Aerospace & Defence | Lighting | Power Management | Internet of Things | Transportation

Components – EMEA

Engineering DACH

TrustRAM and TrustZone on SAML11

Quang Hai Nguyen

Revision 1.0, 06.06.2019

Hands-on

Five Years Out

Revision History

Revision, Date	Editor	Subject (major changes)
Revision 0.1, 03.06.2019	Quang Hai Nguyen	Preliminary
Revision 1.0, 06.06.2019	Quang Hai Nguyen	Release

Table of Contents

List of Abbreviations

TZ
TrustZone 9

List of Figures

List of Icon Identifiers

Table 1: Icon Identifiers List


Extra information about a topic


Task need to be done


Important information or a warning


Result expected to see

Overview

Introduction

This hands-on will show you how to create an application using TrustRAM and TrustZone on SAML11 microcontroller.

Arm TrustZone

TrustZone provides the flexibility for hardware isolation of memories and peripherals, therefore reinforcing the ability of Intellectual Properties (IP) and Data protection. SAML11 provides up to six regions for the Flash, up to two regions for Data Flash, up to two regions for SRAM and the ability to assign peripherals, I/O pins, interrupts to secure or non-secure application.

TrustRAM

Trusted RAM implements 256 bytes of secure memory with address and data scrambling by user-defined key. Trusted RAM is also equipped with chip-level tamper detection and rapid tamper erase to resist micro-probing attacks.

Description

Inside SAML11, there are two applications running which are the secure and non-secure application. The secure application initializes the TrustRAM and prints the content of it on the console terminal. Then the secure application writes the serial number and revision of the onboard ATECC508, which are, in this case, considered as sensitive data into the TrustRAM.

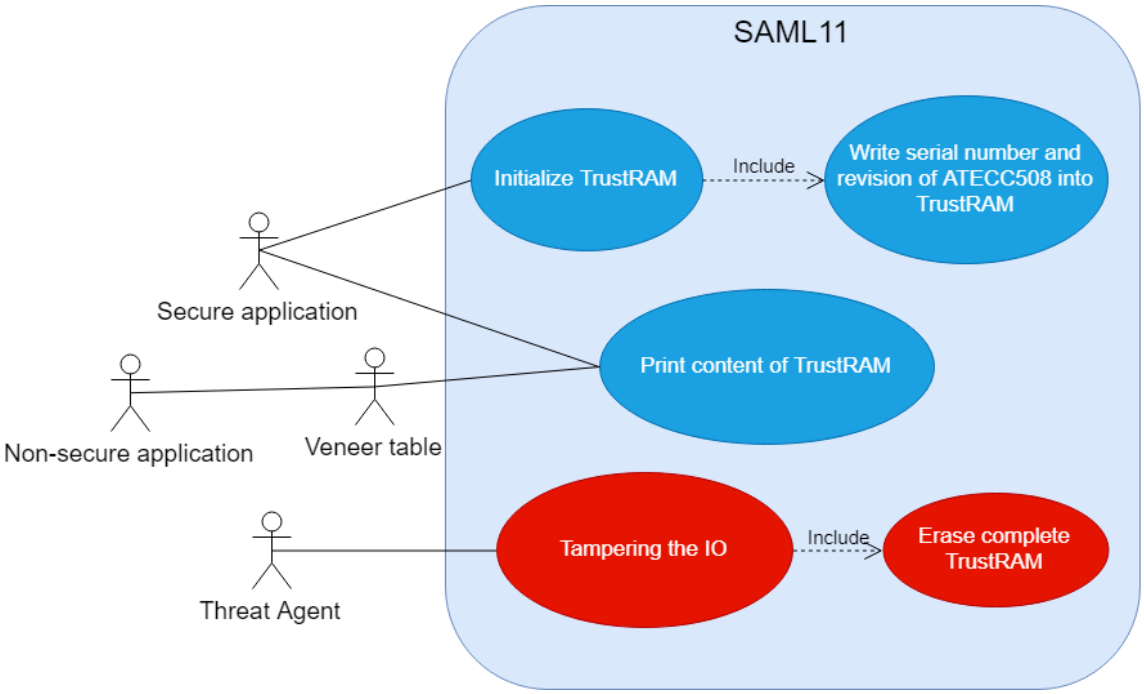


Figure 1: Use case Diagram

The non-secure application is initialized by the secure application. Non-secure application is not allowed to call any functions directly from the secure application but through a veneer table. In this case, it is the function to print the content of the TrustRAM.

When a tamper attempt is detected, the content in the TrustRAM will be automatically erased so the sensitive data is not exposed.

Assignments

- Assignment 1: import the project and explore the project configuration
- Assignment 2: Adding the code in secure world
- Assignment 3: Adding the code in non-secure world
- Assignment 4: Testing the application

Goal

After this hands-on, you will know the benefits of using TZ and TrustRAM for a secure application. In addition, you will have the confidence to demonstrate the hands-on to customers.

Requirement

Hardware

- SAML11 Xplained
- Type A-to-micro USB cable

Software

- Atmel Studio version 7
- TeraTerm

The software is already installed if you are using the virtual machine. Otherwise please refer to the Installation Guide for more information on how to get these programs.

Assignments

Assignment 1: import the project and explore the project configuration

Open the project with AtmelStudio

Navigate to the project folder and double click on the .atsln file. After clicking AtmelStart starts automatically.

