

# **Adaptation and Evaluation of a Post-Quantum Cryptographic Algorithm on a Resource-Constrained Device**

*Case Study: SQIsign on ARM Cortex-M4 Microcontroller*

# Main « Actors »

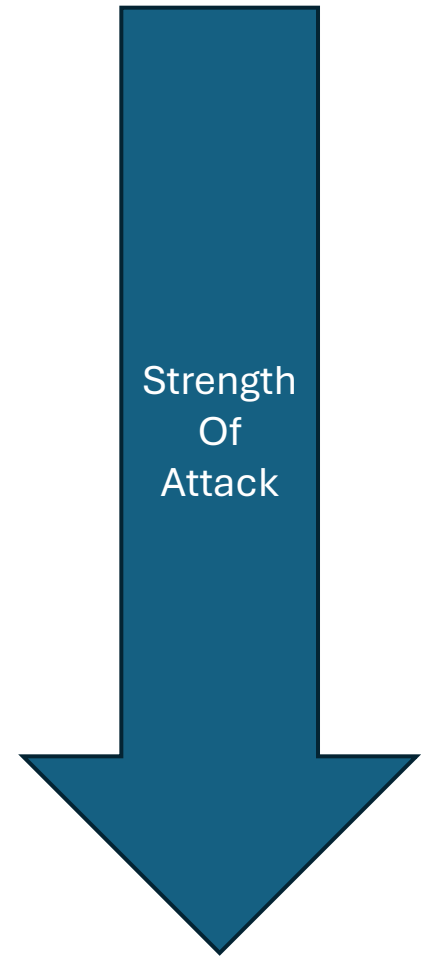
- SQLsign
- GMP
- ARM Cortex M4
- STM32

# SQLsign

- Digital signature scheme ( Based on elliptic curve, isogeny and quaternion)
- Participate NIST competition
- Need to give example for level security 1,3 and 5

# Levels of security required by NIST

- 1) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g., AES-128)
- 2) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 256-bit hash function (e.g., SHA-256/ SHA3-256)
- 3) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g., AES-192)
- 4) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 384-bit hash function (e.g., SHA-384/ SHA3-384)
- 5) Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g., AES-256)



# GMP

- “GNU Multiple Precision Arithmetic Library”
  - Use for multiple precision to compute
  - Use for the calculation inside Quaternions part

# ARM Cortex M4

- TODO

# STM32

- TODO

# State of the Art

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQISIGN on the ARM Cortex-M4 core is the pqm4 project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with pqm4, is provided in the submission package; instructions for its use can be found in the README.md file.

Source : <https://sqisign.org/spec/sqisign-20250205.pdf>



# State of the Art

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQISIGN on the ARM Cortex-M4 core is the pqm4 project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with pqm4, is provided in the submission package; instructions for its use can be found in the README.md file.

Source : <https://sqisign.org/spec/sqisign-20250205.pdf>

# State of the Art

The verification routines of the reference implementation are compatible with 32-bit embedded systems based on e.g., the ARM Cortex-M4 core. However, the build system and provided applications are not suitable for a bare-metal environment as typically provided in such systems. The recommended platform for evaluating SQISIGN on the ARM Cortex-M4 core is the pqm4 project [KPR+]. A helper shell script, to copy the source files to a folder structure compatible with pqm4, is provided in the submission package; instructions for its use can be found in the README.md file.

Source : <https://sqisign.org/spec/sqisign-20250205.pdf>

# Experimentation

- Phase 1 : Adaptation
- Phase 2 : Evaluate SQLsign
- Phase 3 : Implementation of a solution
- Phase 4 : The Bug
- Phase 5 : Evaluate SQLsign with all level of difficulties
- Phase 6 : Data gathering

# Phase 1

- Update build system for bare-metal environment
- Update applications for bare-metal environment

[illegible]

# SQLsign (version 2) : all files

# Update build system for bare-metal environment

- Example of the works for verification
  - All files has been flattened
  - No more Cmake

