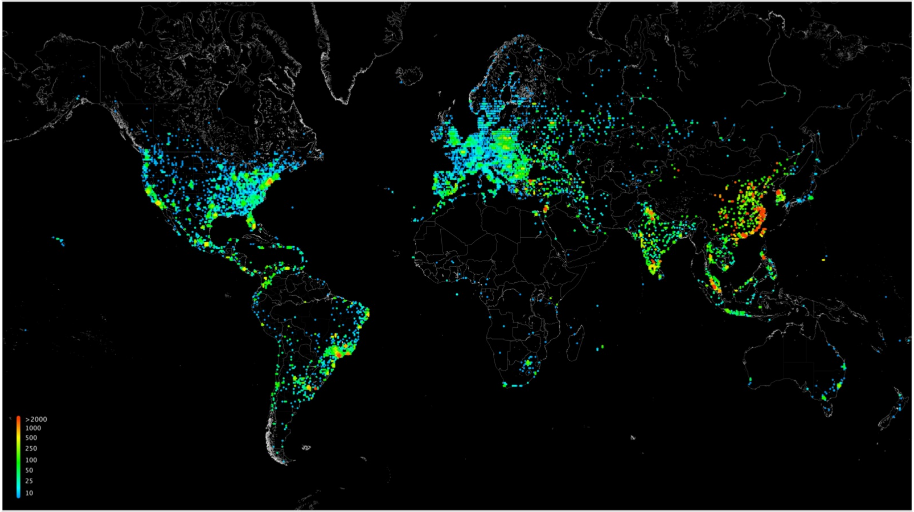
Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base

Job Noorman Pieter Agten Wilfried Daniels Raoul Strackx
Anthony Van Herrewege Christophe Huygens Bart Preneel
Ingrid Verbauwhede Frank Piessens

16 Aug 2013

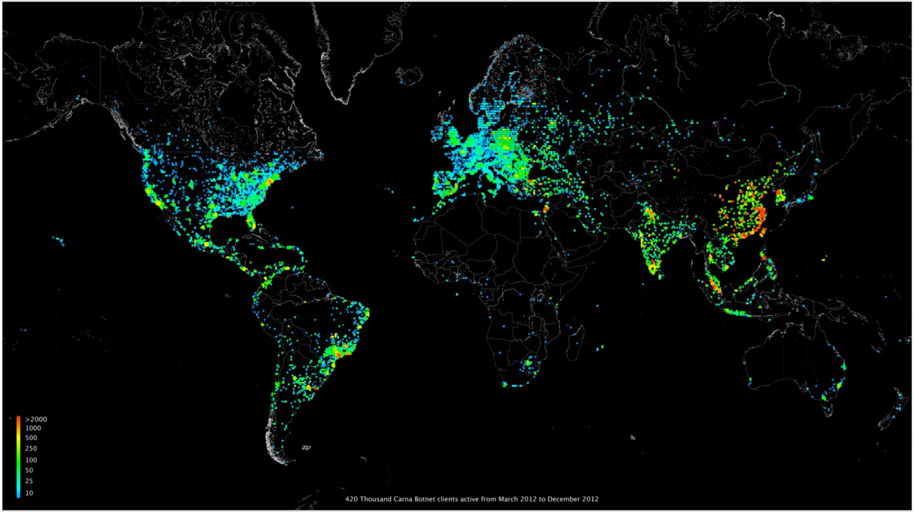
KU LEUVEN
DISTRINET RESEARCH GROUP

iMinds
CONNECT.INNOVATE.CREATE


>2000
1000
500
250
100
50
25
10

Carna Botnet

Port scanning /0 using insecure embedded devices (Anonymous researcher)



Carna Botnet client distribution March to December 2012. ∼420K Clients

Although very relevant, low-end devices lack effective security features

More threats on embedded devices

Due to network connectivity and third-party extensibility

No effective solutions exist

It's "a mess" (Viega and Thompson)

Researchers are exploring this area

E.g., SMART (El Defrawy et al.)

