ARM TrustZone for ARMv8-M for software engineers

Ashok Bhat
Product Manager, HPC and Server tools

ARM Tech Symposia India
December 7th 2016

©ARM 2016
The need for security

- Communication protection
  - Cryptography, authentication

- Data protection
  - Secret data (keys, personal information)

- Firmware protection
  - IP theft, reverse engineering

- Operation protection
  - Maintaining service and revenue

- Anti-tamper protection
  - Related to all other protections
Potential security threats

- Device access via communication channel
- Random communication traffic
- Device access via debug port
- Malicious firmware updates

Connection

- Potential security threats
- Malicious firmware updates
- Device access via debug port
- Random communication traffic
- Device access via communication channel
ARM TrustZone for ARMv8-M

Security foundation in hardware
ARMv6-M, ARMv7-M, and ARMv8-M architecture

Scalable architecture for microcontrollers

ARMv8-M baseline (Cortex-M23)
- Lowest cost and smallest implementations

ARMv8-M mainline (Cortex-M33)
- For general purpose microcontroller products
- Highly scalable
- Optional DSP and floating-point extensions
TrustZone for ARMv8-A

Non-secure states

- Rich OS, e.g. Linux
- Secure app/libs
- Secure OS

Secure states

Secure monitor

TrustZone for ARMv8-M

Non-secure states

- Secure app
- Non-secure app
- Secure OS

Secure states

- Secure app/libs
- Non-secure OS

Secure transitions handled by the processor to maintain embedded class latency
Security defined by memory map

All transactions from core and debugger are checked

- All addresses are either secure or non-secure

- Policing managed by Secure Attribution Unit (SAU)
  - Internal SAU similar to MPU
  - Supports use of external system-level definition
  - For example, based on flash blocks or per peripheral

- Banked MPU configuration
  - Independent memory protection per security state

- Load/stores acquire non-secure (NS) attribute based on address
  - Non-secure access to secure address → memory fault
ARMv8-M additional states

Existing handler and thread modes are mirrored

- Secure and non-secure code runs on a single CPU
  - For efficient embedded implementation

- Secure state for trusted code
  - New secure stack pointers for robust operation
  - Addition of stack-limit checking

- Dedicated resources for isolation between domains
  - Separate memory protection units for secure and non-secure
  - Private SysTick timer for each state

- Secure side can configure target domain of interrupts
High performance cross-domain calls

Efficient implementation focused on microcontroller

- Security inferred from instruction address
  - Secure memory considered to hold secure code

- Direct function calls across boundary
  - High performance and high security
  - Multiple entry points
  - No need to go via ‘monitor’ for transitions

- Uses Secure Gateway (SG) instruction
  - Only permitted in special secure memory with non-secure callable (NSC) attribute
ARMv8-M programmer’s model: Memory map

Non-secure state

Non-secure memory view is identical with Cortex-M

Branches to fixed memory locations access secure firmware

Secure memory is invisible

Vector table for Non-secure handlers
Secure memory view shows additional Flash, RAM, and peripherals

Access to all regions is possible in secure state

Regions can be configured in secure state using the SAU

Vector table for secure handlers
A simplified use case

Composing a system with secure and non-secure projects

- **Non-secure** projects cannot access secure resources
- **Secure** project can access everything
- **Secure and non-secure** projects may implement independent time scheduling
Software development tools and software components

Accelerate software creation for ARMv8-M devices with TrustZone
Tools and components for software development

- Keil MDK IDE & debugger
- ARM Compiler 6
- CMSIS v5
- Fast Models
- ULINK debug adapters
- MPS2 Cortex-M Prototyping System
CMSIS: Pathway to the ARM ecosystem

- Vendor-independent hardware abstraction layer for Cortex-M series
  - Open source software framework with processor HAL, DSP library, and RTOS kernel

- Consistent, generic, and standardized software building blocks
  - Optimized API that software creation, code portability, and middleware interfaces

- Infrastructure to accelerate time to market for device deployment
  - Software Packs to distribute device support, board support, and software building blocks

3668 devices supported
1.2M+ source files on GitHub
3M+ downloads in past six months
Keil MDK Microcontroller Development Kit
Most comprehensive development solution supporting over 3600 devices

CMSIS defines software packs that are created by ARM, silicon vendors, and middleware partners
For each project the version of the Software Packs may be specified
ARM C/C++ Compiler extensions for ARMv8-M

- C-Preprocessor macro `__ARM_FEATURE_CMSE` indicates secure or non-secure mode

- Function attributes to support calls between secure and non-secure mode
  - `__attribute__((cmse_nonsecure_entry))` Secure function that can be call by non-secure code
  - `__attribute__((cmse_nonsecure_call))` Call to non-secure function from secure code

```c
int SecureFunc (int v)
    attribute ((cmse_nonsecure_entry))
{
    SecureFunc PROC
        SG
    return v+1;
        ADDS     r0,r0,#1
}
    BXNS     lr
ENDP
```

Linker generates a export file with secure function entries
CMSIS-CORE for secure mode projects

Files relating to CMSIS-CORE including device specific files

partitions.h provides initial setup for SAU and configures non-secure mode memory areas and interrupts