<a href="#">the-squigs-pkg / crypto_sign / xsquig_h1 / ref</a>	
24 people · verification-only SQuig round 2 version 1	
Name	Last commit message
<code>.</code>	
<code>.apph</code>	Verification-only SQuig (INST round 2 version)
<code>.basic</code>	Verification-only SQuig (INST round 2 version)
<code>.commenc</code>	Verification-only SQuig (INST round 2 version)
<code>.config.mk</code>	Verification-only SQuig (INST round 2 version)
<code>.ed_basic</code>	Verification-only SQuig (INST round 2 version)
<code>.ed_basic.h</code>	Verification-only SQuig (INST round 2 version)
<code>.ec.c</code>	Verification-only SQuig (INST round 2 version)
<code>.ech</code>	Verification-only SQuig (INST round 2 version)
<code>.ec_jacc</code>	Verification-only SQuig (INST round 2 version)
<code>.ec_params.c</code>	Verification-only SQuig (INST round 2 version)
<code>.ec_params.h</code>	Verification-only SQuig (INST round 2 version)
<code>.encode_verification</code>	Verification-only SQuig (INST round 2 version)
<code>.encoded_catch</code>	Verification-only SQuig (INST round 2 version)
<code>.fp.c</code>	Verification-only SQuig (INST round 2 version)
<code>.fph</code>	Verification-only SQuig (INST round 2 version)
<code>.fpdc</code>	Verification-only SQuig (INST round 2 version)
<code>.fpdh</code>	Verification-only SQuig (INST round 2 version)
<code>.fp_constants.h</code>	Verification-only SQuig (INST round 2 version)
<code>.fp52SHE_Sc.c</code>	Verification-only SQuig (INST round 2 version)
<code>.hdc</code>	Verification-only SQuig (INST round 2 version)
<code>.hdh</code>	Verification-only SQuig (INST round 2 version)
<code>.hd_writing_transform.c</code>	Verification-only SQuig (INST round 2 version)
<code>.hd_writing_transform.h</code>	Verification-only SQuig (INST round 2 version)
<code>.high</code>	Verification-only SQuig (INST round 2 version)
<code>.hog_chars.c</code>	Verification-only SQuig (INST round 2 version)
<code>.mpc</code>	Verification-only SQuig (INST round 2 version)
<code>.mph</code>	Verification-only SQuig (INST round 2 version)
<code>.perm_f_abc</code>	Verification-only SQuig (INST round 2 version)
<code>.pmh</code>	Verification-only SQuig (INST round 2 version)
<code>.sph</code>	Verification-only SQuig (INST round 2 version)
<code>.sqsig.c</code>	Verification-only SQuig (INST round 2 version)
<code>.sqsig_namespaces</code>	Verification-only SQuig (INST round 2 version)
<code>.theta_logsmisc</code>	Verification-only SQuig (INST round 2 version)
<code>.theta_logsmish</code>	Verification-only SQuig (INST round 2 version)
<code>.theta_structmisc</code>	Verification-only SQuig (INST round 2 version)
<code>.theta_structsh</code>	Verification-only SQuig (INST round 2 version)
<code>.tosh</code>	Verification-only SQuig (INST round 2 version)
<code>.tush</code>	Verification-only SQuig (INST round 2 version)
<code>.verification</code>	Verification-only SQuig (INST round 2 version)
<code>.verity.c</code>	Verification-only SQuig (INST round 2 version)
<code>.welic</code>	Verification-only SQuig (INST round 2 version)
<code>.wlopc</code>	Verification-only SQuig (INST round 2 version)

Source : [https://github.com/SQISign/the-sqisign-pqm4/tree/SQISign/crypto\\_sign/sqisign\\_lvl1/ref](https://github.com/SQISign/the-sqisign-pqm4/tree/SQISign/crypto_sign/sqisign_lvl1/ref)

# Update build system for bare-metal environment

- Main Goal of the SQIsign team is to adapt their code for PQM4
  - PQM4 :
    - Collection of post-quantum cryptographic algorithms for the ARM Cortex-M4
    - benchmarking and testing framework
- My Goal is to adapt to STM32CubeIDE

=> The same steps

# What are the implications of not using current build system ?

- No more automatization in term of variable selection
  - Manually add to the code
- Need duplicate the work by 3 ( for level security 1,3 and 5) select files dependent of the level security
  - Manual selection
- Need to select file manually between 32 bits and 64 bits architecture
- Need to be carefull to not select file optimize for Broadwell
- ...



# Exemple\_nistapi.c

- Basic file of SQLsign project which contain :
  - Key Generation
  - Signature
  - Verification
  - Small tests to evaluate each result

# Update build system for bare-metal environment

- Methodology
  - Begin with an empty makefile
  - Use `exemple_nistapi.c` as base file
    - Try to compile
    - Get an error
    - Solve it



# Case 1

- Undefined function => grep and find the function => add file to makefile
  - Main issue => “sqisign\_namespace.h”

```
#define PARAM_JOIN3(a, b, c) sqisign_##a##_##b##_##c  
#define PARAM_JOIN3(a, b, c) PARAM_JOIN3_(a, b, c)  
#define PARAM_NAME3(end, s) PARAM_JOIN3(SQISIGN_VARIANT, end, s)
```

} Rename function



Grep does not work anymore

# Case 2

- Selection of variable :
  - Reverse the code to make a choice
  - Read “Algorithm specifications and supporting documentation”

```
#if RADIX == 32
#if defined(SQISIGN_GF_IMPL_BROADWELL)
#define NWORDS_FIELD 8
#else
#define NWORDS_FIELD 9
#endif
#define NWORDS_ORDER 8
#elif RADIX == 64
#if defined(SQISIGN_GF_IMPL_BROADWELL)
#define NWORDS_FIELD 4
#else
#define NWORDS_FIELD 5
#endif
#define NWORDS_ORDER 4
#endif
#define BITS 256
#define LOG2P 8
```

