Advanced Computer Systems – 2018

Final project

Submitted by: Eyal Golombek

Date: 19/3/18

Project Idea and Goal:

The goal of the project was to create a secure authentication token that will allow users to authenticate themselves in an easy, fast and highly secure way when performing purchases on-line.

This product will help fight online credit card theft since it required the user to be in physical proximity to our board.

The result should be a very small product that can be attached to a key chain, making the experience of shopping on-line very simple and secure!

The security of the system relies on A-Symetric encryption, and private keys. The private key is stored on the board, and has to be generated along with the credit card, on the day the bank distributes the credit card to the user.

The idea came to me when I heard that massive amount of credit card frauds that happen every year, and the fact that dealing with it is so simple.
In 2016 the net loss from credit card frauds was estimated at 24.71 Billion dollars
https://www.creditdonkey.com/credit-card-fraud-statistics.html

The implementation:

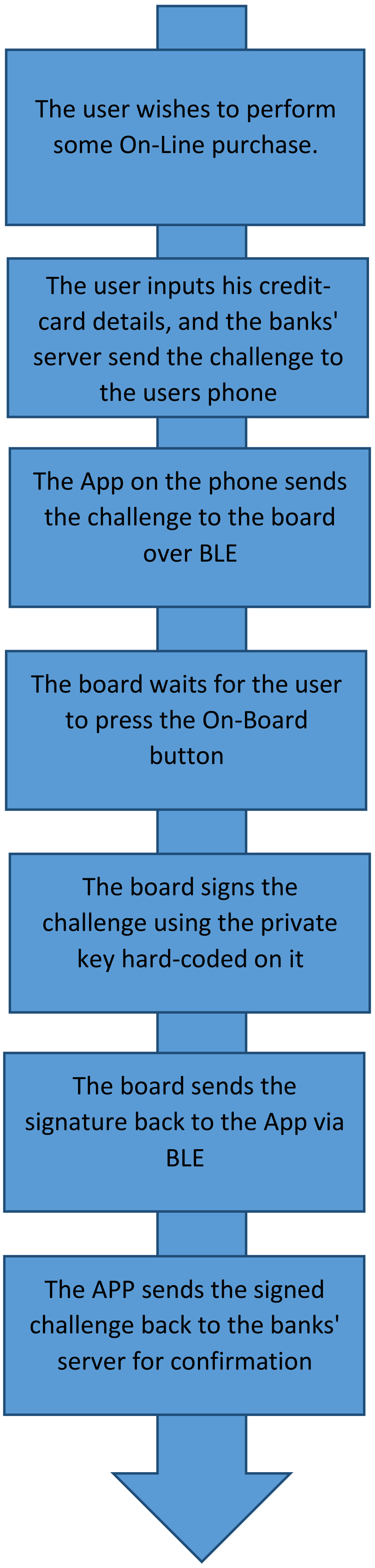
Using the CC1350 Launchpad I ported an open-source code library called mbedTLS to run under the environment of the TI-RTOS. Using the mbedTLS library the board can perform RSA signatures to challenges it receives over BLE. The signatures are performed using a private key hard-coded into the board, which is never revealed, making the entire system highly secure.

Moreover, I have created a simple Android APP to be a proof of concept for applications that on-line shops / banks will have to create if they wish to use the product.

When the user wishes the verify the identity of the credit card holder, it uses the APP to send a challenge to the board. The boards' LEDs will start blinking, indicating an incoming challenge. The board will now wait for the user to press the on-board button, making the board perform the RSA signature operation.

Once done, the board will indicate it has finished, by setting the green LED on, and sending BLE notification to the APP. The app will than verify the challenge was answered correctly, and will indicate the board to return to its original state.

