ACSAC'15
Los Angeles, California, USA

Soteria: Offline Software Protection within Low-cost Embedded Devices

Johannes Götzfried*, Tilo Müller*, Ruan de Clercq†, Pieter Maene†,
Felix Freiling*, and Ingrid Verbauwhede†

*Department of Computer Science
FAU Erlangen-Nuremberg, Germany

†COSIC, Department of Electrical Engineering (ESAT)
KU Leuven, Belgium

December 10, 2015

Outline

State-of-the-Art Software Protection

Mostly based on *Obfuscation*

- Transformations making programs harder to analyze
- Some programs provably *can* be obfuscated (e.g. Password Checks)
- Some programs provably *cannot* be obfuscated (e.g. Quines)

$\rightarrow$ In general: Obfuscation only increases the time needed for analysis

State-of-the-Art Software Protection

Mostly based on *Obfuscation*

- Transformations making programs harder to analyze
- Some programs provably *can* be obfuscated (e.g. Password Checks)
- Some programs provably *cannot* be obfuscated (e.g. Quines)

$\rightarrow$ In general: Obfuscation only increases the time needed for analysis

Software Protection for Embedded Devices:
Attackers with clear economic motivations

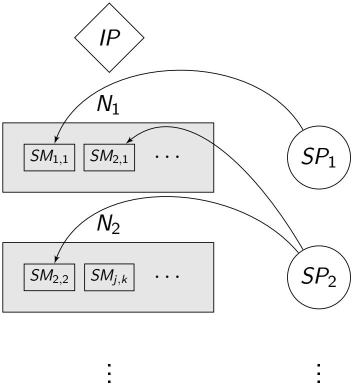
- Customizers tampering with data
  Example: Amount of consumed energy measured by smart meters
- Competing industrial entities analysing software
  Example: Re-engineering of a competitive product

Outline

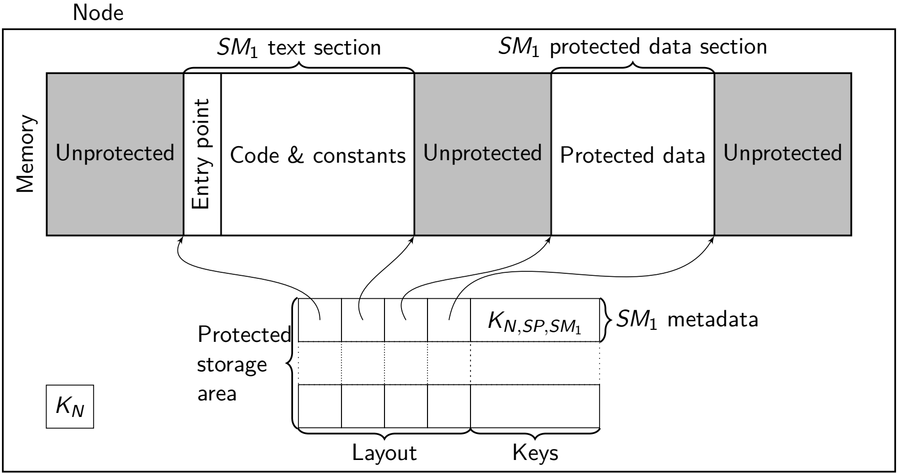
Sancus: System Overview

Low-cost extensible security architecture

- Strict isolation of software modules
- Secure communication and attestation
- Zero-software trusted computing base



Sancus: Software Modules



Sancus: Design Details

- Program-Counter based access control

From/To	Entry	Text	Protected	Unprotected
Entry	`r-x`	`r-x`	`rw-`	`rwx`
Text	`r-x`	`r-x`	`rw-`	`rwx`
Unprotected/ Other SM	`r-x`	`r--`	`---`	`rwx`

