Keywords: RSA key-pair, MAXQ1103, Crossworks compiler, Modulo Arithmetic Accelerator (MAA)

APPLICATION NOTE 4347

RSA Key Generation Using the DeepCover Secure Microcontroller (MAXQ1103)

Dec 19, 2008

Abstract: Maxim's RSA key generation library provides easy-to-use interfaces to generate RSA key pairs using the DeepCover® Secure Microcontroller (MAXQ1103). The MAXQ1103 is designed for financial terminal applications and has a number of security features including RSA. The RSA library uses the modulo arithmetic accelerator (MAA) which provides cryptographic operations up to 2048 bits. The MAA allows the user to compute a set of operations that are important for many cryptographic operations. The article also explains why the MAXQ1103 Evaluation (EV) Kit and the CrossWorks development environment provide an ideal platform to develop secure applications.

Introduction

This application note demonstrates the generation of RSA key-pair sets using the DeepCover® Secure Microcontroller (MAXQ1103). The article also demonstrates how to encrypt and decrypt the plain text messages using RSA key-pair sets. To demonstrate the timing of the RSA computation, the article shows data from the DS5250 high-speed, secure microcontroller which illustrates the performance improvement achieved with the MAXQ1103.

The MAXQ1103 microcontroller is designed for financial terminal applications and has a number of security features including RSA. The hardware modulo arithmetic accelerator (MAA) provides cryptographic operations up to 2048 bits. The MAA allows the user to compute a set of operations that are important for many cryptographic operations. Example operations include modular exponentiation (a^e mod m); modular multiplication (a × b mod m); modular square (b^2 mod m); modular square followed by modular multiply ((b^2 mod m) × a mod m); modular addition; and modular subtraction.

The MAXQ1103 Evaluation (EV) Kit and CrossWorks development environment provide an ideal platform to develop these secure applications. The EV kit comes with all the tools necessary for development: 4MB of external program memory; 4MB of external data memory; 2 serial ports; 2 smart-card chips (one full size and one SIM card); a USB connector; an LCD screen; a 16-bit keypad; and a prototyping area.

Getting Started with RSA Key-Pair Generation

The sample application binary (`rsa_1103.hex`) and sample application code that generate the RSA key pair can be obtained by writing to: Tech Support.

The following information will help you build and execute the RSA key-pair sample application program which is written in C and uses the CrossWorks compiler for MAXQ30.

Setting Up the MAXQ1103 EV Kit

Refer to application note 4273, "Getting Started with the MAXQ1103 Evaluation Kit and the CrossWorks Compiler for the MAXQ30," for details on setting up the development environment for the MAXQ1103.

The MAXQ1103 EV kit is shown in **Figure 1**. The hardware components required to generate the RSA key pair are:

1. MAXQ1103 EV kit board.
2. JTAG board.
3. JTAG cable (to connect the MAXQ1103 EV kit board and JTAG board).
4. 9-pin serial cable (to connect the PC's COM port and EV kit's serial port 0).
5. Two regulated power supplies (5V, ±5%, 300mA, center positive); one supply is for the MAXQ1103 EV kit and the other for the JTAG board.

The jumper settings for the EV kit are shown in the table below.