CMSIS-CORE device files
CMSIS-CORE header files
generated from CMSIS-SVD
User program

startup_<device>.c
CMSIS device startup

system_<device>.c
CMSIS system and clock configuration

partitions.h
Secure attributes and interrupt assignment

<user>.c/c++
User application
main() { ... }
<device>.h
CMSIS device peripheral access

partitions.h
CMSIS-CORE extensions

- Partition setup and verification
  - Setup for **SAU**
  - Secure/non-secure Pointer validation

- Additional Functions to access:
  - New NVIC features
  - Secure and Non-secure MPU
  - Non Secure SysTick
  - New Special Registers

- API for RTOS interface
  - Management of secure stack memory
Debugging of software projects

- MDK offers debugging with:
  - Fast Model simulation environment for software development prior silicon
  - MPS2 target connection for testing with microcontroller prototypes
- Secure & Non Secure Debug Access

Enter password for Secure Debug Access
System visibility to processor and peripherals

MDK Debugger provides detailed dialogs for processor, core peripherals, and device peripherals.

- CMSIS-SVD delivers information about device specific peripherals
System visibility to software components

- Software components are “black box” for the application programmer
- MDK Debugger gives visibility to status and events of software components
- Supports secure firmware and requires no source and debug information

MDK Debugger + Event Recorder

XML File

Status and Event Views

Execution Status

Event Information

Software Component

- Supports secure firmware and requires no source and debug information

XML File

Status and Event Views

MDK Debugger + Event Recorder

Software Component
CMSIS-RTOS2 Secure system demo on Cortex-M33
Demonstration of ARMv8-M security features and system recovery

- Full source code is part of AppNote 291: Using TrustZone on ARMv8-M
Writing secure code for ARMv8-M processors
Potential attacks

How to avoid software design flaws in secure applications

- Return from secure to non-secure state
  - CPU Registers may still contain secret information

- Data pointers that obtain ‘trusted’ data in non-secure state
  - Non-secure code may provide incorrect pointers that address secure memory

- Asynchronous modifications to data processing in secure state
  - Non-secure interrupts could change values that are being processed in secure state
Return from secure to non-secure state

CPU registers may contain secret information

```
Secure mode

void decrypt(int32_t *data) {
    key = SECRET;
    //do the work
}

Non-secure mode

void spy_function() {
    decrypt(NULL);
    print_content_registers();
}
```

![Diagram showing transitions between secure and non-secure states and affected registers](image-url)
Return from secure to non-secure state
Clear shared CPU register content to avoid information leakage

Secure state

```c
void decrypt(int32_t *) __attribute__((cmse_nonsecure_entry));
void decrypt(int32_t *data) {
    key = SECRET;
    //do the work
}
```

```c
SG
MOV r3, #SECRET
@ do the work
MOV r3, #0
MSR APSR_flags, r3
BXNS lr
```

Non-secure state

```c
void spy_function() {
    decrypt(NULL);
    print_content_registers();
}
```

- Clear R0-R3 when used
- Clear status flags

ARM Compiler does not leak secure CPU register data to non-secure mode

SECURITY RISK – Solved! ARM Compiler clears CPU registers that may contain secure data.
Obtain ‘trusted’ data in non-secure code

Call Secure function and provide a data pointer

```c
void access_data(int32_t *, int32_t) __attribute__((cmse_nonsecure_entry));

void access_data(int32_t *data, int32_t n) {
    for (int32_t i=0; i<n; i++) {
        *data = array[i];
        data++;
    }
}
```

SECURITY RISK!
Is this a valid address to non-secure memory? If not, secure data may get corrupted.
Obtain ‘trusted’ data in non-secure code

Check for valid non-secure memory addresses

ARMv8-M provides Test Target (TT) instruction to check memory attributes:
- Returns MPU and SAU configuration information

ARM Compiler provides intrinsic functions for pointer validation:

```c
void *cmse_check_pointed_object(void *p, int flags);
void *cmse_check_address_range(void *p, size_t size, int flags);
```

Secure state

```c
void decrypt(int32_t *data) {
    int32_t *ptr;
    do {
        ptr = cmse_check_pointed_object(data, CMSE_NONSECURE);
        if (ptr == NULL) break;
        *ptr = decrypt_word(*ptr);
    } while (*ptr++);
}
```

SECURITY RISK – Solved!
Verify pointer target addresses with ARM Compiler intrinsic functions.
Asynchronous modifications to data processing

Non-secure interrupt functions may corrupt data currently processed

- Secure code should never trust non-secure data
- Non-secure memory may be modified by interrupt handlers
  - High priority interrupt is non-secure state can interrupt secure code execution
- A debugger access restriction can still change non-secure memory

```c
void setup_entry(struct config *c, int value) {
    if (c->index > 0 && c->index < sizeof(array)) {
        array[c->index] = value;
    }
}
```

SECURITY RISK!
Non-secure data may be altered during secure code execution
Asynchronous modifications to data processing

Ensure data processing in secure memory

- Copy non-secure data before validation
- Use ‘volatile’ attribute to disable potential compiler access optimizations

Secure state

```c
void setup_entry(volatile struct config *c, int value) {  
  int index_s = c->index;
  if (index_s > 0 && index_s < sizeof(array)) {  
    array[index_s] = value;
  }
}
```

SECURITY RISK – Solved!
Object is ‘volatile’ to avoid value propagation and value is validated.
Summary

- ARMv8-M provides the architecture for the next generation of secure connected embedded devices
- Software and tools make it easy for developers to use secure mode
- CMSIS provides software building blocks for faster time to market of embedded applications that require security