Goal: design and implement a low-cost, extensible security architecture

Strong isolation of software modules

Given third-party extensibility

Secure communication and attestation

Both locally and remotely

Counteracting attackers with *full* control over infrastructural software

Zero-software Trusted Computing Base

Target: a generic system model

Infrastructure provider

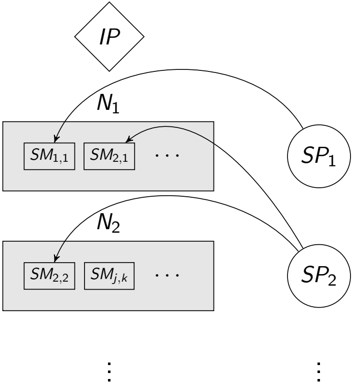
IP owns and administers nodes N_i

Software providers

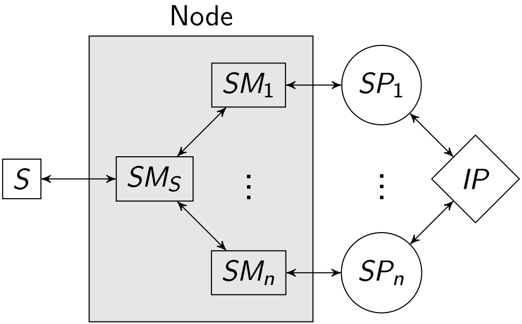
SP_j wants to use the insfrastructure

Software modules

$SM_{j,k}$ is deployed by SP_j on N_i



Example node configuration



Preview

1. Module isolation
2. Key management
3. Remote attestation and secure communication
4. Secure linking
5. Results

Overview

Modules are bipartite with a *public* text section and a *protected* data section