Jumper	Status
JU1	Short
JU3	Short
JU4	Short

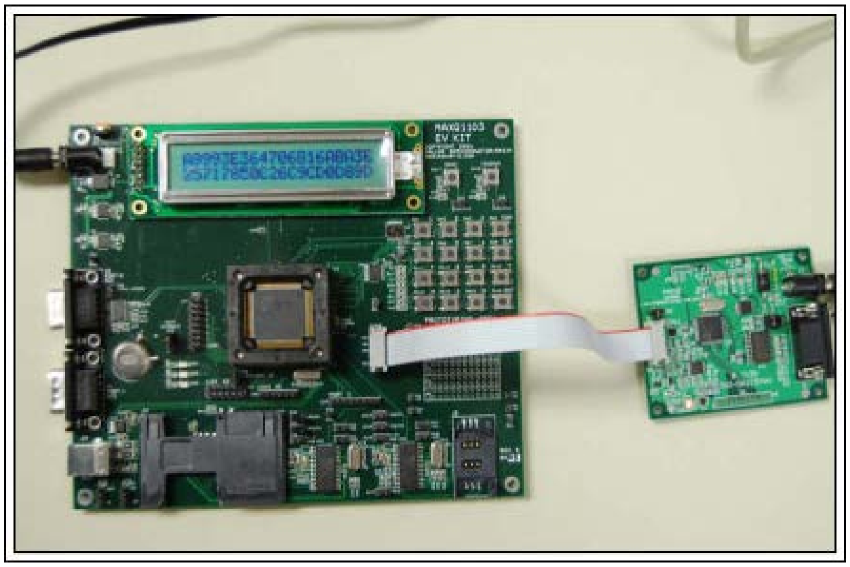


Figure 1. MAXQ1103 EV kit with JTAG board.

Follow these steps to set up the EV kit and begin using the software for this application.

1. Install the CrossWorks compiler for MAXQ30. The tool suite is available from Rowley Associates and is version 2.0.0.2008063000.2293 at the time of this publication.
2. Connect the serial cable between the EV kit's port 0 and the PCs COM port so you can observe the application output onto the PC.
3. Connect the serial cable between the JTAG board and PCs COM port. This connection is used to download the application onto the EV kit board.
4. Open the project `rsa_1103.hzp`.

5. Click on Project, then Rebuild to generate the `rsa_1103.hzx` output file. This file will be downloaded to the MAXQ1103 EV kit. Additionally you can generate the `rsa_1103.hex` file by modifying the project properties. Go to Project properties, then Linker Options, then additional Output Format. Choose "hex" from the dropdown list.
6. Connect to the target using the "Connect to the target" tab in your Targets window.
7. The application prints the results onto serial port 0 of the EV kit.

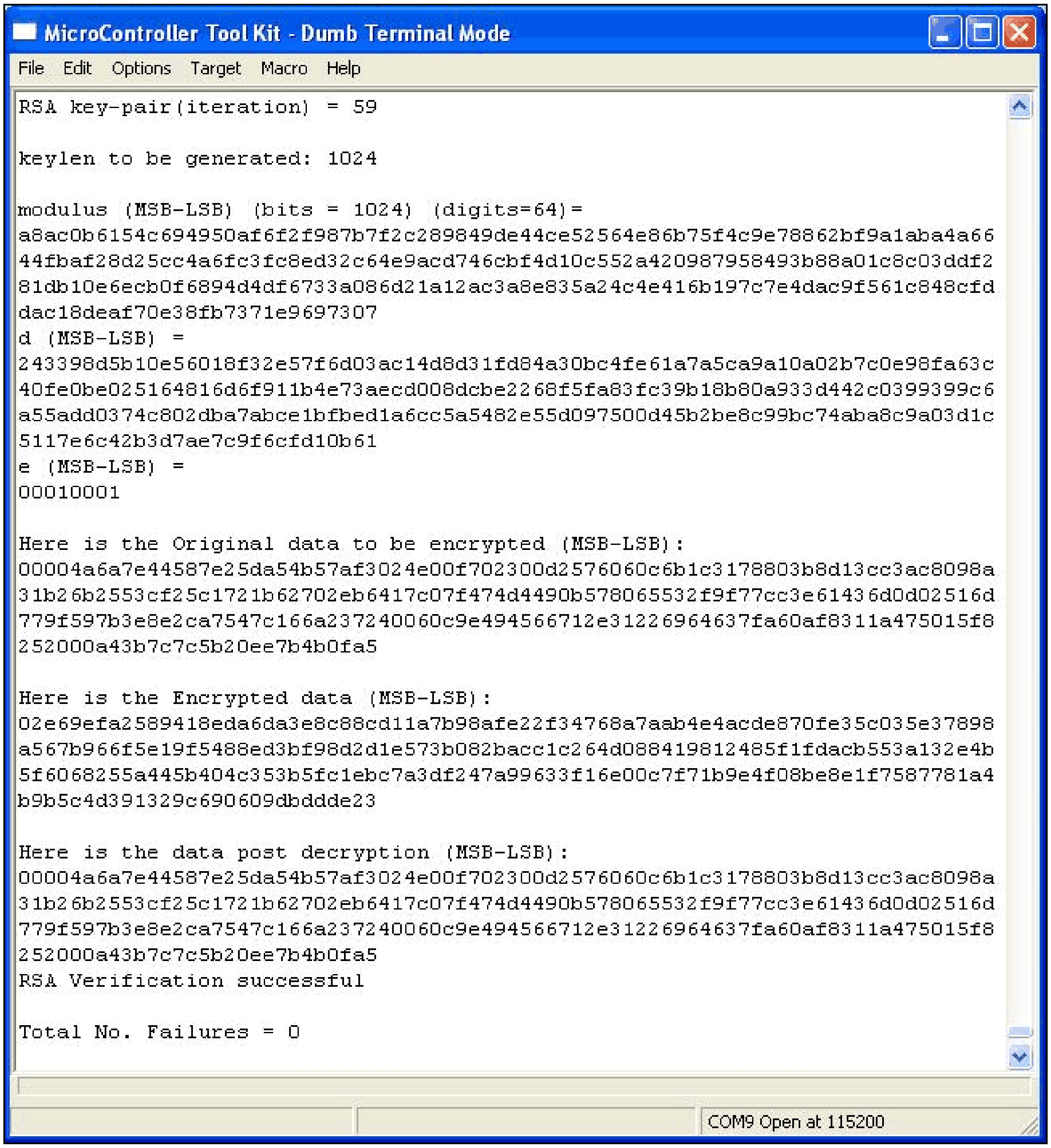
   Open the hyperterminal and configure the appropriate COM port connection for 115200, 8 data bits, parity none, 1 stop bit, and no flow control.

