Agenda

The right level of security
Hardware support
Compartmentalization scenarios
Interaction between isolated components
Challenges in IoT

High volume, low cost, low power
- Microcontrollers
  - Small die
  - No MMU (single, physical address space)
  - XIP Flash code
  - Small SRAM

Wide spectrum of use-cases
- Different threat models
- Scalable solutions

Holistic approach to IoT security needed
Establishing the “right” level of security

Secure domain
Basic isolation – create a Secure Processing Environment

Protected TCB
Separate Root of Trust from Secure Partitions within SPE

Multiple tenancy in secure PE
More robustness – isolate all partitions from each other

Non-Secure isolation
Access policies for NS threads
Hardware isolation
... the foundation for software security

Physical isolation (e.g. dual-core system):
Dedicate cores/resources
Shared memory system or Mailbox
Concurrent execution

Temporal isolation (e.g. Arm-v8M):
Privilege control – using MPU
Secure/Non-secure states (Secure Attribution)
Shared Processing Element, resources
Interaction scenarios
Execution flows
Crossing boundaries in single processing element

Crossing from Non-secure to secure state
• Non-secure thread requests secure service

Isolated driver code
• ISR execution in unprivileged partition

Asynchronous events in non-secure PE
• Non-secure interrupt pre-empts secure operation
• Non-secure context awareness
• Concurrent secure service requests from non-secure threads
Non-secure call to secure service

Security state change only permitted using dedicated entry points

Wrapper function triggers privileged management code

Secure Partition Management code
- Performs parameter sanitization
- Sets up Secure Partition (container)
Non-secure call to secure service

NS thread mode
Client

- Call Secure Service

Secure thread

- Call Secure Service

S thread mode
Wrapper code

Secure veneer (NS Client ctx)

- Call Secure Request SVC

- Sanitize parameters

- Save NS Client ctx

- Setup SP context

- Perform secure service

S handler mode
Context management

Secure Request SVC

- Sanitize parameters

- Save NS Client ctx

- Setup SP context

- Call Response handler

S unprivileged thread
Sandboxed context

Secure Service function

- Save SP context

- Restore NS Client context

- Call Response handler
Secure interrupt deprivileging

- Privileged ISR is wrapper
  - Triggers Partition Manager

- Sandbox created
  - Returns to thread mode

- Secure Partition code
  - Executes deprivileged ISR
Secure interrupt deprivileging

Original mode

Original context

Interrupted code

- Gets interrupted

S handler mode

Wrapper code

Privileged ISR

- Call IRQ request SVC

S handler mode

Context management

IRQ Request SVC

- Set up MPU sandbox
- Switch PSP
- Ret. to unpriv. thread

S unprivileged thread

Sandboxed context

Secure Partition ISR

- Handle interrupt

Interrupted code

- Continue execution

Privileged ISR

- Return to original state

IRQ Done SVC

- Restore MPU config, PSP
- Return to priv. ISR

Secure Partition ISR

- Call IRQ Done SVC
Non-Secure interrupts

Pre-emption of secure execution

Non-secure thread is executing
Thread calls Secure Service
Non-secure IRQ pre-empts operation
Secure context is stacked
Non-secure ISR is executed
Return from ISR resumes secure execution
Context Management Functions

Non-secure context awareness in Arm-v8M

1. Non-secure threads created
2. Thread₁ calls Secure Service₁
3. Non-secure IRQ pre-empts operation -> context change
4. Thread₂ calls secure service₂
5. Secure service₂ returns
6. Thread₂ yields
7. Secure Service₁ returns

NS RTOS provides notification to SPM about context creation, deletion, load or store operation, enabling NS context-dependent access to secure assets/services.
Implementations
Secure Services implemented as functions

Arm-v8M architecture support
Secure Partition is a library of secure services
Synchronous execution of secure services
Programming model closely follows embedded/MCU concepts
Low footprint – on demand allocation of resources

Trusteed Firmware M library model

Non-secure PE
Thread₁
OS kernel

NS Client Container
NS call wrapper
Secure Partition
Secure Service

Secure Partition Manager
TF-M Core
Connection/message based interaction
Robust, more prescriptive framework
Static allocation of secure resources
Asynchronous processing of service requests
Less architecture dependent -> well suited for physical separation

Non-secure PE
Non-secure application
NS Partition Interface
NS Client Context
Secure Partition
Thread
Secure Partition Manager
Non-secure application
NS Partition Interface
NS Client Context
Secure Partition
Thread
Secure Partition
Thread
OS kernel
Secure Partition Manager
TF-M Core
Interaction with secure threads
TF-M Inter-Process Communication (IPC)

- For TF-M Thread model
- Secure Partitions provide secure services
  - NSPE is reflected as one Non-Secure Partition
- One thread in one Secure Partition
- While loop in thread waiting for messages
- Client call sent as messages
  - Non-Secure Partition is a client
  - Secure Partition could be a client
- Service Interrupt is handled asynchronously
Security Consideration on Compartmentalization

- No shared memory
- Streamed read/write APIs for copying memory
- Memory integrity checking based on isolation level
- Peripheral usage is also isolated
- Runtime protection rule change to isolate
Expand NSP with Arm-v8M TrustZone