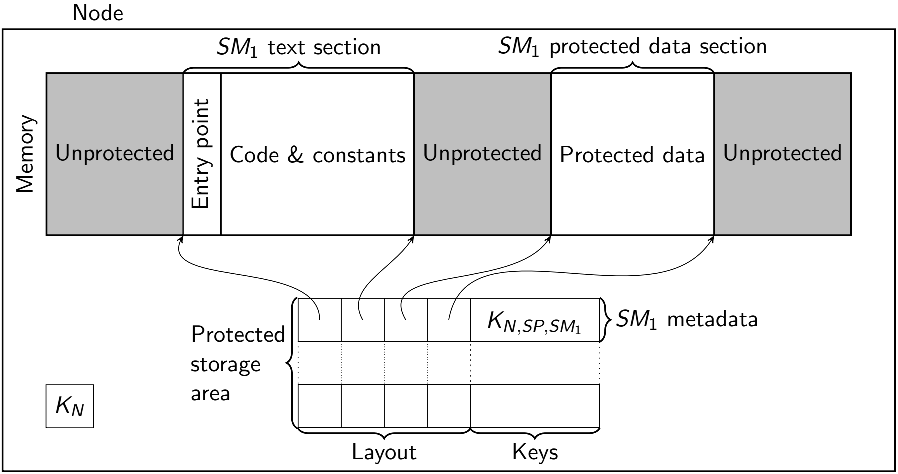
Public text section

Containing code and constants

Protected data section

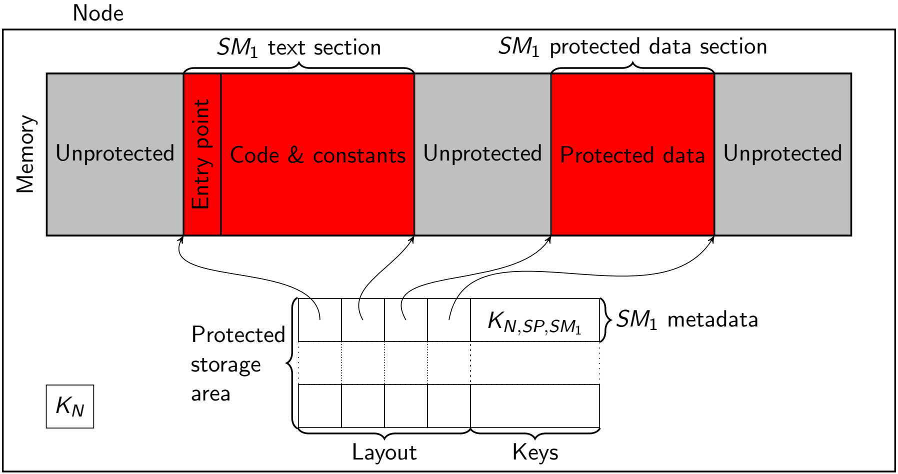
Containing secret runtime data

Node with one software module loaded



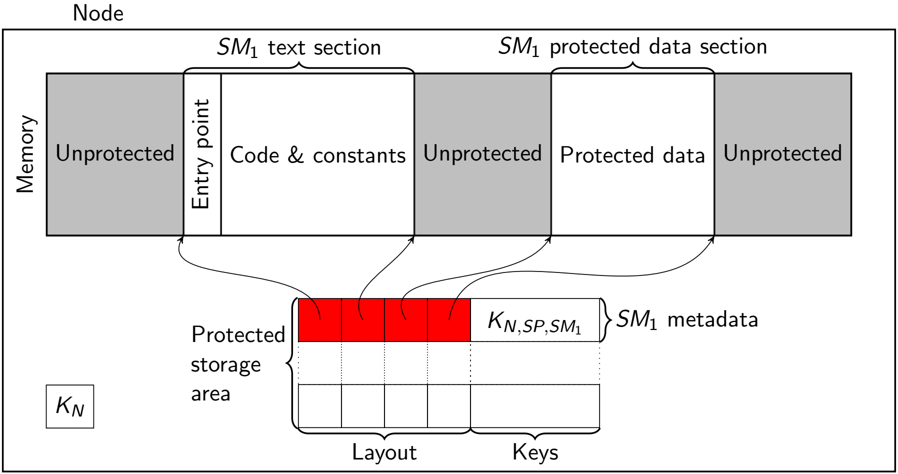
Node with one software module loaded

Public and protected sections



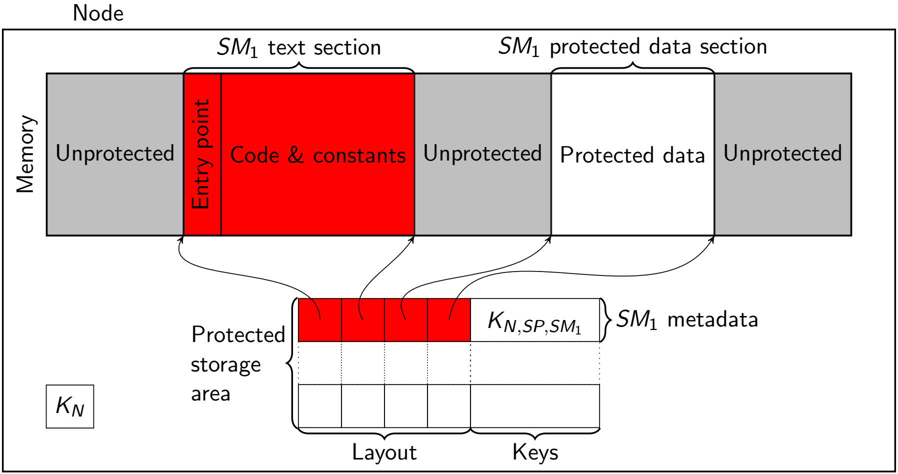
Node with one software module loaded

Module layout



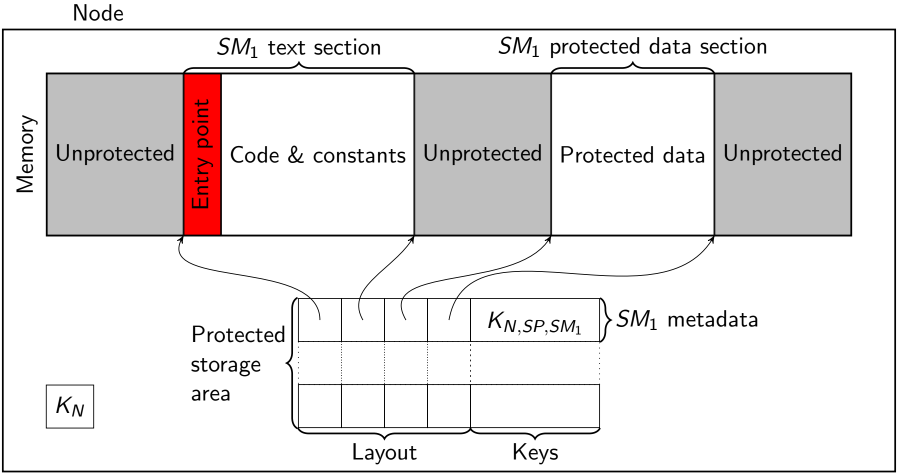
Node with one software module loaded

Module identity



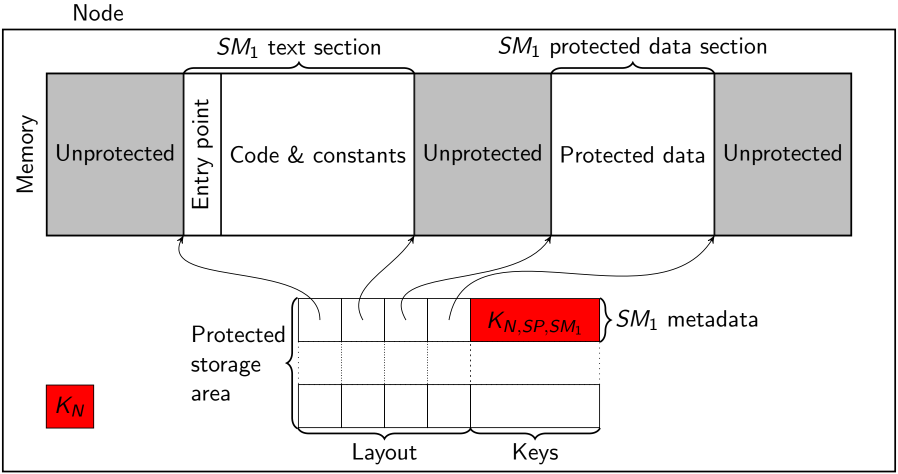
Node with one software module loaded

Module entry point



Node with one software module loaded

Module keys



Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

From/to	Text	Protected	Unprotected
Text			
Other			

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

From/to	Text	Protected	Unprotected
Text			
Other			

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

From/to	Text	Protected	Unprotected
Text			
Other			

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