- Isolation can be enabled/disabled with dedicated new instructions
  - `protect` *layout*,*SP*
  - `unprotect`
- Hierarchical key derivation
  - $K_{N,SP} = kdf(K_N, SP)$ [based on *SP* ID]
  - $K_{N,SP,SM} = kdf(K_{N,SP}, SM)$ [based on *SM* identity]
- Shared secret between *SM* on *N* and *SP*: $K_{N,SP,SM}$
  - Can be used for remote attestation with an HMAC

Outline

Attacker Model

Not within our attacker model

- No DoS protection
- No hardware attacks
  - RAM dumping
  - Chip probing

Within our attacker model

- Control of all peripheral components
- Control of all software components
  - Including high-privilege software components, e.g., OS

Basic Idea: Offline SW-Protection

→ We want: Offline SW-Protection

- Problem: SMs are able to access each others text section (`r--`)

From/To	Entry	Text	Protected	Unprotected
Entry	`r-x`	`r-x`	`rw-`	`rwx`
Text	`r-x`	`r-x`	`rw-`	`rwx`
Unprotected/ Other SM	`r-x`	`---`	`---`	`rwx`

Design of Soteria

Problem: Code resides unencrypted within ROM

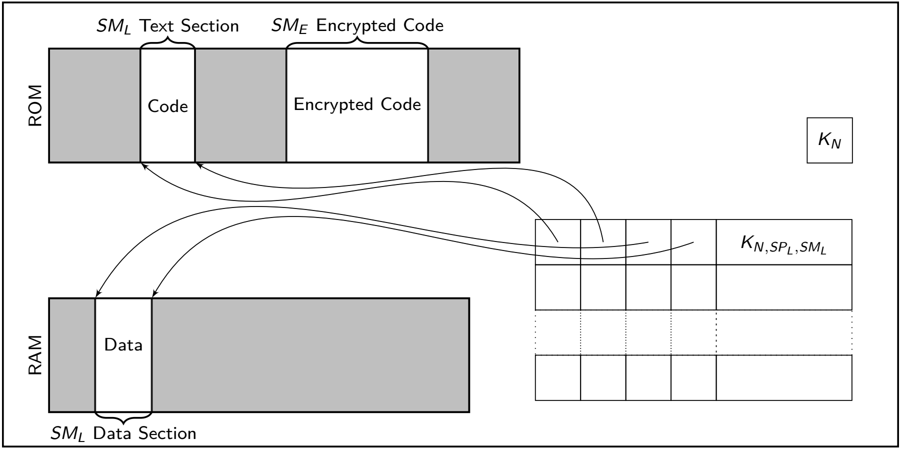
- Encrypt Code within ROM
- Decrypt Code to RAM just before SM loading

Loading Process

1. Loader SM_L derives $K_{N,SP_L,SM_L,SM_E} = E_{SM_E} = kdf(K_{N,SP_L,SM_L}, \widetilde{SM_E})$
2. Loader SM_L decrypts SM_E with E_{SM_E} and calls `protect`
   - SM_L uses authenticated encryption (AES-128 in CCM mode of operation)
   - Decryption and `protect` is done atomically
3. SM_L is able to load the next encrypted module or to unprotect itself

