MAX32555 SecureROM package installer and quick start guide

Contents

1 Version **1**

2 Purpose **1**

3 Release Notes **1**

4 Tested configuration **1**

5 3rd parties packages **1**

6 Installation **2**

6.1 Windows users only 2

6.2 Linux users only 2

7 Use **3**

7.1 program the CRK 3

7.2 program the application 4

7.3 Execution 6

8 Notes **6**

9 Glossary **6**

10 Tools purpose **7**

1 Version

- this document version is 1.1.0 (9-jul-2015)
- 1.0.0: initial document release, for A1 support
- 1.1.0: document update for A2 support

2 Purpose

- the MAX32555 SecureROM package is a set of tools, examples and documents enabling user to securely program, load and run applications on the MAX32555 secure microcontroller .
- this package enables users to set a MAX32555 in phase 3, program the CRK on the MAX32555, move the MAX32555 to phase 4, program an application and runs it on the MAX32555.
- this package requires installation of 3rd parties tools.
- This is run on a PC, connected to the MAX32555.
- this package demonstrates:
  - how to set up the MAX32555 life-cycle,
  - how to program the customer CRK, using a test CRK,
  - how to program a demo application in the internal flash, to be run in the internal RAM.
  - a glossary is available at the end of this document

3 Release Notes

- this package version supports MAX32555 A1 and A2
- this package version is validated on Windows Seven 64-bit

4 Tested configuration

- on PC side, this package has been tested on:
  - Windows Seven 64-bit, using Cygwin 64-bit 1.7,
  - Ubuntu 12.04 LTS,
- on embedded side, this package has been tested on
  - MAX32555 A1, A2
  - Evkit 1.0

5 3rd parties packages

- the Maxim Integrated delivered package contains documentation, examples and tools, from Maxim Integrated.
- 3rd party tools to be installed are:
  - For Windows users only:
    - Cygwin
      - a collection of tools which provide a Linux look and feel environment for Windows
      - needed for tools execution
      - download and install guide

6 Installation

6.1 Windows users only

1. copy the SecureROM package in /cygdrive/c/securerom (i.e. c:/securerom)
2. cygwin installation
   a. use the Cygwin_install.pdf document
3. set up package application
   a. goto secureROM package home directory, e.g. securerom
   b. select the targeted MAX32555 version, A1, by using the option --soc=A1 (default is A1)
   c. note that MAX32555 A2 is compliant with A1, so selecting A1 is ok for A2 chips.

```
$ cd securerom
$ bash setup.sh --soc=A1
  Setting up MAX32555 rev.A1's SecureROM Package... for cygwin
  dos2unix: converting file Host/customer_scripts/lib/romversion to Unix  ↩
      format ...
   Info:: New environment variable LHASSA_SCRIPTS_PATH added by
          LHASSA PACKAGE's setup.sh script into /home/<username>/. ↩
              bash_profile.
   >>> You need to logout then login again to make this variable available ↩
       .
   >>> You can also do "source /home/<username>/.bash_profile" to make  ↩
       this variable available (in this terminal session only).
```

   d. once this is done, the LHASSA_SCRIPTS_PATH variable can be used for locating the package directory.

6.2 Linux users only

1. copy the SecureROM package in ~/securerom
2. set up package application
   a. goto secureROM package home directory, e.g. securerom
   b. select the targeted MAX32555 version, A1 by using the option --soc=A1 (default is A1)
   c. note that MAX32555 A2 is compliant with A1, so selecting A1 is ok for A2 chips.

```
$ cd securerom
$ bash setup.sh linux ./
  Setting up MAX32555 rev.A1's SecureROM Package... for linux
   Info:: New environment variable LHASSA_SCRIPTS_PATH added by
          LHASSA PACKAGE's setup.sh script into /home/<username>/. ↩
              bash_profile.
   >>> You need to logout then login again to make this variable available ↩
       .
   >>> You can also do "source /home/<username>/.bash_profile" to make  ↩
       this variable available (in this terminal session only).
```

   d. once this is done, the LHASSA_SCRIPTS_PATH variable can be used for locating the package directory.