From/to	Text	Protected	Unprotected
Text		rw-	
Other		---	

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

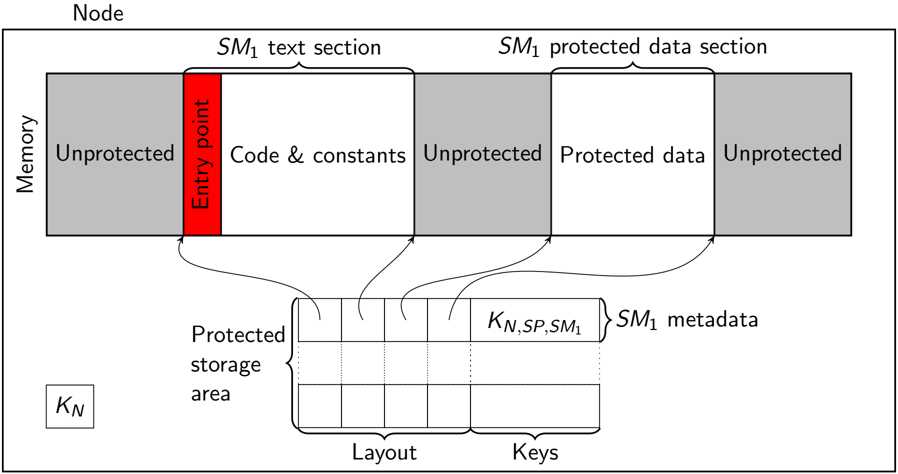
Only accessible from text section

Protection against code misuse (e.g., ROP)

From/to	Text	Protected	Unprotected
Text	r-x	rw-	
Other	r--	---	

Node with one software module loaded

Module entry point



Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

Protection against code misuse (e.g., ROP)

Enter module through single entry point

From/to	Text	Protected	Unprotected
Entry	r-x	rw-	
Text	r-x	rw-	
Other	r--	---	

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

Protection against code misuse (e.g., ROP)

Enter module through single entry point

From/to	Entry	Text	Protected	Unprotected
Entry	`r-x`	`r-x`	`rw-`	
Text	`r-x`	`r-x`	`rw-`	
Other	`r-x`	`r--`	`---`	

Modules are isolated using *program-counter based memory access control*

Variable access rights

Depending on the current program counter

Isolation of data

Only accessible from text section

Protection against code misuse (e.g., ROP)