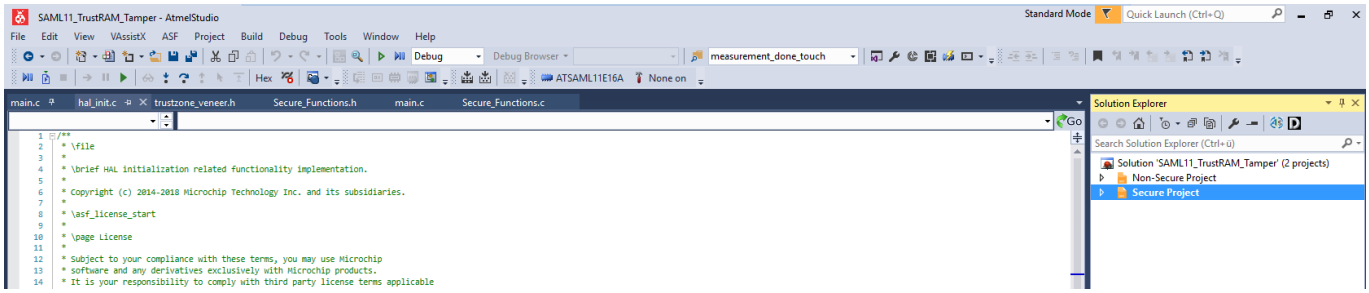


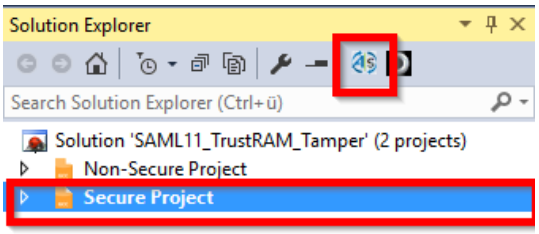
Figure 2: AtmelStudio

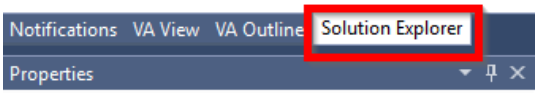
Alternatively, you can start AtmelStudio first, then choose File → Open → Project/Solution → point to .atsln file

Because the application runs on both secure and non-secure world, you will see there are two projects in the solution, which are Secure Project and Non-secure Project

In the solution explorer, focus on Secure project and choose AtmelStart icon

AtmelStart is an online tool so please make sure that you have the internet connection





If AtmelStart started correctly, the project configuration should be as following:

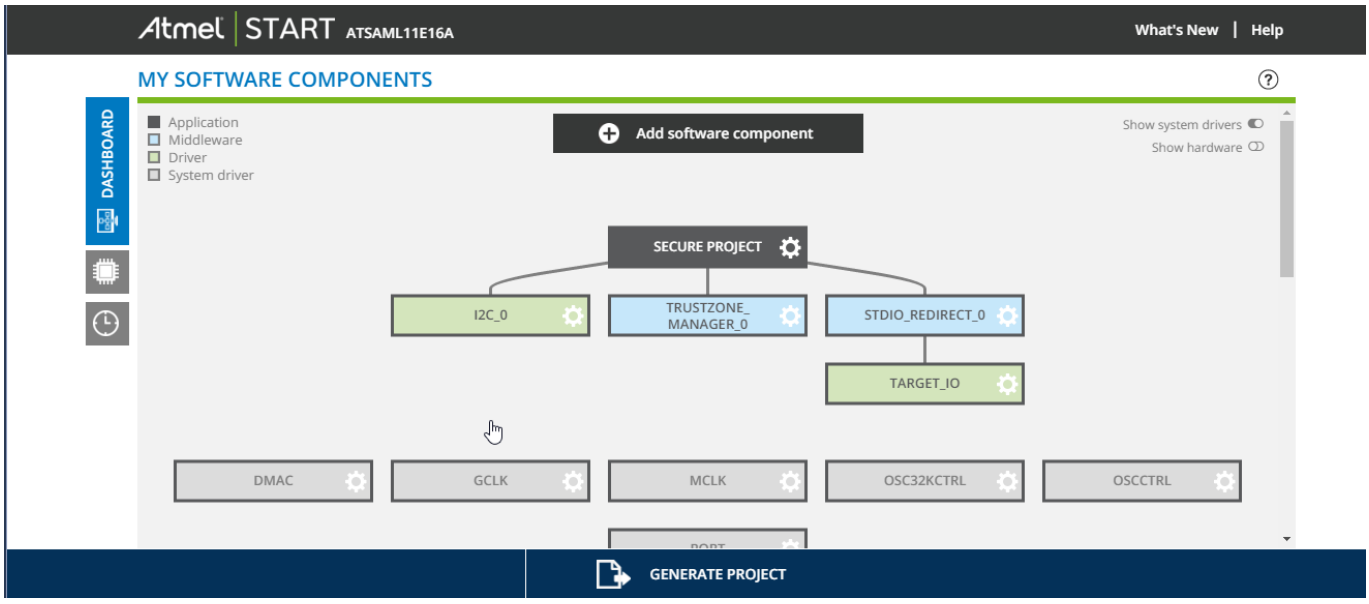
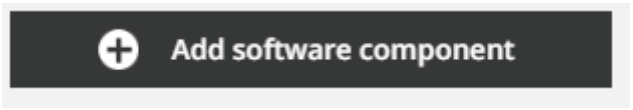


Figure 3: Project configuration in secure world

The project includes three components I2C, TrustZone manager, and stdio redirect. TrustZone manager initializes the memory zone of the microcontroller. Stdio redirect allows routing debug information to the terminal on the PC. I2C block handles the communication between the SAML11 Xplained and the ATECC508 on board.

The components RTC and TrustRAM will be manually initialized because they are not fully supported by AtmelStart at the moment.

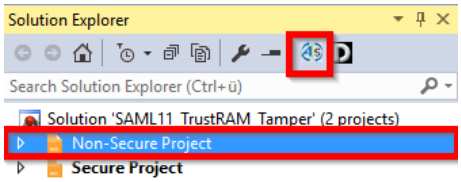
Those components can be added by click at Add software components button

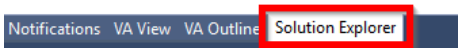


You are more than welcome to explore the detail of each component by clicking on it.