- Non-Secure Processing Environment
  - Non-Secure Application
    - Client API
  - Secure and Non-Secure Callable
    - Secure Gateway

- Secure Processing Environment
  - NS Partition Interface
    - Secure Entry
  - Secure Partition #1
    - Secure Service
  - Secure Partition #2
    - Secure Service

Hardware Stack Pointer is switched in world transition
SP_NS <-> SP_S

SP_NonSecure
SP_Secure

Client API
Client and Service API
Message Manager
Scheduler
Secure Partition Manager (SPM)
Single NS Thread requests Secure Service

Non-Secure Processing Environment

Non-Secure Partition
- Non-Secure Application
  - Client API
- Secure and Non-Secure Callable
  - Secure Gateway

Secure Processing Environment

NS Partition Interface
- Secure Entry
- Secure Service

Secure Partition #1
- Secure Service

Secure Partition #2
- Secure Service

Frame generated during client API calling
- OS libraries
  - Frame
  - SP_Secure

Client API
- Client and Service API
  - Messages
  - Scheduler
  - Secure Partition Manager (SPM)
Multiple NS Thread request Secure Service

Non-Secure Processing Environment

Non-Secure Partition

Non-Secure Application

Non-Secure Callable

Secure and Non-Secure Callable

Secure Gateway

Frame generated during client API calling

OS libraries

OS Kernel

Secure Gateway

NS Partition Interface

Secure Entry

Secure Partition #1

Secure Service

Secure Partition #2

Secure Service

Client API

Client and Service API

Messages

Scheduler

Secure Partition Manager (SPM)
Multi-Thread NSPE Secure Call Solution 1

Frame generated during client API calling. Deny second secure calling since pending call.

Non-Secure Processing Environment
- Non-Secure Partition
  - Non-Secure Application
    - Non-Secure Callable
      - Client API
      - Secure Gateway
  - Non-Secure Application
    - OS libraries
    - OS Kernel
      - Frame #1
      - SP_Secure

Secure Processing Environment
- Secure Partition Interface
  - Secure Entry
  - Secure Partition #1
    - Secure Service
  - Secure Partition #2
    - Secure Service

Client API
- Client and Service API
  - Messages
    - Scheduler
      - Secure Partition Manager (SPM)
Multi-Thread NSPE Secure Call Solution 2

Non-Secure Processing Environment

Non-Secure Partition
- Non-Secure Application
  - Non-Secure Callable
    - Client API
- OS libraries
  - OS Kernel

Secure Processing Environment

NS Partition Interface
- Secure Entry

With these API, Non-Secure thread gets dedicated secure stack memory for secure call

Secure Context Management
- Scheduler
  - Secure Partition Manager (SPM)

NS Scheduler

sync thread status with SPM via Privileged API

Secure Context Management
- Messages
  - PSA Client and Service API
- PSA Client

Secure Stack
- Secure Stack 1
- Secure Stack 2
- Secure Stack 3

Thread
- Thread 1
- Thread 2
- Thread 3
Solution 2 Calling Process

Non-Secure Processing Environment

Non-Secure Partition

Thread 2

Client API

OS libraries

OS Kernel

Secure and Non-Secure Callable

Secure Gateway

NS Partition Interface

Secure Stack 2

Secure Partition #1

Secure Service

Secure Stack 2

CMSIS TZ Secure Context Management

PSA Client

PSA Client and Service API

Scheduler

Secure Partition Manager (SPM)

Secure call enters into dedicated secure stack

Secure Processing Environment

Secure Partition #2

Secure Service

Secure and Non-Secure Callable

CMSIS TZ Secure Context Management

PSA Client

PSA Client and Service API

Scheduler

Secure Partition Manager (SPM)
Non-Secure Interrupt Preempts Secure Service

OS Kernel would do ISR service task. For Non-Secure scheduler, it associated interrupted Secure Partition context with the caller Non-Secure Thread.
Secure Interrupt Preempts Execution

ISR creates interrupt message while interrupt happens and scheduler switches into the Secure Partition who is waiting for the interrupt message.

Non-Secure Processing Environment
- Non-Secure Partition
  - Non-Secure Application
    - Client API
  - OS Kernel

Secure Processing Environment
- Secure and Non-Secure Callable
  - Secure Entry
  - Secure Partition #1
  - Secure Service
  - Secure Gateway
  - Secure Execution
  - Scheduler
  - Secure Partition Manager (SPM)
  - Client and Service API
  - Wait_int()
  - Ex...
Compartmentalization in IoT – No one-size-fits-all

Secure/non-secure isolation:
- physical
- temporal

Privilege control:
- none
- within secure domain
- within non-secure domain

Interaction:
- function calls
- IPC
- hardware mailbox
How to get involved

TF-M is part of the Open Source/Open Governance trustedfirmware.org project
  • Code base: https://git.trustedfirmware.org/

TF-M Team @ OpenIoT Summit Europe 2018
  • Shebu Kuriakose
  • Ashutosh Singh
  • Ken Liu
  • Miklos Balint

Get in touch
  • Come round to the Arm booth during the summit
  • Contact TF-M team at support-trustedfirmware@arm.com

More info on developer.arm.com and trustedfirmware.org
Thank You!
Danke!
Merci!
谢谢!
ありがとうございます!
Gracias!
Kiitos!
감사합니다
धन्यवाद