7 Use

7.1 program the CRK

1. program the test CRK and move MAX32555 life cycle phase from 3 to 4 (to be done once per chip)

   a. Note: the SCP packets have already been generated (and stored in prod_p3_write_crk) for this test CRK, this is why programming can be done immediately. These packets are an example of what Maxim Integrated generates from the CRK public key supplied to Maxim Integrated.

   b. the connection between the Host and the MAX32555 is a serial connection, 115200 8N1,

   c. connect the EvKit UART0 connector to the targeted COM port,

   d. run writecrk.sh with the chosen COM port (in our example, COM1 for Windows, /dev/ttyS0 for Linux) and prod_p3_write_cr as parameters

```
 $ cd $LHASSA_SCRIPTS_PATH/Host/customer_scripts/scripts
 $ bash ./writecrk.sh COM1 ../../../SCP_Packets/prod_p3_write_crk
Ready to execute /cygdrive/c/securerom/PACKAGE_LHASSA_1.2.0/SCP_Packets/ ↩
   prod_p3_write_crk
Power cycle the MAX32555 system then press [Enter] IMMEDIATELY!
Please wait...
Open file: packet.list
Open serial port: COM1 (timeout: 2s)
Start SCP session (use -v for details)


0000001 SEND> connection_request
0000002 WAIT< connection_reply
0000003 SEND> ack
0000004 SEND> hello_request
0000005 WAIT< ack
0000006 WAIT< hello_reply
error: received packet is not the expected one
======================================
Expected Command
ScpCmd:
ScpCmdHdr:
  - SYNC: ('\xbe', '\xef', '\xed')
  - CTL: 5
  - DL: 54
  - ID: 145
  - CKS: 98
Data:
2000003248454 ↩
   c4c4f20484f53540100000000000c00500abcdef01000102abcdaef60000000000
       0000000000000000000000000000
Checksum: a836acf6
--------------------------------------
Received Command
ScpCmd:
ScpCmdHdr:
  - SYNC: ('\xbe', '\xef', '\xed')
  - CTL: 5
  - DL: 54
  - ID: 145
  - CKS: 98
Data:
```

```
2000003248454 ↩
   c4c4f20484f5354060001020300004005000abcdef01000102abcdf6ae0000000000
       000000000000000000000000000
Checksum: ae138608
======================================
0000007 SEND> ack
0000008 SEND> write_crk
0000009 WAIT< ack
0000010 WAIT< write_crk_response
0000011 SEND> ack
0000012 SEND> disconnection_request
0000013 WAIT< disconnection_reply
SCP session OK
SUCCESS.
```

e. Note1: the MAX32555 is now in phase 4, with the test CRK programmed in its OTP. Any SCP command will now require a signature with this key to be approved.

f. Note2: the error message displayed during the CRK programming is related to the MAX32555 (varying) USN which is different from the (fixed) USN used in the packet. It is not an issue but an expected warning.

7.2 program the application

1. program test application in the MAX32555 internal flash,

   a. signed application

      i. the application signing operation is automatically performed within the COBRA environment, in the Eclipse IDE, usually using the "format binary" target

      ii. in the SLA/build, there is a binary example of such signed application, sla_template.sbin

      iii. this .sbin has been signed using the test CRK private key.

      iv. the version of the application shall be consistent with the MAX32555 version. For example, the version to be used with the A1 or the A2 is 1.0.0, coded 01000000. This has been set up automatically by the setup.sh script. The hexadecimal parameters are the load address and the jump address. These parameters have also been provided in the COBRA environment, at the signature step.

   b. generate the packets for the application programming

```
 $ cd $LHASSA_SCRIPTS_PATH/Host/customer_scripts/scripts
 $ bash ./build_application.sh ../../../SLA/build/sla_template.sbin  ↩
    buildapp ../keys/maximtestcrk.key
SBL/SCP packets builder v3.7.4 (build 1) (c)Maxim IC 2006-2013
--warning: this tool does not handle keys in a manner compliant with PCI  ↩
   PTS--
WARNING: <session_build.ini> not found
<buildapp/scp.log> created
Generated SCP packets in: buildapp
Use sendscp.sh <serial_port_spec> buildapp to run the SCP session  ↩
   generated by this script.
SUCCESS.
```

      i. buildapp directory has been created and packets have been generated and stored in

   c. program the generated packets

      i. connect the EvKit to the targeted serial port (COM1 is for Windows, to be replaced by /dev/ttyS0 for Linux)