Please do not modify the configuration

The same procedure is applied to the Non-Project. In the solution explorer, focus on Secure project and choose AtmelStart icon





If AtmelStart started correctly, the project configuration should be as following:

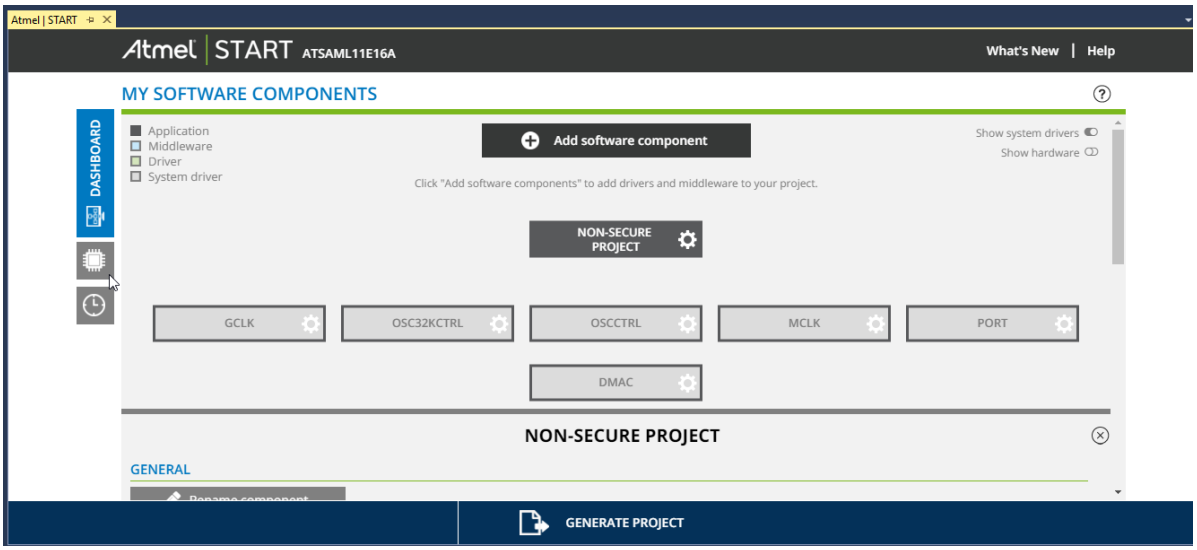


Figure 4: Project configuration in non-secure world

The software components for the non-secure world are empty because they are initialized by the secure world. Non-secure world is only allowed to call those components through defined functions in the veneer table.

You are more than welcome to explore the detail of each component by clicking on it.

Please do not modify the configuration

Compile the project and see task list

To compile the project, please click Build → Build Solution or simply press F7

After compiling the project, click View → Task List to see the tasks need to be done for this hands-on. Those Tasks will be discussed in detail in the next assignment

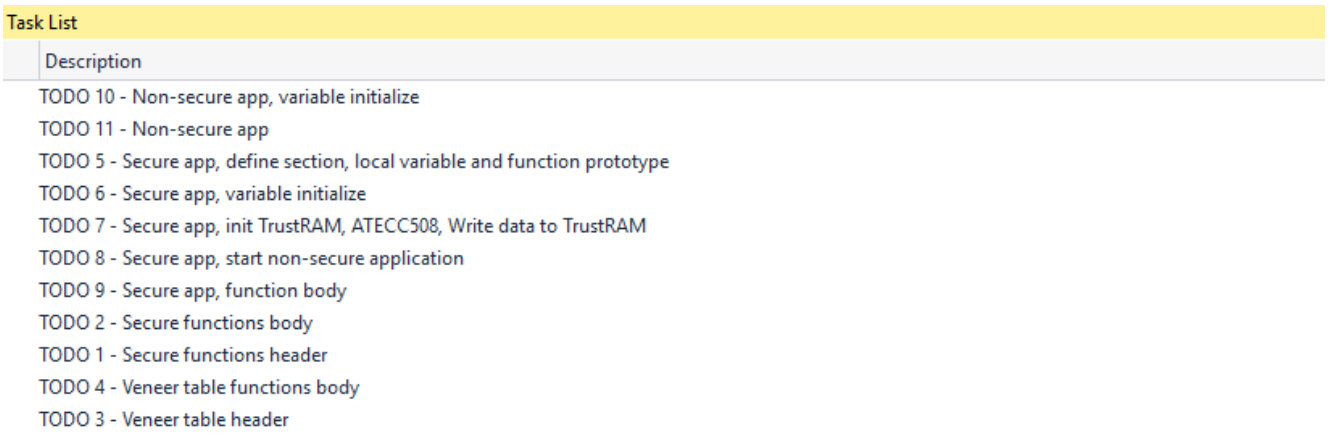


Figure 5: Task list of the hands-on

Assignment 2: Adding the code in secure world

If you get stuck at any point, there is a solution available. But please try it on your own before using the solution. Learning by doing 😊.