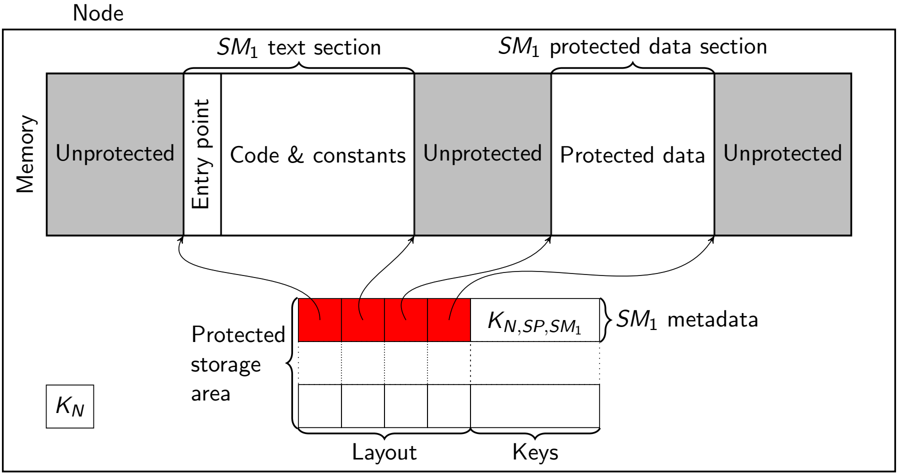
Enter module through single entry point

From/to	Entry	Text	Protected	Unprotected
Entry	r-x	r-x	rw-	rwx
Text	r-x	r-x	rw-	rwx
Other	r-x	r--	---	rwx

Isolation can be enabled/disabled using new instructions

Node with one software module loaded

Module layout



Isolation can be enabled/disabled using new instructions

`protect` *layout, SP*

Enables isolation at *layout*

`unprotect`

Disables isolation of current SM

Overview

Providing a flexible, inexpensive way for secure communication

Establish a shared secret

Between *SP* and its module *SM*

Use symmetric crypto

Public-key is too expensive for low-cost nodes

Ability to deploy modules without *IP* intervening

After initial registration, that is

Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

Able to derive all keys

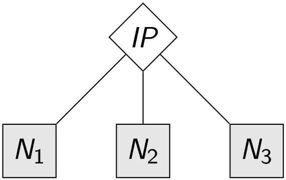
Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