Use-case Explanation:


The user wishes to perform some On-Line purchase.
The user inputs his credit-card details, and the banks' server send the challenge to the users phone
The App on the phone sends the challenge to the board over BLE
The board waits for the user to press the On-Board button
The board signs the challenge using the private key hard-coded on it
The board sends the signature back to the App via BLE
The APP sends the signed challenge back to the banks' server for confirmation

Challenges:

Challenge No. 1 - Big number operations on Arm Cortex M3:

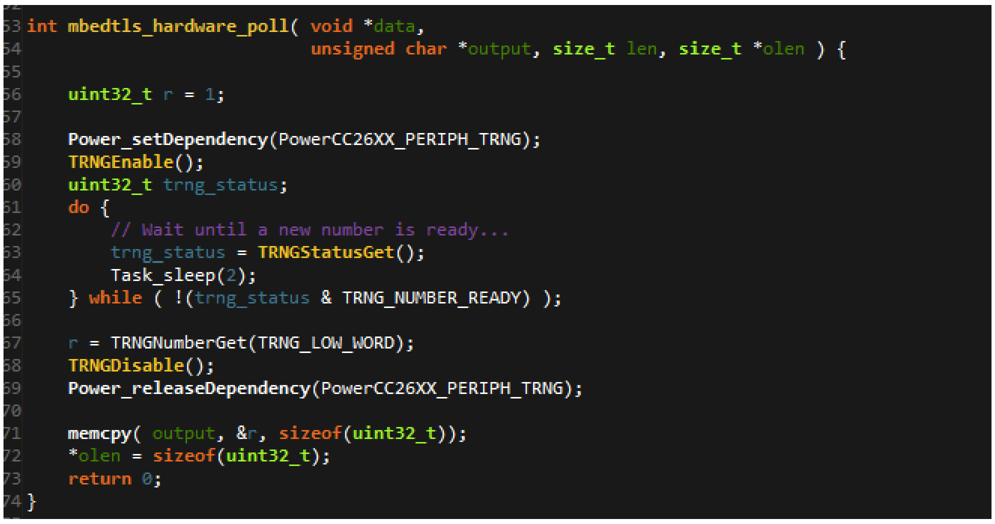
Any cryptographic library needs to perform mathematic operations with very large numbers. The mbedTLS library assumes the CPU can perform 64 bit number multiplication natively. The Arm Cortex M3 has no such op-code, so I had to remove the support for that and use only 32 bit operations.

Moreover, mbedTLS assumes it runs on x86/64 CPUs, thus it utilizes the AES-NI and VIA-Padlock op-codes which perform cryptographic operations natively in x86 environment.

Challenge No. 2 – Generating random numbers:

Any cryptographic library required that ability to generate random numbers. mbedTLS assumes some sort of POSIX API for random using the rand() function. TI-RTOS doesn't have any default random function.

In order to solve this problem I implanted my own random generating function, which uses the TRNG chip on-board. I had to use the TRNG driver present in the TI SDK and to integrate it with the power-controlling driver that will control the power to the TRNG device.



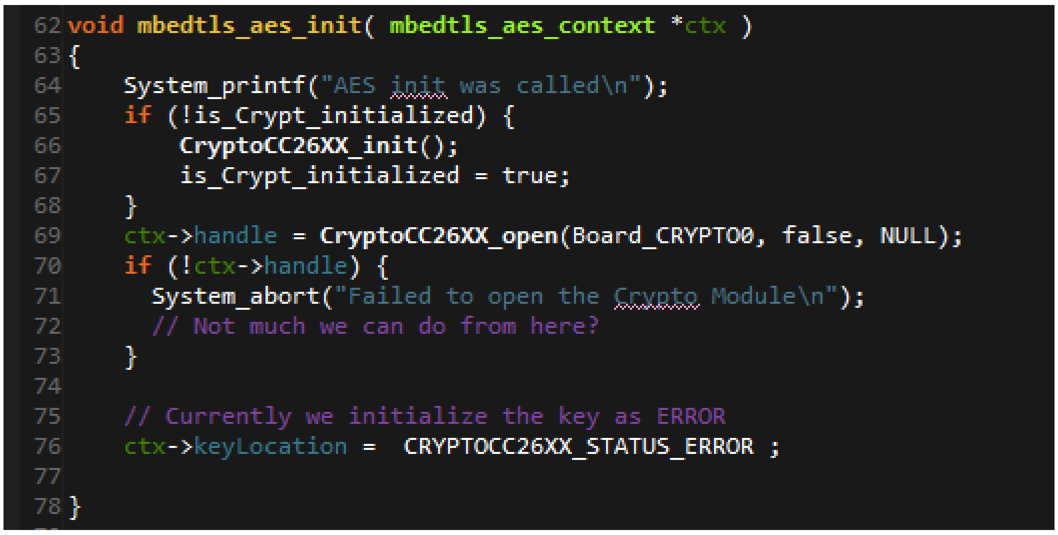
```
int mbedtls_hardware_poll( void *data,
                           unsigned char *output, size_t len, size_t *olen ) {

    uint32_t r = 1;

    Power_setDependency(PowerCC26XX_PERIPH_TRNG);
    TRNGEnable();
    uint32_t trng_status;
    do {
        // Wait until a new number is ready...
        trng_status = TRNGStatusGet();
        Task_sleep(2);
    } while ( !(trng_status & TRNG_NUMBER_READY) );

    r = TRNGNumberGet(TRNG_LOW_WORD);
    TRNGDisable();
    Power_releaseDependency(PowerCC26XX_PERIPH_TRNG);

    memcpy( output, &r, sizeof(uint32_t));
    *olen = sizeof(uint32_t);
    return 0;
}
```