Please refer the text file for easily copy paste the code

Modify the code in Secure_Funtions.h

TODO 1 – Secure functions header

Double click on the TODO 1 in the Task list to jump to the code section

Right below the TODO line, add the following code:

```
/**
 *  @brief Initialize the RTC
 *
 *         @return             NULL     always return
 *
 *         It is including the tamper detection setup
 *
 *         @date    29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void sc_RTC_Init(void);

/**
```

```
 *  @brief Write data to the Trust RAM
 *
 *          @param          data    pointer to byte array written to RAM
 *          @param          size    number of byte written
 *          @param          offset  where to write data in RAM, starting from 0x00
 *
 *          @return         NULL    always return
 *
 *          Maximum 128 bytes can be written into RAM since we are using silent mode
 *
 *          @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void sc_TRAM_Write(uint8_t * data, uint8_t size, uint8_t offset);

/**
 *  @brief Read Data from Trust RAM
 *
 *          @param          ptr             buffer to store data read from RAM
 *          @param          size            number of byte to read
 *          @param          offset          where to read byte, starting from 0x00
 *
 *          @return NULL    always return
 *
 *          Maximum 128 bytes can be read from RAM since it is running silent mode
 *
 *          @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void sc_TRAM_Read(uint8_t * data, uint8_t size, uint8_t offset);

/**
 *  @brief Init  Trust RAM
 *
 *          @return         NULL    always return
 *
 *          the initialization includes silent mode, tamper detection
 *
 *          @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void sc_TRAM_Init(void);

/**
 *  @brief Print text in secure mode
 *
 *          @param          string  Text to be printed
 *
 *          @return         NULL    always return
 *
 *          Text in secure mode will be printed in green
```

```
 *
 *         @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void sc_ConsolePuts (uint8_t * string);

/**
 *  @brief Print text in non secure mode
 *
 *         @param          string     Text to be printed
 *
 *         @return         NULL       always return
 *
 *         Text in non secure mode will be printed in red
 *
 *         @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void nsc_ConsolePuts (uint8_t * string);

/**
 *  @brief Print a bytes on the console terminal in non secure mode
 *
 *         @param          ptr                pointer to byte array to print
 *         @param          length     number of byte to print
 *
 *         @return         NULL       always return
 *
 *         @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void nsc_PrintBytes(uint8_t * ptr, uint8_t length);

/**
 *  @brief Get serial number and revision from Trust RAM
 *
 *         @param          serial_buff             buffer storing serial number
 *         @param          serial_size             size of serial number
 *         @param          rev_buff                buffer storing revision
 *         @param          rev_size                size of revision
 *
 *         @return         NULL       always return
 *
 *         This function allows non-secure application to read out the serial number\n
 *         and revision from Trust RAM
 *
 *         @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void nsc_GetRevSerialNumber(uint8_t *serial_buff, uint8_t serial_size, uint8_t * rev_buff, uint8_t rev_size);
```

```
/**
 *  @brief          Read out complete data in Trust RAM
 *
 *          @param          ptr                 buffer storing data from RAM
 *          @param          size                size of RAM (128 bytes)
 *
 *          @return         NULL    always return
 *
 *          It allows non-secure application accesses the data in trust RAM
 *
 *          @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
void sc_ReadWholeRAM(u_int8_t *buff, uint8_t size);
```

Those functions take care of the initialization of the TrustRAM and the RTC for tamper detection. In addition, they handle the read/write data from/to TrustRAM.

Please refer the function header for more information about the function

Modify the code in Secure_Funtions.c

TODO 2 – Secure functions body

Double click on the TODO 2 in the Task list to jump to the code section

Right below the TODO line, add the following code:

```
void sc_RTC_Init(void)
{
        /* Configure PA08 as RTC IN0 (peripheral I) */
        PORT_SEC->Group[0].WRCONFIG.reg =
(uint32_t)(PORT_WRCONFIG_WRPINCFG|PORT_WRCONFIG_WRPMUX|PORT_WRCONFIG_PINMASK(1<<8)|PORT_WRCO
NFIG_PMUXEN|PORT_WRCONFIG_PMUX(8));

        /* Set APB Clock */
        MCLK->APBAMASK.reg |= MCLK_APBAMASK_RTC;

        /* Select RTC clock on XOSC32K */
        OSC32KCTRL->RTCCTRL.reg = OSC32KCTRL_RTCCTRL_RTCSEL_ULP1K;

        /* Reset RTC */
        RTC->MODE0.CTRLA.reg = RTC_MODE0_CTRLA_SWRST;
        while(RTC->MODE0.SYNCBUSY.bit.SWRST);
```

```
	RTC->MODE0.CTRLA.reg = (RTC_MODE0_CTRLA_MODE_COUNT32 | RTC_MODE0_CTRLA_PRESCALER_DIV1 |
RTC_MODE0_CTRLA_COUNTSYNC);

	/*Configure RTC Tamper on IN0 */
	RTC->MODE0.TAMPCTRL.reg = ( RTC_TAMPCTRL_IN0ACT_WAKE |	// Tamper action : Wake and set Tamper
flag
	RTC_TAMPCTRL_TAMLVL0 |			// Tamper edge : rising
	RTC_TAMPCTRL_DEBNC0			// Tamper Debounce  :Detect edge with synchronous stability
	);

	/* Enable Tamper event output*/
	RTC->MODE0.EVCTRL.reg = (RTC_MODE0_EVCTRL_TAMPEREO | RTC_MODE0_EVCTRL_OVFEO);

	/* Enable RTC */
	RTC->MODE0.CTRLA.reg |= RTC_MODE0_CTRLA_ENABLE;
	while(RTC->MODE0.SYNCBUSY.bit.ENABLE);
}

void sc_TRAM_Init(void)
{
	/* Enable TRAM in Tamper mode */
	TRAM->CTRLA.reg = TRAM_CTRLA_SWRST;
	while (TRAM->SYNCBUSY.bit.SWRST);
	/* Enable Data Scrambling with internal key */
	TRAM->DSCC.reg = (TRAM_DSCC_DSCEN |TRAM_DSCC_DSCKEY(0xCAFE));
	/* Enable TRAM security features (TAMPER , data remanence prevention , silent access*/
	TRAM->CTRLA.reg = (TRAM_CTRLA_TAMPERS|TRAM_CTRLA_SILACC|TRAM_CTRLA_DRP);
	TRAM->CTRLA.reg |= TRAM_CTRLA_ENABLE ;
}

void sc_TRAM_Write(uint8_t * data, uint8_t size, uint8_t offset)
{
	uint8_t i;
	uint8_t *addr_b;
	addr_b = (uint8_t*)&TRAM->RAM[0].reg;

	/*return if data bigger than RAM size*/
	if(size > 128)
		return;

	/* Initialize TRAM content */
	for (i=0; i < size ; i++) {
		*(addr_b + offset + i) = data[i];
	}

}

void sc_TRAM_Read(uint8_t * data, uint8_t size, uint8_t offset)
{
	int i;
	uint8_t *addr_b;
	addr_b = (uint8_t*)&TRAM->RAM[0].reg;
```

```
	/*Stop operation if size bigger than TRAM size*/
	if(size + offset > 128)
		return;

	for (i=0; i < size; i++)
	{
		data[i] = *(addr_b + offset + i);
	}
}

void sc_ReadWholeRAM(u_int8_t *buff, uint8_t size)
{
	if(size != 128)
		return;

	sc_TRAM_Read(buff, 128, 0);
}

void sc_ConsolePuts (uint8_t * string)
{
	/* Set display foreground color to green */
	printf("\033[0;32m");
	/* Print string on console */
	printf("%s", string);
}

void nsc_ConsolePuts (uint8_t * string)
{
	/* Set display foreground color to red */
	printf("\033[0;31m");
	/* Print string on console */
	printf("%s", string);
}

void nsc_PrintBytes(uint8_t * ptr, uint8_t length)
{
	uint8_t i = 0;
	uint8_t line_count = 0;

	/* Set display foreground color to red */
	printf("\033[0;31m");

	for(;i < length; i++) {
		printf("0x%02x, ",ptr[i]);
		line_count++;
		if(line_count == 8) {
			printf("\r\n");
			line_count = 0;
		}
	}

	printf("\r\n");
}
```

```
void nsc_GetRevSerialNumber(uint8_t *serial_buff, uint8_t serial_size, uint8_t * rev_buff, uint8_t rev_size)
{
	/*Make sure to have correct size*/
	if(serial_size != 9 || rev_size != 4)
		return;

	sc_TRAM_Read(serial_buff, serial_size, 0x00 );
	sc_TRAM_Read(rev_buff, rev_size, 0x00 + serial_size );

}
```

