

Schedule

Exam date: (exam date to be defined)

Project presentations dates: to be defined (most probably in CW22 2024)

Final grade calculation: 30% project grade

70% exam grade

Project definition ICSS

1. Administrative topic:

- Project teams will be formed from up to maximum 4 members.
- Each team will have to implement the project requirements individually

2. Project definition:

Implement and integrate a module that calculates a hash using the assigned team algorithm over the button driver (first 2kB of the driver, memory range 0x2000 – 0x27FF). The module will assure the security over the system already implemented during the semester classes. The protection mechanism will have to work as follows: in case the protected driver gets modified, the system will have to detect the change and protect itself by closing the main functionality of the system. Once the security module detects a change in between the pre-calculated hash and the one calculated during runtime, the button detection and led toggling will have to stop working.

The system will perform (trigger) a hash calculation over the protected area every 10 seconds.

Details on the algorithm and the exact memory range that will be covered by the module can be found in this document in [Algorithms description](#) chapter and [Project hints](#) chapter. Please consult the list of teams in order to see what algorithm is assigned to each project team. Integrate the module into the CyberSecurity Proteus project.

The project will have to contain the following parts in order to be accepted and sustained:

1. Documentation: following parts are mandatory to be present in documentation:
 - a. Assigned algorithm presentation (in detail)
 - b. Complete set of specifications released
 - c. Complete architecture description
2. Complete Proteus project including source code for software firmware

3. A zip file containing all the source files from the project (.c source files and .h header files) extracted from Proteus project

All project teams will use the button driver provided together with this project requirements document. The driver provided to you is one of the drivers developed with one of the laboratory working groups from the proteus project. In case the provided driver will not work, you will have to adapt your project interfaces with other modules in order to integrate this button driver. **There are no changes allowed to be done on the provided button driver on your side. Making changes in the provided button driver will lead to a wrong hash value obtained over the memory block.**

When the project will be sustained, the project team will have to provide ONE printed version of the project documentation. Each project team will have to send by email (alexandru.costache@unitbv.ro) a zip file with the entire project with 3 days before the presentation date.

Please read with care all the project specifications / requirements / hints before starting the implementation.

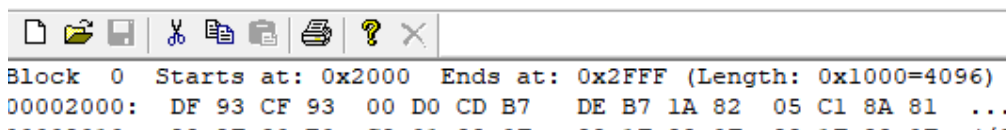
4. Algorithm example: SHA1 hash (160 bit output size)

Please pay attention on each algorithm specific parts like output size (specified in between brackets) and also on the inputs specific like padding when the input blocks are not a fixed multiple size specific to each algorithm.

I will use as an example in this document the SHA1 algorithm for the following chapters. **Please have in mind that each team will have to work / implement the corresponding hash algorithm.**

5. Project hints:

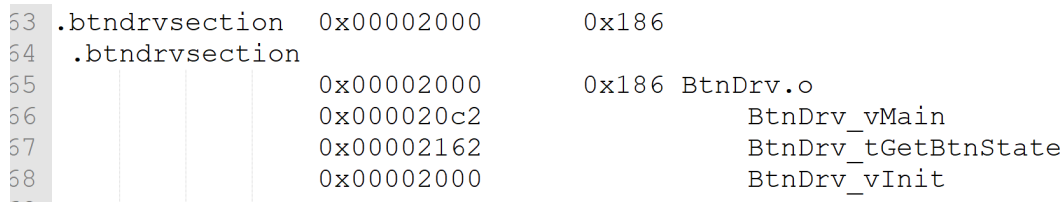
The button driver is located in flash in the reserved memory area **0x2000 – 0x27FF (2kB of flash)**:



```

Block 0 Starts at: 0x2000 Ends at: 0x27FF (Length: 0x1000=4096)
00002000: DF 93 CF 93 00 D0 CD B7 DE B7 1A 82 05 C1 8A 81 ...

```



```

53 .btndrvsection 0x00002000 0x186
54 .btndrvsection
55 0x00002000 0x186 BtnDrv.o
56 0x000020c2 BtnDrv_vMain
57 0x00002162 BtnDrv_tGetBtnState
58 0x00002000 BtnDrv_vInit

```

Example of BtnDrv functions details from map file

The hash (SHA1 hash in this example) will have to be calculated over the 2kB of memory of the button driver.