Able to derive all keys

Every node N stores a key K_N

Generated at random



Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

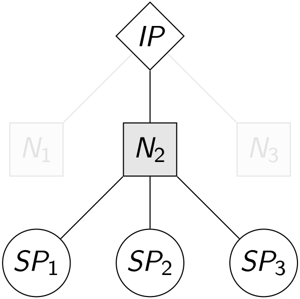
Able to derive all keys

Every node N stores a key K_N

Generated at random

Derived key based on *SP* ID

$K_{SP} = kdf(K_N, SP)$



Key derivation scheme allowing both Sancus and *SP*'s to get the same key

Infrastructure provider is trusted party

Able to derive all keys

Every node N stores a key K_N

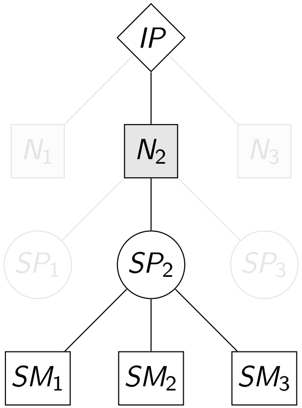
Generated at random

Derived key based on SP ID

$K_{SP} = kdf(K_N, SP)$

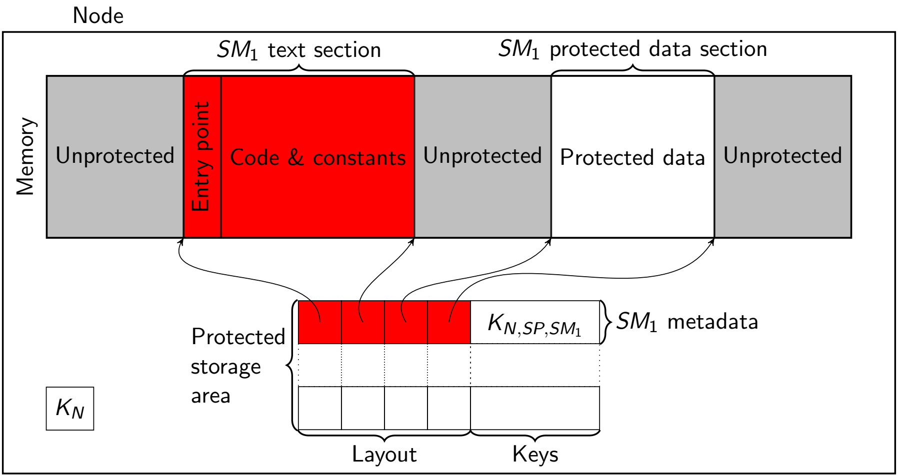
Derived key based on SM identity

$K_{SM} = kdf(K_{SP}, SM)$



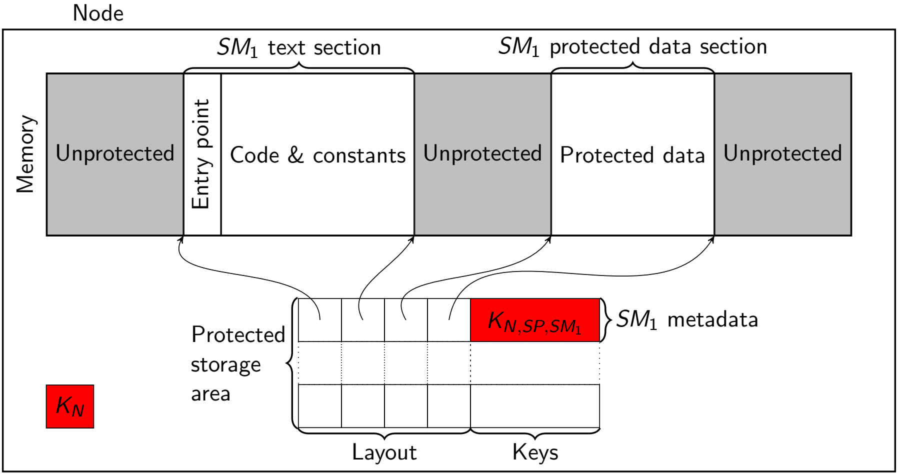
Node with one software module loaded

Module identity



Node with one software module loaded

Module keys



Isolation can be enabled/disabled using new instructions

`protect` *layout*, *SP*

Enables isolation at *layout* and calculates $K_{N,SP,SM}$

`unprotect`

Disables isolation of current SM

Overview

Ability to use $K_{N,SP,SM}$ proves the integrity and isolation of SM deployed by SP on N

Only N and SP can calculate $K_{N,SP,SM}$

N knows K_N and SP knows K_{SP}

$K_{N,SP,SM}$ is calculated *after* enabling isolation

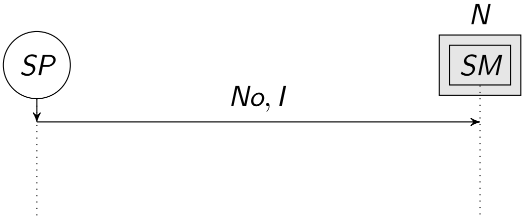
No isolation, no key; no integrity, wrong key

Only SM on N is allowed to use $K_{N,SP,SM}$

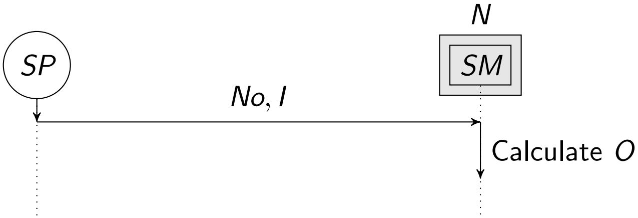
Enforced through special instructions

Secure communication is provided by calculating MACs using the module key

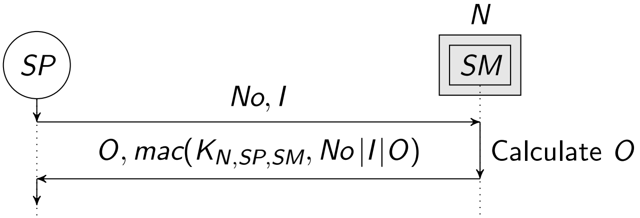
Secure communication is provided by calculating MACs using the module key



Secure communication is provided by calculating MACs using the module key



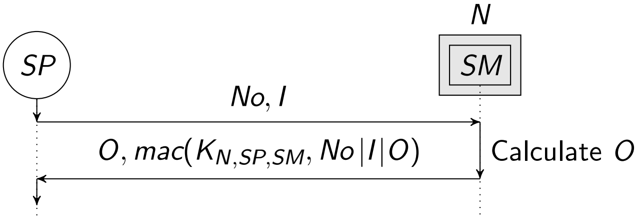
Secure communication is provided by calculating MACs using the module key



MAC is calculated by a `mac-seal` instruction

Using the key of the calling *SM*

Secure communication is provided by calculating MACs using the module key



MAC is calculated by a `mac-seal` instruction

Using the key of the calling *SM*

MAC can be recalculated by *SP*...