TODO 3 – Veneer table header

Double click on the TODO 3 in the Task list to jump to the code section. In this section we fill in the APIs in the veneer table, which allows the non-secure application calling the APIs from the secure application. For our hands-on, the non-secure application is allowed to call these APIs:

- Print string on the terminal
- Print byte on the terminal
- Print serial number and revision on the terminal
- Print the content of the TrustRAM on the terminal

Right below the TODO line, add the following code:

```
/**
 *  @brief Print text in non secure mode
 *
 *	@paramstring	Text to be printed
 *
 *	@return NULL	always return
 *
 *	Text in non secure mode will be printed in red
 *
 *	@date	29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
extern void nonsecure_ConsolePuts (uint8_t * string);

/**
 *  @brief Print a bytes on the console terminal in non secure mode
 *
 *	@paramptr		pointer to byte array to print
 *	@paramlength	number of byte to print
 *
 *	@return NULL	always return
 *
 *	@date	29.05.2019 - initial
 *
 *  @bug No known bugs.
```

```
 */
extern void nonsecure_PrintBytes(uint8_t * ptr, uint8_t length);

/**
 *  @brief Get serial number and revision from Trust RAM
 *
 *          @paramserial_buff                   buffer storing serial number
 *          @paramserial_size                   size of serial number
 *          @paramrev_buff              buffer storing revision
 *          @paramrev_size              size of revision
 *
 *          @returnNULL     always return
 *
 *          This function allows non-secure application to read out the serial number\n
 *          and revision from Trust RAM
 *
 *          @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
extern void nonsecure_GetRevSerialNumber(uint8_t *serial_buff, uint8_t serial_size, uint8_t * rev_buff, uint8_t rev_size);

/**
 *  @brief          Read out complete data in Trust RAM
 *
 *          @paramptr                   buffer storing data from RAM
 *          @paramsize      size of RAM (128 bytes)
 *
 *          @returnNULL     always return
 *
 *          It allows non-secure application accesses the data in trust RAM
 *
 *          @date   29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
extern void nonsecure_ReadWholeRAM(uint8_t *buff, uint8_t size);
```

Please refer the function header for more information about the function

Modify the code in veneer table

TODO 4 – Veneer table function body

Double click on the TODO 4 in the Task list to jump to the code section. This section contains the body of the function we have initialized in TODO 3.

Right below the TODO line, add the following code:

```
void __attribute__((cmse_nonsecure_entry)) nonsecure_ConsolePuts (uint8_t * string)
{
	nsc_ConsolePuts (string);
}
void __attribute__((cmse_nonsecure_entry)) nonsecure_PrintBytes(uint8_t * ptr, uint8_t length)
{
	nsc_PrintBytes(ptr, length);
}

void __attribute__((cmse_nonsecure_entry)) nonsecure_GetRevSerialNumber(uint8_t *serial_buff, uint8_t serial_size,
uint8_t * rev_buff, uint8_t rev_size)
{
	nsc_GetRevSerialNumber(serial_buff, serial_size, rev_buff, rev_size);
}

void __attribute__((cmse_nonsecure_entry)) nonsecure_ReadWholeRAM(u_int8_t *buff, uint8_t size)
{
	sc_ReadWholeRAM(buff, size);
}
```

Modify the code in main.c of the secure application

TODO 5 – Secure app, define section, local variable and function prototype

Double click on the TODO 5 in the Task list to jump to the code section. This section contains the initialization of the local variable and function prototype used in main.c.