   OR

   You can use Maxim's Microcontroller Tool Kit (MTK2) software to see the application results. Install the MTK2 and open the MTK2 in dumb terminal mode. Configure the appropriate serial port for 115200 baud rate, and open the serial connection.
8. Click on Debug, then Run to load and run the application. This application note uses MTK2 to observe the results.

The application will now prompt you for some data entry with the request to "Enter key length bits to be generated:"

Enter the number (for example, 1024) and wait for the application to display the results. The application displays the execution status shown in **Figure 2**. It takes approximately 5 seconds to generate a 1024-bit-length RSA key pair, then encrypt and decrypt the random message. This time can vary for each execution. The average times taken to generate an RSA key-pair for various bit lengths are tabulated in **Table 1**.


MicroController Tool Kit - Dumb Terminal Mode
File Edit Options Target Macro Help
COM9 Open at 115200

```
RSA key-pair(iteration) = 59

keylen to be generated: 1024

modulus (MSB-LSB) (bits = 1024) (digits=64)=
a8ac0b6154c694950af6f2f987b7f2c289849de44ce52564e86b75f4c9e78862bf9a1aba4a66
44fbaf28d25cc4a6fc3fc8ed32c64e9acd746cbf4d10c552a420987958493b88a01c8c03ddf2
81db10e6ecb0f6894d4df6733a086d21a12ac3a8e835a24c4e416b197c7e4dac9f561c848cfd
dac18deaf70e38fb7371e9697307
d (MSB-LSB) =
243398d5b10e56018f32e57f6d03ac14d8d31fd84a30bc4fe61a7a5ca9a10a02b7c0e98fa63c
40fe0be025164816d6f911b4e73aecd008dcbe2268f5fa83fc39b18b80a933d442c0399399c6
a55add0374c802dba7abce1bfbed1a6cc5a5482e55d097500d45b2be8c99bc74aba8c9a03d1c
5117e6c42b3d7ae7c9f6cfd10b61
e (MSB-LSB) =
00010001

Here is the Original data to be encrypted (MSB-LSB):
00004a6a7e44587e25da54b57af3024e00f702300d2576060c6b1c3178803b8d13cc3ac8098a
31b26b2553cf25c1721b62702eb6417c07f474d4490b578065532f9f77cc3e61436d0d02516d
779f597b3e8e2ca7547c166a237240060c9e494566712e31226964637fa60af8311a475015f8
252000a43b7c7c5b20ee7b4b0fa5

Here is the Encrypted data (MSB-LSB):
02e69efa2589418eda6da3e8c88cd11a7b98afe22f34768a7aab4e4acde870fe35c035e37898
a567b966f5e19f5488ed3bf98d2d1e573b082bacc1c264d088419812485f1fdacb553a132e4b
5f6068255a445b404c353b5fc1ebc7a3df247a99633f16e00c7f71b9e4f08be8e1f7587781a4
b9b5c4d391329c690609dbddde23

Here is the data post decryption (MSB-LSB):
00004a6a7e44587e25da54b57af3024e00f702300d2576060c6b1c3178803b8d13cc3ac8098a
31b26b2553cf25c1721b62702eb6417c07f474d4490b578065532f9f77cc3e61436d0d02516d
779f597b3e8e2ca7547c166a237240060c9e494566712e31226964637fa60af8311a475015f8
252000a43b7c7c5b20ee7b4b0fa5
RSA Verification successful

Total No. Failures = 0
```