He knows the *correct* $K_{N,SP,SM}$

Ability to use $K_{N,SP,SM}$ proves the integrity and isolation of SM deployed by SP on N

Only N and SP can calculate $K_{N,SP,SM}$

N knows K_N and SP knows K_{SP}

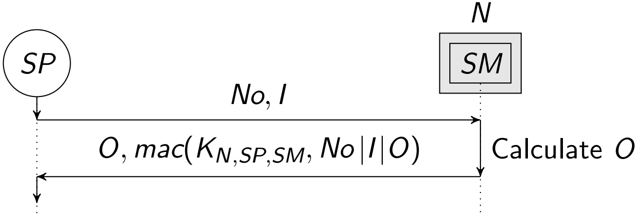
$K_{N,SP,SM}$ is calculated *after* enabling isolation

No isolation, no key; no integrity, wrong key

Only SM on N is allowed to use $K_{N,SP,SM}$

Enforced through special instructions

Secure communication is provided by calculating MACs using the module key



MAC is calculated by a `mac-seal` instruction

Using the key of the calling *SM*

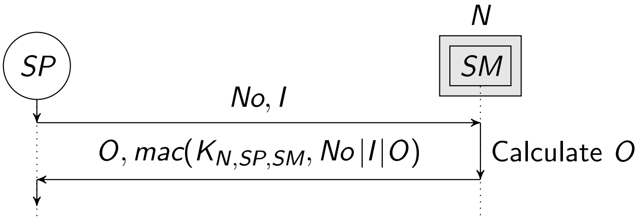
MAC can be recalculated by *SP*. . .

He knows the *correct* $K_{N,SP,SM}$

. . . providing trust in the authenticity of messages

Only *SM* can create the correct MAC

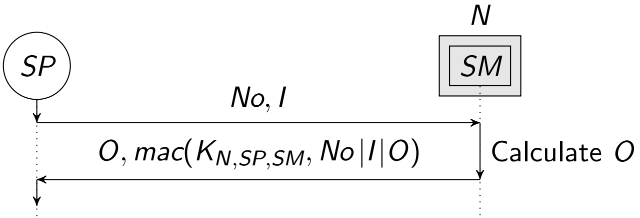
Remote attestation is provided through secure communication



Attest integrity, isolation and liveliness

Of *SM* by *SP*

Remote attestation is provided through secure communication



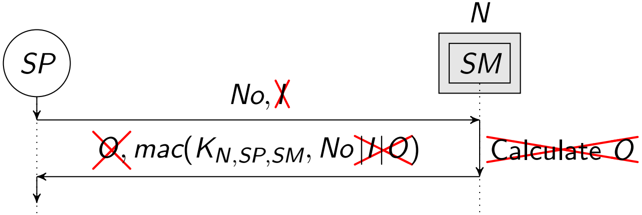
Attest integrity, isolation and liveliness

Of *SM* by *SP*

Integrity and isolation attested by MAC, liveliness by nonce

Thus included in secure communication

Remote attestation is provided through secure communication



Attest integrity, isolation and liveliness

Of *SM* by *SP*

Integrity and isolation attested by MAC, liveliness by nonce

Thus included in secure communication

$\Rightarrow$ remote attestation $\subset$ secure communication

So can be achieved more easily

Overview

1. Module isolation
2. Key management
3. Remote attestation and secure communication
4. Secure linking
   - Goals
   - Verifying modules
   - Optimizing multiple calls
5. Results

Enabling efficient and secure local inter-module function calls

Verify the *SM* that is to be called

Is it the correct, isolated *SM*?

Inherently different from secure communication

May belong to different *SP*s; no shared secret

We can rely on protected local state

Gives rise to interesting optimizations

Modules are verified by calculating a MAC over their identity

Module A wants to call module B

A is deployed with a MAC of B's identity using A's key

In an unprotected section since it is unforgeable

Modules are verified by calculating a MAC over their identity

Module A wants to call module B

A is deployed with a MAC of B's identity using A's key

In an unprotected section since it is unforgeable

A calculates the MAC of B's *actual* identity

If they match B can safely be called

Modules are verified by calculating a MAC over their identity

Module A wants to call module B

A is deployed with a MAC of B's identity using A's key

In an unprotected section since it is unforgeable

A calculates the MAC of B's *actual* identity

If they match B can safely be called

Done through new instruction: `mac-verify`

Need ensurance on B's isolation

The expensive MAC calculation is needed only once

We only need to know if the same module is still there

After initial verification, that is

The expensive MAC calculation is needed only once

We only need to know if the same module is still there

After initial verification, that is

Sancus assigns unique IDs to modules

Never reused within a boot-cycle

`mac-verify` returns the ID of the verified module

Can be stored in the protected section

Later calls can use a new instruction: `get-id`

Check if the same module is still loaded

Overview

Complete implementation of Sancus based on the MSP430 architecture

Based on the openMSP430 project

Very mature open-source MSP430 implementation

Built on existing primitives:

- MAC: HMAC
- KDF: HKDF
- Hashing: SPONGENT-128/128/8 (Bogdanov et al.)