Right below the TODO line, add the following code:

```
/* TZ_START_NS: Start address of non-secure application */
#define TZ_START_NS                 0x00008000
#define DATA_OFFSET_IN_RAM    0x00
#define REVISION_SIZE              0x04
#define TRUST_RAM_SIZE          128

/* Handle the response status from the secure element*/
#define CHECK_STATUS(s)                                                    \
if(s != ATCA_SUCCESS) {                                                    \
	printf("status code: 0x%x\r\n", s);                                \
	printf("Error: Line %d in %s\r\n", __LINE__, __FILE__);            \
	while(1);                                                          \
}


/* Local variable section ----------------------------------------------------------*/

/* typedef for non-secure callback functions */
typedef void (*ns_funcptr_void) (void) __attribute__((cmse_nonsecure_call));

/**
 *	@brief data structure for secure element instant
```

```
 *
 *          It contains the information to initialize the communication between controller and secure element
 */
ATCAIfaceCfg cfg_ateccx08a_i2c_host = {
            .iface_type                             = ATCA_I2C_IFACE,
            .devtype                                = ATECC608A,
            .atcai2c.slave_address      = 0xC0,
            .atcai2c.bus                            = 1,
            .atcai2c.baud                           = 400000,
            .wake_delay                             = 800,
            .rx_retries                             = 20,
            .cfg_data           = &I2C_0
};

/* Local function prototype section ---------------------------------------------------------*/

/**
 *  @brief Print a bytes on the console terminal
 *
 *          @paramptr                   pointer to byte array to print
 *          @paramsize          number of byte to print
 *
 *          @return NULL        always return
 *
 *          @date    29.05.2019 - initial
 *
 *  @bug No known bugs.
 */
static void print_bytes(uint8_t * ptr, uint8_t length);
```

TODO 6 – Secure app, variable initialize

Double click on the TODO 6 in the Task list to jump to the code section.

Right below the TODO line, add the following code:

```
volatile ATCA_STATUS status;
uint8_t serial_number[ATCA_SERIAL_NUM_SIZE];
uint8_t revision_number[REVISION_SIZE];
uint8_t ram_buff[TRUST_RAM_SIZE];

/* Pointer to Non secure reset handler definition*/
ns_funcptr_void NonSecure_ResetHandler;
```

TODO 7 – Secure app, init TrustRAM, ATECC508, write data to TrustRAM

Double click on the TODO 7 in the Task list to jump to the code section. In this section we will initialize the TrustRAM and the ATECC508. After that, we write the serial number and revision number into the TrustRAM and print the whole TrustRAM content on the terminal.

Right below the TODO line, add the following code:

```
sc_ConsolePuts((uint8_t *)"hello world from secure application\r\n");

/*Initial TrustRAM and display its content*/
sc_RTC_Init();
sc_TRAM_Init();
sc_ReadWholeRAM(ram_buff, TRUST_RAM_SIZE);
print_bytes(ram_buff,TRUST_RAM_SIZE);

/*Intial ATECC508, read out revision number, serial number and write them to TrustRAM*/
status = atcab_init( &cfg_ateccx08a_i2c_host );
CHECK_STATUS(status);

sc_ConsolePuts((uint8_t *)"Initializing ATECC508\r\n");

status = atcab_read_serial_number((uint8_t*)&serial_number);
CHECK_STATUS(status);
status = atcab_info(revision_number);
CHECK_STATUS(status);

sc_TRAM_Write(serial_number, ATCA_SERIAL_NUM_SIZE, DATA_OFFSET_IN_RAM);
sc_TRAM_Write(revision_number, REVISION_SIZE, DATA_OFFSET_IN_RAM + ATCA_SERIAL_NUM_SIZE);

sc_ConsolePuts((uint8_t *)"ATECC508 is initialized. Revision and serial number are stored in Trust Ram\r\n");
```

TODO 8 – Secure app, start non-secure application

Double click on the TODO 8 in the Task list to jump to the code section. In this section we initialize the non-secure application and jump into the non-secure world

Right below the TODO line, add the following code:

```
/* Set non-secure main stack (MSP_NS) */
__TZ_set_MSP_NS(*((uint32_t *)(TZ_START_NS)));

/* Get non-secure reset handler */
NonSecure_ResetHandler = (ns_funcptr_void)(*((uint32_t *)((TZ_START_NS) + 4U)));

/* Start Non-secure Application */
NonSecure_ResetHandler();
```

TODO 9 – Secure app, function body

Double click on the TODO 9 in the Task list to jump to the code section.

Right below the TODO line, add the following code:

```
static void print_bytes(uint8_t * ptr, uint8_t length)
{

        uint8_t i = 0;
        uint8_t line_count = 0;
        for(;i < length; i++) {
                printf("0x%02x, ",ptr[i]);
```

```
                line_count++;
                if(line_count == 8) {
                        printf("\r\n");
                        line_count = 0;
                }
        }

        printf("\r\n");
}
```

Assignment 3: Adding the code in non-secure world

Modify the code in main.c of the non-secure application

TODO 10 – Non-secure app, variable initialize

Double click on the TODO 10 in the Task list to jump to the code section.

Right below the TODO line, add the following code:

```
#define SERIAL_NUM_SIZE                 0x09
#define REVISION_SIZE           0x04
#define TRUST_RAM_SIZE          128

uint8_t rev[REVISION_SIZE];
uint8_t ser[SERIAL_NUM_SIZE];
uint8_t ram_buff[TRUST_RAM_SIZE];
```

TODO 11 – Non-secure app

Double click on the TODO 11 in the Task list to jump to the code section. In this section, the application waits until the user presses the button on the SAML11 Xplained board. If the button is pressed, the application prints the content of the TrustRAM on the terminal.

Right below the TODO line, add the following code:

```
nonsecure_ConsolePuts((uint8_t *)"Hello World from non secure application\r\n");


/* Replace with your application code */
while (1) {

        /*Waiting for user input to read the data from Trust RAM*/
        nonsecure_ConsolePuts((uint8_t *)"\r\n\r\n");
        nonsecure_ConsolePuts((uint8_t *)"Press SW0 to call and print serial number and revision\r\n");
        while(gpio_get_pin_level(SW0));

        /* Read data from Trust RAM and print on Terminal*/
        nonsecure_ReadWholeRAM(ram_buff, TRUST_RAM_SIZE);
        nonsecure_ConsolePuts((uint8_t *)"Data in RAM after initialization\r\n");
```

```
        nonsecure_PrintBytes(ram_buff, TRUST_RAM_SIZE);

        nonsecure_GetRevSerialNumber(ser, SERIAL_NUM_SIZE, rev, REVISION_SIZE);

        nonsecure_ConsolePuts((uint8_t *)"Serial number: \r\n");
        nonsecure_PrintBytes(ser, SERIAL_NUM_SIZE);
        nonsecure_ConsolePuts((uint8_t *)"Revision number: \r\n");
        nonsecure_PrintBytes(rev, REVISION_SIZE);

        delay_ms(500);
}
```