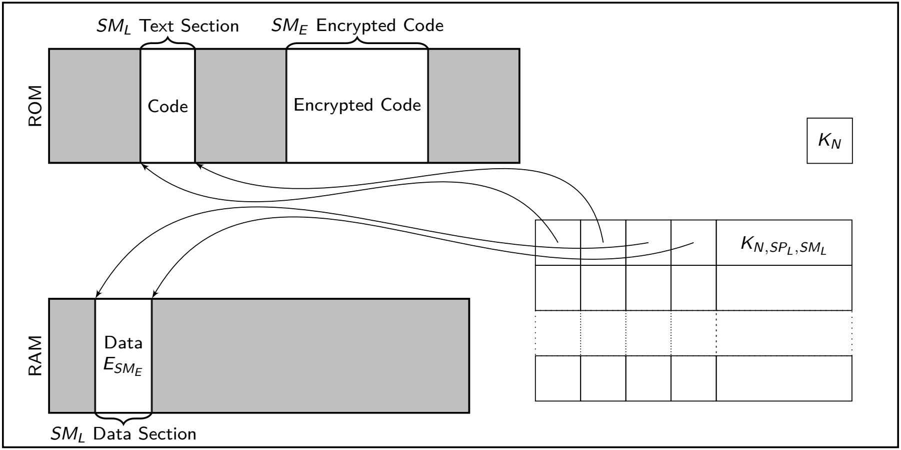
Loading Steps of a Module

Initial situation: SM_L is active and SM_E is encrypted



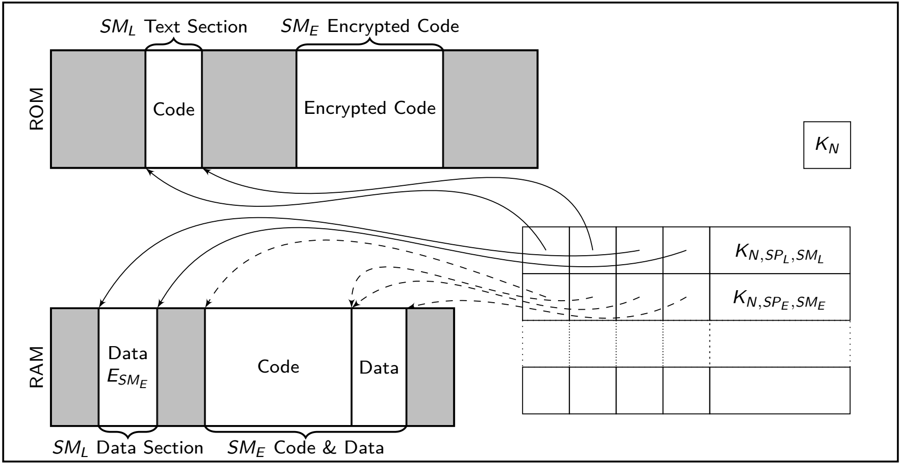
Loading Steps of a Module

1. Loader SM_L derives E_{SM_E}



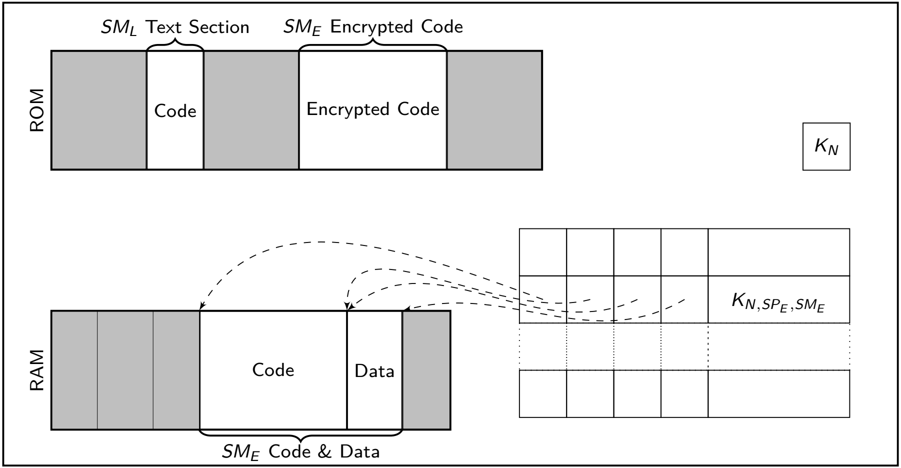
Loading Steps of a Module

2. SM_E gets decrypted to RAM and protected



Loading Steps of a Module

3. SM_L wipes data section and calls `unprotect`



Security Argument

- Before Loading: SM_E is encrypted within ROM (or RAM)
- After Loading: SM_E is protected by MAL
- If SM_L is tampered with:
  - E_{SM_E} is not derived correctly
    $\rightarrow$ authenticated decryption fails
- If SM_E is tampered with (before loading):
  - Integrity property is violated
    $\rightarrow$ authenticated decryption fails
- If a reset is triggered:
  - RAM is wiped
    $\rightarrow$ no decrypted fragments of SM_E can be found