- start addresses 0x2000
- end address 0x27FF (last byte that needs to be part of the hash input is at address 0x27FF)
- length of the block 0x800

```

00002000: DF 93 CF 93 00 D0 CD B7 DE B7 1A 82 05 C1 8A 81
00002010: 28 2F 30 E0 C9 01 88 0F 99 1F 88 0F 99 1F 88 0F
00002020: 99 1F 82 1B 93 0B 01 96 88 0F 99 1F FC 01 E2 58
00002030: FF 4F A0 81 B1 81 8A 81 28 2F 30 E0 C9 01 88 0F
00002040: 99 1F 88 0F 99 1F 88 0F 99 1F 82 1B 93 0B 01 96
00002050: 88 0F 99 1F FC 01 E2 58 FF 4F 01 90 F0 81 E0 2D
00002060: 80 81 48 2F 8A 81 88 2F 90 E0 9C 01 22 0F 33 1F
00002070: C9 01 88 0F 99 1F 88 0F 99 1F 88 0F 99 1F 82 1B
00002080: 93 0B FC 01 EC 57 FF 4F 80 81 28 2F 30 E0 81 E0
00002090: 90 E0 02 C0 88 0F 99 1F 2A 95 E2 F7 80 95 84 23
000020A0: 8C 93 8A 81 88 2F 90 E0 9C 01 22 0F 33 1F C9 01
000020B0: 88 0F 99 1F 88 0F 99 1F 88 0F 99 1F 82 1B 93 0B
000020C0: FC 01 EE 57 FF 4F 01 90 F0 81 E0 2D 80 81 48 2F
000020D0: 8A 81 88 2F 90 E0 9C 01 22 0F 33 1F C9 01 88 0F
000020E0: 99 1F 88 0F 99 1F 88 0F 99 1F 82 1B 93 0B FC 01
000020F0: EC 57 FF 4F 80 81 28 2F 30 E0 81 E0 90 E0 02 2E
00002100: 02 C0 88 0F 99 1F 0A 94 E2 F7 84 23 89 83 89 81
00002110: 88 23 E9 F1 8A 81 88 2F 90 E0 9C 01 22 0F 33 1F
00002120: C9 01 88 0F 99 1F 88 0F 99 1F 88 0F 99 1F 82 1B
00002130: 93 0B FC 01 EB 57 FF 4F 11 82 10 82 8A 81 88 2F
00002140: 90 E0 9C 01 22 0F 33 1F C9 01 88 0F 99 1F 88 0F
00002150: 99 1F 88 0F 99 1F 82 1B 93 0B FC 01 E9 57 FF 4F
00002160: 11 82 10 82 8A 81 88 2F 90 E0 9C 01 22 0F 33 1F
00002170: C9 01 88 0F 99 1F 88 0F 99 1F 88 0F 99 1F 82 1B
00002180: 93 0B FC 01 E7 57 FF 4F 11 82 10 82 42 C0 8A 81
00002190: 88 2F 90 E0 9C 01 22 0F 33 1F C9 01 88 0F 99 1F
000021A0: 88 0F 99 1F 88 0F 99 1F 82 1B 93 0B FC 01 EB 57
000021B0: FF 4F 81 E0 90 E0 91 83 80 83 8A 81 88 2F 90 E0
000021C0: 9C 01 22 0F 33 1F C9 01 88 0F 99 1F 88 0F 99 1F
000021D0: 88 0F 99 1F 82 1B 93 0B FC 01 E9 57 FF 4F 81 E0
000021E0: 90 E0 91 83 80 83 8A 81 88 2F 90 E0 9C 01 22 0F
000021F0: 33 1F C9 01 88 0F 99 1F 88 0F 99 1F 88 0F 99 1F

```

Example of memory block

There is a predefined array **CySecStaticHash** with a length of 20 bytes (for SHA1 algorithm) located in flash memory. The **CySecStaticHash** array (defined in CySecDrv.c module) holds a pre-calculated hash value that will be used by the CySecDrv to validate the button driver during the execution. If the hash value calculated during runtime is different from the pre-calculated value stored in the **CySecStaticHash** array, then the application shall protect the LED's connected to the microcontroller by not switching them on/off (turn off the toggle functionality of the LED's when the hashes mismatch).

After each team will implement and calculate the hash value over the block requested, the hash value obtained will be sent by email to alexandru.costache@unitbv.ro for validation. Once the value will be validated by the teacher, then the content of the **CySecStaticHash** can be modified in code with the actual value of the hash.

Hints to help your development process:

- AVR ATMEGA32 microcontroller has a Hardware architecture, which means that the program flash and data flash have separate memory spaces. In order to correctly access the program flash addresses, you will have to use the `<avr/pgmspace.h>` functions, macros, like in the example below where the byte located at address 0x2000 is read from flash:

```
#include <avr/pgmspace.h>

uint8_t *address = (uint8_t *)0x2000;

static volatile uint8_t ucTest;

ucTest = pgm_read_byte(address);
```

ucTest variable will hold the data from program flash memory located at address 0x2000 after one execution. In order to cover the entire block, this reading will need to be integrated into a loop that will parse all the addresses from the requested memory range.

- All the hash related calculations and mechanism to trigger and collect the security status of the system must be provided by the CySecDrv module (CySecDrv.c and CySecDrv.h files). The module implementation, interface must be defined first in the architecture.
- Use the CyberSecurity Proteus project developed during the laboratories to work on this project. The project has already the correct definitions that will locate the button driver and the **CySecStaticHash** array to the correct program flash addresses.
- Main steps that should be followed:
 1. Implement and test flash memory read access
 2. Implement the specific hash algorithm into the CySecDrv module
 3. Test the CySecDrv module with a small string and compare the value obtained with the hash calculated via some online hash calculator
 4. Replace the test string with the program flash read access implemented at point 1 and calculate the hash over the program flash memory range 0x2000 – 0x27FF

Project teams definition

Project team handling and algorithms assignement will be done using the xls with the team assignements.

Info will be moved to the elearning platform (as a separated document) once all the teams are defined and each team has an algorithm assigned.