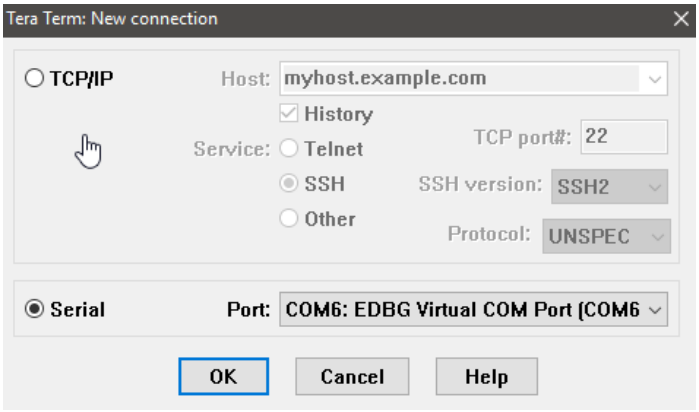
Assignment 4: Testing the application

Compile the project again.

If there is no problem, the compilation output should be as following

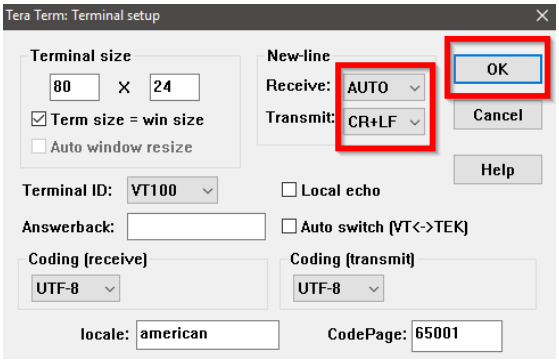
Setup TeraTerm

Before running the application, we need to setup TeraTerm.



COM port may be different on your machine

Go to Setup → Terminal and set the following configuration



Press reset button on the SAML11 Xplained

If it is setup correctly, you should see the following output on the terminal

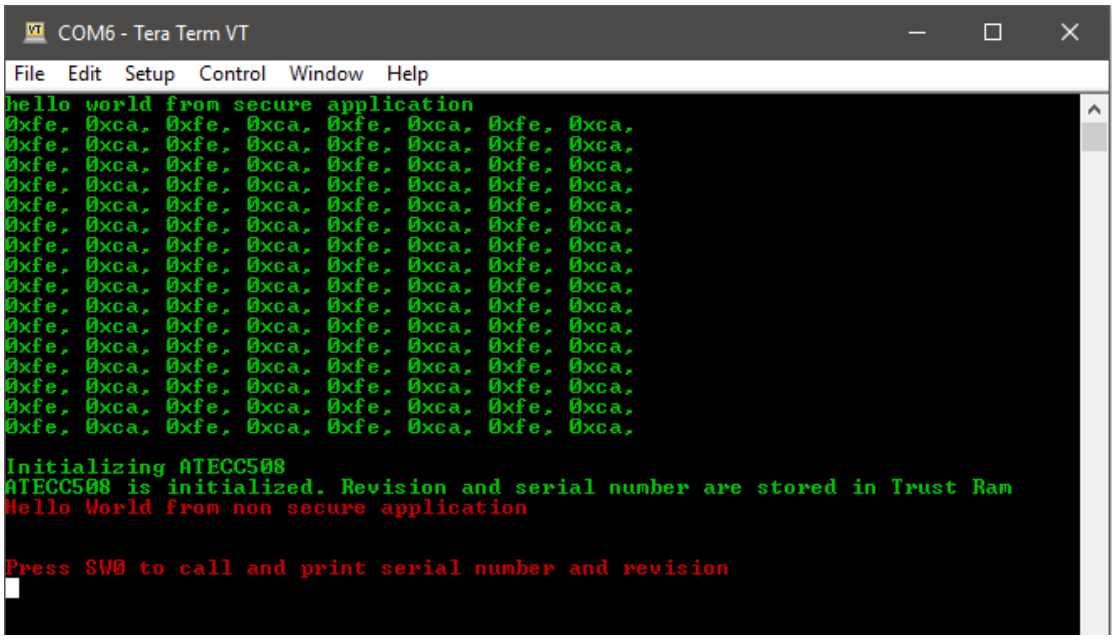


Figure 6: printed result on the terminal

The green text depicts the information coming from the secure application meanwhile the red one shows the information coming from the non-secure world.

After the initialization, the TrustRAM is filled with the scrambling word, which is 0xcafe. Then the Serial number and revision are written into the TrustRAM. The user can pressed the SW0 button on the evaluation kit to retrieve the serial number, revision and also the complete content of the TrustRAM.