      ii. use sendscp.sh script to run the SCP session generated by this script

```
 $ cd $LHASSA_SCRIPTS_PATH/Host/customer_scripts/scripts
 $ bash ./sendscp.sh COM1 buildapp
Ready to execute /cygdrive/c/securerom/PACKAGE_LHASSA_1.2.0/Host/ ↩
   customer_scripts/scripts/buildapp
Power cycle the MAX32555 system then press [Enter] IMMEDIATELY!
Please wait...
Open file: packet.list
Open serial port: COM1 (timeout: 2s)
Start SCP session (use -v for details)
 0000001 SEND> connection_request
 0000002 WAIT< connection_reply
 0000003 SEND> ack
 0000004 SEND> hello_request
 0000005 WAIT< ack
 0000006 WAIT< hello_reply
 error: received packet is not the expected one
 ========================================
 Expected Command
 ScpCmd:
 ScpCmdHdr:
  - SYNC: ('\xbe', '\xef', '\xed')
  - CTL: 5
  - DL: 54
  - ID: 145
  - CKS: 98
 Data:
 2000003248454 ↩
    c4c4f20484f535401000000000000c000000000000000000000000000000000000000000000

 Checksum: 0ff7f7b7
 ----------------------------------------
 Received Command
 ScpCmd:
 ScpCmdHdr:
  - SYNC: ('\xbe', '\xef', '\xed')
  - CTL: 5
  - DL: 54
  - ID: 145
  - CKS: 98
 Data:
 2000003248454 ↩
    c4c4f20484f535401000001040000400500abcdef01000102abcdf6ae000000000000000000

 Checksum: b6742160
 ========================================
 0000007 SEND> ack
 0000008 SEND> del_mem
 0000009 WAIT< ack
 0000010 WAIT< del_mem_response
 0000011 SEND> ack
 0000012 SEND> write_mem
 0000013 WAIT< ack
 0000014 WAIT< write_mem_response
 0000015 SEND> ack
 0000016 SEND> write_mem
 0000017 WAIT< ack
```

```
0000018 WAIT< write_mem_response
0000019 SEND> ack
0000020 SEND> write_mem
0000021 WAIT< ack
0000022 WAIT< write_mem_response
0000023 SEND> ack
0000024 SEND> write_mem
0000025 WAIT< ack
0000026 WAIT< write_mem_response
0000027 SEND> ack
0000028 SEND> disconnection_request
0000029 WAIT< disconnection_reply
SCP session OK
SUCCESS.
```

7.3 Execution

1. start the application by resetting the board
   a. after more than 10s (5s for SCP-USB time window, 5s for SCP-Serial time window), the EvKit LED0 blinks.

8 Notes

- 32-bit Windows binary files (libucl.dll and session_build.exe) can be found in Host/session_build/windows_32bits. If your system is not 64-bit compliant, copy these files
- it is possible to change the SCP ports (UART and USB) timings, using a script and session build tool
  – for no UART timing: write-timeout 0 0000
  – for no USB timing: write-timeout U 0000
  – for no VBUS timing: write-timeout V 0000
  – for USB timing set up at 5s (i.e. 5000ms = 0x1388): write-timeout U 8813
  – the typical use is:

```
$ cd Host/customer_scripts/scripts
$ mkdir session_timeout
$ ../lib/session_build.exe session_mode=SCP_LHASSA_ECDSA verbose=yes  ↩
   output_file=session_timeout/session_timeout pp=ECDSA script_file= ↩
   script_timeout.txt ecdsa_file=../keys/crk_ecdsa_angela_test.key
$ cd session_timeout
$ ls -1 *.packet >packet.list
$ cd ..
$ bash ./sendscp.sh COM1 session_timeout
```

9 Glossary

- CRK: Customer Root Key, the customer ECDSA 256-bit public key, to be written in the OTP, used for signatures verifications,
- OTP: One-Time Programming Memory, an embedded memory used for configuration data storage,
- phase: the MAX32555 life cycle is made of several phases: phase 3 allows to program the CRK, phase 4 allows to program and run applications,

- SCP: Secure Communication Protocol: the secure protocol over UART and USB used for firmware, keys updates, OTP configuration, applets execution
- SLA: Second-Level Application: the final application launched by the SecureROM.
- USN: Unique Serial Number, a value unique per chip.

10 Tools purpose

- for more details, see UG28S04 *Secure ROM code User Guide*
- session_build: builds the SCP packets from a script (typically the signed CRK or a signed final application)
- serial_sender: sends the SCP packets to the chip