Outline

Implementation Details

Hardware Implementation

- Based on the openMSP430 project from OpenCores
- Patched the OMSP430 to get RAM executable
- Patched the Sancus MAL to prevent read access to other modules
- Included memory wipe after reset
- Successfully tested on the XC6VLX240T Virtex-6 FPGA (UART and GPIO)

Implementation Details

Hardware Implementation

- Based on the openMSP430 project from OpenCores
- Patched the OMSP430 to get RAM executable
- Patched the Sancus MAL to prevent read access to other modules
- Included memory wipe after reset
- Successfully tested on the XC6VLX240T Virtex-6 FPGA (UART and GPIO)

Software Implementation

- Library supporting encrypted modules
- Fully compatible to existing modules
- Implementation of SM_L

Implementation Details

Hardware Implementation

- Based on the openMSP430 project from OpenCores
- Patched the OMSP430 to get RAM executable
- Patched the Sancus MAL to prevent read access to other modules
- Included memory wipe after reset
- Successfully tested on the XC6VLX240T Virtex-6 FPGA (UART and GPIO)

Software Implementation

- Library supporting encrypted modules
- Fully compatible to existing modules
- Implementation of SM_L

Toolchain Modifications

- Automatically identify encrypted modules
- Transparently encrypt them (authenticated encryption)
- Host software is not part of the TCB
- Based on LLVM and pyelftools

Encryption Details

AES-128 in CCM mode of operation:

- According to RFC 3610
- Authentication tag length of sixteen bytes
- Two bytes length field
  $\rightarrow$ Maximum SM size of 64 kilobytes
- No associated data
- Thirteen bytes nonce: $\widetilde{SM_E}$ (zero padded)
  $\rightarrow$ Unique identifier $\widetilde{SM_E}$: Name + current version of SM_E

Outline

Area and Power

Evaluation on XC6VLX240T Virtex-6 FPGA with core running at 20Mhz:

- Plain openMSP430 core: 1,146 slice regs and 2,520 LUTs
- Overhead of Soteria compared to Sancus

	Sancus		Soteria		Overhead	
	REGs	LUTs	REGs	LUTs	REGs	LUTs
1 SM	1,897	3,686	1,938	3,894	41	208
2 SMs	2,110	4,100	2,150	4,322	40	222
3 SMs	2,323	4,378	2,363	4,620	40	242
4 SMs	2,536	4,778	2,576	5,034	40	256

- Power overhead of Soteria compared to Sancus: 0.2%

Performance

- No additional performance overhead once an application is running
- Constant overhead for resetting: $2 + \text{DRAM\_SIZE}/2$ cycles
- Constant overhead for protecting the loader: 72,976 cycles
- Constant overhead for destroying the loader: 800 cycles
- Overhead for loading software modules of different sizes:

Size (bytes)	Total Time (cycles / ms)
208	424,312 (21.216)
256	507,536 (25.377)
512	951,464 (47.573)
768	1,395,384 (69.769)
1024	1,839,304 (91.965)

- Implementation of AES-128 in CCM mode has been tweaked for size
  - $\approx$ 2 kilobytes of ROM
  - $\approx$ 200 bytes of RAM

Outline

Conclusion

Soteria as a software protection solution

- Zero-software trusted computing base
- Soteria allows *offline* software protection
- Confidentiality of code and data *before* and after loading

Soteria is lightweight

- Loader module only needs 200 bytes of RAM (AES)
- Only very little area and power overhead
- No additional performance overhead during runtime

Thank you for your attention!

Further Information:

https://www1.cs.fau.de/soteria