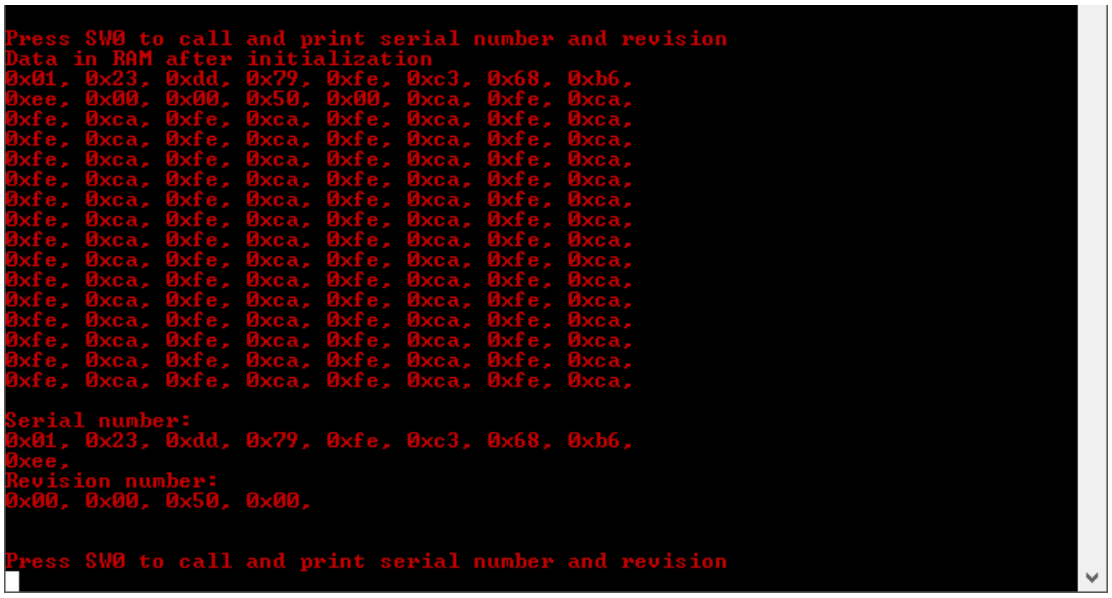
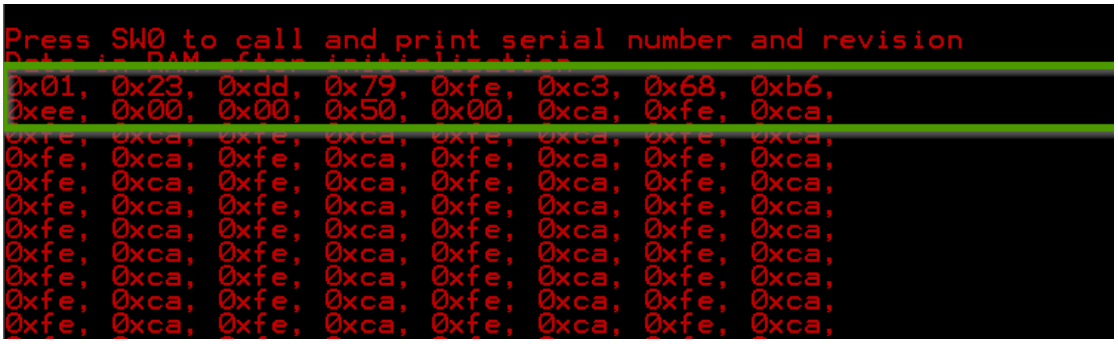


Figure 7: Content of the TrustRAM

If we observe the content of the RAM, we can see that the first 13 bytes of the TrustRAM are the serial number and revision of the ATECC508



Touch the extension EXT1 on the SAML11 evaluation kit to simulate the tamper event and read the content of the RAM again.

Press SW0 again to read the data from the TrustRAM. Because of the tamper event, the data in the RAM is wiped out. Therefore we can only read out 0x00 from the RAM.

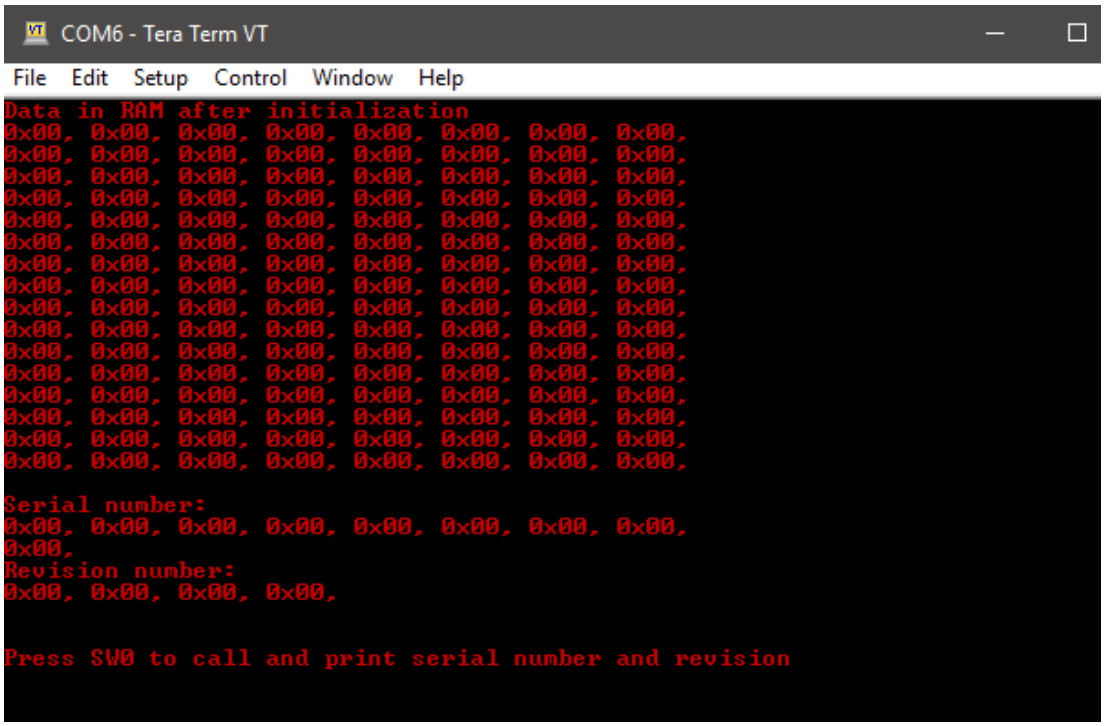


Figure 8: Content of TrustRAM after tamper event is detected

Congratulation! You have finished the hands-on section

Contact information

Janus Piwek
Dipl.-Ing. (FH)
Market Development Engineer - Microchip
Arrow Central Europe GmbH

e-mail: jpiwek@arroweurope.com

The End