Challenge No 3. – Integration with the On-Board AES:

In order to generate cryptographically secure random, we use an algorithm called ctr_drbg, which uses AES CTR mode. **AES uses very large tables which take up a lot of memory**, so I had to substitute the programmatic implementation of AES with the on-board chip capable of performing AES.

I had to use the AES driver supplied by TI-RTOS and implement its functions. AES encryption is performed as a transaction to the external device.

Here is a snippet of the code, which performs the initialization of the AES chip:



```c
void mbedtls_aes_init( mbedtls_aes_context *ctx )
{
    System_printf("AES init was called\n");
    if (!is_Crypt_initialized) {
        CryptoCC26XX_init();
        is_Crypt_initialized = true;
    }
    ctx->handle = CryptoCC26XX_open(Board_CRYPTO0, false, NULL);
    if (!ctx->handle) {
      System_abort("Failed to open the Crypto Module\n");
      // Not much we can do from here?
    }

    // Currently we initialize the key as ERROR
    ctx->keyLocation =  CRYPTOCC26XX_STATUS_ERROR ;

}
```

Using the on-chip AES saved a lot of very valuable memory. (Read more about the scarcity of memory in the following sections)

Challenge No. 4 – The board has only 20KB of RAM Memory:

The CC1350 has only 20KB of RAM. The project tries to use a BLE Stack and to perform operations on very large numbers, all stored in memory.

When I just started working the project would crash all the time without any indication to the reason. Later I realized I kept having stack over flows when performing those large number operations.

When I tried making all the stacks much larger, I ran out of heap memory.

The trickiest part was the fact that the ICALL library allocates its own heap – using any left out space. Any left out space from the other tasks is used as the heap of the ICALL framework. This fact made the BLE stop working without any notification, since the ICALL simply didn't have any heap space left, but there was no way to monitor the lack of heap space, since the regular heap had a lot of space left.

Steps I made in order to solve the scarcity of RAM memory:

- Removed all un-necessary parts of the mbedTLS Library (RC4, Eliptic Curves, TLS, SSL, BlowFish, DES/3DES, Diffie-Hellman, SHA1, SHA512, X509, md5, PKCSv21…) This reduced the size of the compiled code, leaving more RAM for calculations. (Since the code is also loaded into the RAM, taking up valuable space)
- Removed the use of file systems, found in mbedTLS. TI-RTOS doesn't produce a file system we want to work with.
- Used the boards own implementation of AES (see previous section)
- **Changed most cryptographic function to use Heap memory instead of Stack memory, making it more reusable across different tasks.**
- I reduced the size of the stack of the idle task.
- Reduced the number of priorites for tasks from 6 to 4 since I have 4 tasks
- Measured the maximum size of the stack of any task, and reduced the size of that tasks' stack to be the bare minimum required.
- I removed any un-necessary characteristics from the BLE stack which took up a lot of memory (Such as device information)

- Led blinking used a task to count the time. In order to save up memory space, I stop the blinking for the duration of the RSA calculation, thus saving up around 300 bytes of the tasks stack.

Challenge No. 5 – Debugging low memory environment:

In order to have the ability to print debug messages I changed the System.SupportProxy from SysCallback which does nothing to SysMin which supports a message queue. How ever this required 1KB of memory in order to sustain, so I have to disable this feature because I had no memory to spare. This made debugging quite difficult.

Attachments:

Code:

https://github.com/sonic21/IOTProjectBoard

https://github.com/sonic21/IOTProjectAPP