to $PRV$. Upon completion of SMART execution, $K$ is no longer accessible. Also, all memory used by SMART code is securely erased. The only value based (statistically dependent) on $K$ is the output $C$.

A9: For each invocation, SMART computes $C$ based on the contents of the requested memory segment $[a, b]$. Although $C$ is guaranteed to be computed correctly, it may or may not result in $PRV$ passing attestations, since $[a, b]$ might be previously corrupted by $ADV$.

A10: Any erroneous state (e.g., violation of assertions A3, A5, A6) leads to a hardware reset. Upon reset, all data memory and registers are erased, which prevents $K$ leakage. This boot-time memory erase also guarantees that, if power loss occurs during SMART execution, no information about $K$ is retained in memory.

A11: Observing normal execution of SMART should leak no information about $K$. Therefore, SMART execution time and amount of memory used must not be key-dependent.
Design and Operation Issues

• If Prover infected, ROM code and malware share the same MCU resources
  
  – Malware can set up execution environment to compromise ROM code and extract key
  
  – Malware can schedule interrupts to occur asynchronously while key (or some function thereof) is in main memory
  
  – Malware can use code gadgets in ROM to access key
    – Return-Oriented Programming (ROP)
  
  – ROM code might leave traces of key in memory after its 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
  - Used DEPUTY
  - Upon detecting error reboot and clear memory

- Interrupts disabled immediately upon ROM entry
  - Before key usage (enabled upon exit)
  - DINT instruction must itself be atomic

- Erase key-related material before end of execution
Cost of Adding ROM and Access Control

Implemented on two commodity low-end MCU platforms (AVR and 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>103</td>
<td>113</td>
<td>10%</td>
</tr>
<tr>
<td>Core</td>
<td>11.3</td>
<td>11.6</td>
<td>2.6%</td>
</tr>
<tr>
<td>Sram</td>
<td>4 kB</td>
<td>26.6</td>
<td>0%</td>
</tr>
<tr>
<td>Flash</td>
<td>32 kB</td>
<td>65</td>
<td>0%</td>
</tr>
<tr>
<td>ROM</td>
<td>6 kB</td>
<td>10.3</td>
<td>-</td>
</tr>
<tr>
<td>MSP430 MCU</td>
<td>128</td>
<td>141</td>
<td>10%</td>
</tr>
<tr>
<td>Core</td>
<td>7.6</td>
<td>8.3</td>
<td>9.2%</td>
</tr>
<tr>
<td>Sram</td>
<td>10 kB</td>
<td>55.4</td>
<td>0%</td>
</tr>
<tr>
<td>Flash</td>
<td>32 kB</td>
<td>65</td>
<td>0%</td>
</tr>
<tr>
<td>ROM</td>
<td>4 kB</td>
<td>12.7</td>
<td>-</td>
</tr>
</tbody>
</table>

Comparison of chip surface required by each component of original MCU to SMART-modified version. kGE stands for thousands of Gate Equivalents (GE-s). One GE is proportional to the surface of the chip and computed from the module surface divided by the surface of a NAND2 gate, $9.37 \times 10^{-6}$ mm$^2$ with this library.
HMAC Performance and Effort to Implement

HMAC is the most expensive operation to perform attestation in SMART

<table>
<thead>
<tr>
<th>Data Size</th>
<th>Cycles</th>
<th>Time at 8MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KByte</td>
<td>2302281</td>
<td>287 ms</td>
</tr>
<tr>
<td>512 Bytes</td>
<td>1281049</td>
<td>160 ms</td>
</tr>
<tr>
<td>32 Bytes</td>
<td>387471</td>
<td>48 ms</td>
</tr>
</tbody>
</table>

Changes made (in # of HDL lines of code) in AVR and MSP430 processors, respectively, excluding comments and blank lines.

<table>
<thead>
<tr>
<th>Component</th>
<th>Original</th>
<th>Changed</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lines</td>
<td>Lines</td>
<td>Ratio</td>
</tr>
<tr>
<td>AVR, core (VHDL)</td>
<td>3932</td>
<td>151</td>
<td>3.84%</td>
</tr>
<tr>
<td>AVR, tests</td>
<td>2244</td>
<td>760</td>
<td></td>
</tr>
<tr>
<td>MSP430, core (Verilog)</td>
<td>4593</td>
<td>182</td>
<td>3.96%</td>
</tr>
<tr>
<td>MSP430, tests</td>
<td>17665</td>
<td>1122</td>
<td></td>
</tr>
</tbody>
</table>
Secure Remote Attestation for Low-end Devices

• Introduction and Motivation

• Prelims for Remote Attestation

• Secure and Minimal Architecture for (Dynamic) Root of Trust (SMART)

• Future Directions
Open Issues and Research Directions

- Asymmetric vs symmetric cryptography on Prover

- Automated synthesis of attestation and formal verification of implementation

- Platform for more sophisticated or specialized services: secure code update, secure erasure, secure boot

- More experiments and implementation on various platforms/CPS

- Verifier Authentication (very relevant to mitigate denial-of-service)
Questions?
SMART as an API

Algorithm  SMART usage to attest a memory range.

<table>
<thead>
<tr>
<th>input</th>
<th>n nonce sent by ( \mathcal{RF} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( a ) start address to attest</td>
</tr>
<tr>
<td></td>
<td>( b ) end address to attest</td>
</tr>
<tr>
<td></td>
<td>( H ) HMAC result (global variable)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output: HMAC output</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMART ((a, b, \emptyset, False, n, &amp;H, \emptyset));</td>
</tr>
<tr>
<td>Send(H);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>end</th>
<th>challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory range to attest</td>
<td></td>
</tr>
</tbody>
</table>
For More Details


A. Francillon, Q. Nguyen, K. Rasmussen and G. Tsudik, A Minimalist Approach to Remote Attestation, DATE 2014,
• full version in Crypto ePrint Archive: Report 2012/713.