Usable in RTL simulator and FPGA

For easy testability of Sancus

Automatically handling the intricacies of compiling Sancus modules

Placing the runtime stack in the protected section

Prevent access by untrusted code

Clearing registers on module exit

Prevent data leakage

Supporting more than one entry point

Dispatching through a single entry point

Automatically handling the intricacies of compiling Sancus modules

```
#include <sancus/sm_support.h>
#define ID "foo"

int  SM_DATA(ID)  protected_data;
void SM_FUNC(ID)  internal_function() {/*...*/}
void SM_ENTRY(ID) entry_point() {/*...*/}
```

No runtime overhead on "normal" code; moderate overhead given enough computation

No impact on maximum frequency

Critical path not affected

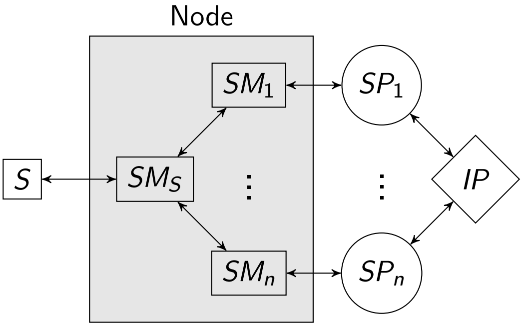
Main overhead from calculating MACs

For verification and output

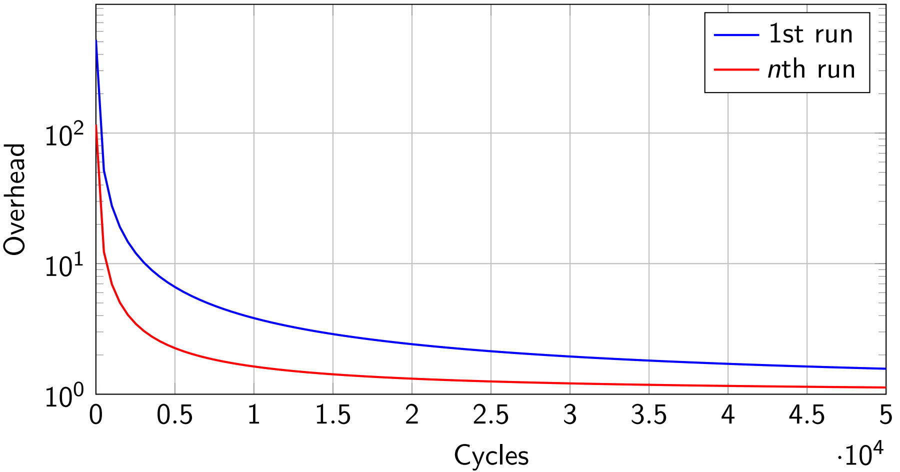
Smaller overhead from entry and exit code

Stack switching, register clearing,. . .

Example node configuration



No runtime overhead on “normal” code; moderate overhead given enough computation



Area overhead

Fixed overhead: 586 registers / $1,138$ LUTs
Mainly MAC and KDF

Per module: 213 registers / 307 LUTs
Mainly key storage

Review

1. Module isolation
   Isolation using *program-counter based access control*
2. Key management
   Hierarchical scheme with keys based on module's *identity*
3. Remote attestation and secure communication
   Attestation based on the ability to use a key
4. Secure linking
   Module verification based on MAC of its identity
5. Results
   Simulator, FPGA and automatic compilation

Sancus: Low-cost trustworthy extensible networked devices with a zero-software Trusted Computing Base

Job Noorman Pieter Agten Wilfried Daniels Raoul Strackx
Anthony Van Herrewege Christophe Huygens Bart Preneel
Ingrid Verbauwhede Frank Piessens

https://distrinet.cs.kuleuven.be/software/sancus/

KU LEUVEN
DISTRINET RESEARCH GROUP

iMinds
CONNECT.INNOVATE.CREATE