Figure 2. Execution status and results of sample application.

Developing a Simple Application Using the RSA Key-Generation Library

The library provides easy-to-use interface functions in C to generate the key pair and encrypt/decrypt the user message using the private/public key. Refer to the `rsalib_1103.h` file to see the prototypes of these interfaces. This application demonstrates the use of these interface functions:

```
rsa_generateKeySet(...)
rsa_bignumModExp(...)
```

Typical uses of these interface functions follow.

```
{
        unsigned long exp = 0x10001;                    // public exponent
        DIGIT *c,*x;
        BIGNUM *d;
        BIGNUM *e;
        BIGNUM *pq;
        DIGIT *plain_text;

        d = rsa_newNum();
        e = rsa_newNum();
        pq = rsa_newNum();

        // generate the public and private key pair
        // 'maxq1103_rnd' is a call-back function to generate random numbers
        using 'random number generator' (RNG) module built into the MAXQ1103
microcontroller.

        err = rsa_generateKeySet(d,e,exp,maxq1103_rnd,pq,keylen);

        if(err != RSA_SUCCESS)
        {       printf("\nFailed to generate RSA Keysets. Error code=%d",err);
                rsa_freeNum(d);
                rsa_freeNum(e);
                rsa_freeNum(pq);
                return;
        }

        // allocate memory for 'plain_text' and 'assign values'
        // allocate memory for 'x' which will contain the encrypted text

        rsa_bignumModExp(x,plain_text,e,pq);             // use public key for
encryption

        // allocate memory for 'c' which will contain the decrypted/original text

        rsa_bignumModExp(c,x,d,pq);                      // use private key for
decryption

}
```

Typical test results for different bit lengths are shown below. These numbers can vary for each execution.

Table 1. Average Time for Generating an RSA Key Pair

RSA Bit Length Generated	Number of Tests Run	Average Time Taken per Test to Generate an RSA Key Pair (seconds)	
		MAXQ1103 EV Kit at 12MHz	DS5250 EV Kit at 22.1MHz
256	60	0.84	4.8
512	60	1.71	10.76
1024	60	4.55	26.6
1536	60	9.98	63.81
2048	60	15.63	122.4

Conclusion

Maxim provides a library for RSA key generation. This library allows applications written in C to access the power and functionality of the MAXQ1103 microcontroller's hardware to generate RSA key pairs up to a maximum of 2048 bits. The library uses MAA and RNG modules built into the MAXQ1103 to compute the RSA key pairs. The hardware MAA supports IEEE® Public Key Cryptographic standard (P1363) for asymmetric cryptographic operations based on DSA, RSA, and ECDSA algorithms.

Related Parts	
DS5250	High-Speed Secure Microcontroller
MAXQ1103	DeepCover Secure Microcontroller with Rapid Zeroization Technology and Cryptography