Exemple inside the file fp\_constants.h

```
#if defined(ENABLE_SIGN)
SQISIGN_API
int
crypto_sign_keypair(unsigned char *pk, unsigned char *sk);

SQISIGN_API
int
crypto_sign(unsigned char *sm, unsigned long long *smlen,
            const unsigned char *m, unsigned long long mlen,
            const unsigned char *sk);
#endif

SQISIGN_API
int
crypto_sign_open(unsigned char *m, unsigned long long *mlen,
                const unsigned char *sm, unsigned long long smlen,
                const unsigned char *pk);
```

Exemple inside the file api.h

# Case 2

```
#if defined(STM32)
    #include "macro_m4.h"
#endif
```



```
#define MINI_GMP
#define RADIX_32
#define GMP_LIMB_BITS 32
#define RADIX 32
#define RANDOMBYTES_SYSTEM
#define ENABLE_SIGN

// #define NDEBUG
// #define DEBUG_VERBOSE
// #define MINI_GMP_DONT_USE_FLOAT_H
```

# Case 3

- Change `#include <file>` => `#include "file"` when a file is related to the project SQIsign
  - STM32 need that for `/Core/Inc` and `/Core/Src`
  - SQIsign did the same thing for the adaptation of verification for PQM4

```
#include <inttypes.h>
#include <mem.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <time.h>

#include <api.h>
#include <rng.h>
#include <bench_test_arguments.h>
#if defined(TARGET_BIG_ENDIAN)
#include <tutil.h>
#endif
```



```
#include <inttypes.h>
#include "mem.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <time.h>

#include "api.h"
#include "rng.h"
#include "bench_test_arguments.h"
#if defined(TARGET_BIG_ENDIAN)
#include "tutil.h"
#endif
```

Exemple inside the file `example_nistapi.c`

# Special case

- GMP has 2 version :
  - GMP : normal version
  - GMP mini : smaller version => better for constrain device

# Update applications for bare-metal environment

- To solve that => static reading of the code and update it with the HAL
  - Tools.c => use of the clock
  - Randombytes\_system.c => use of random



# Difficulties

- Not especially hard but :
  - Time consuming
    - Come to a big project
    - Lack of comment
    - Lack of Knowledge about code of SQLsign
    - sqisign\_namespace.h
    - Some Bad choice led to redo some parts of the work
  - No real notion of bare metal environment at the beginning
  - Lack of information related to the issues in state of the art

# Phase 2 : evaluate SQIsign

NUCLEO-L4R5ZI

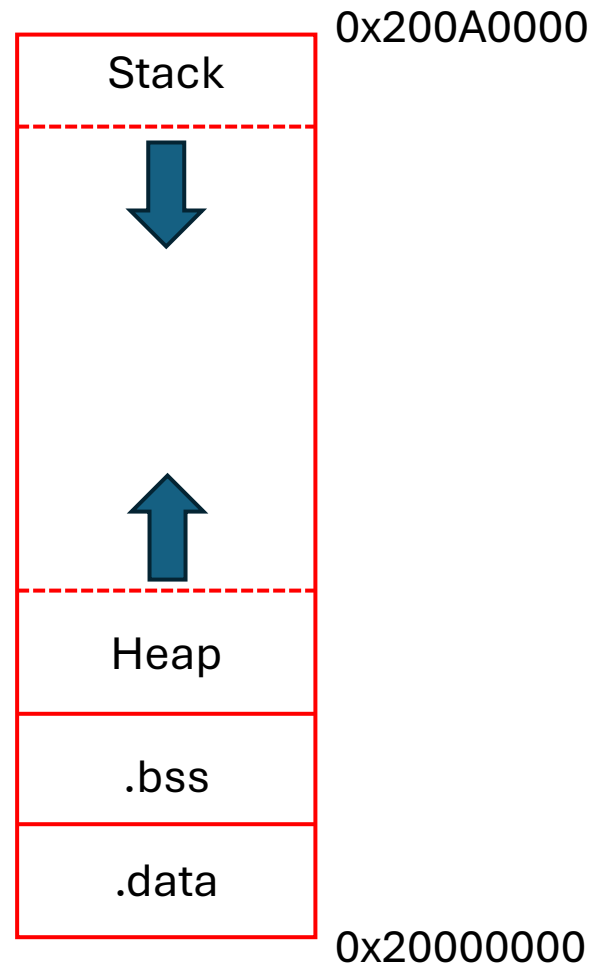


- 2 MB of flash memory
- 640 KB of SRAM
- Selected by SQIsign team

# Run the code

- The code crash
  - Ram does not have enough resources
  - Stack grow too much
- Test with a “canary”
  - Technique use in cybersecurity => « something that does not change »
    - Set a global variable initialize with a value, and never touch it during the execution
      - If it change ( => stack overflow because we are in .data section)

# How RAM work



# What I did ?

The image shows a debugger window with two main panes. The left pane displays a C source file named `dim2id2iso.c` with line numbers 558 to 615. The right pane shows the `Registers` window, listing general registers `r0` through `r15` and the stack pointer `sp`. The `sp` register is highlighted in blue and contains the value `0x2000e268`. A red arrow points from this value to a specific line in the code, labeled "Break point". A red bracket on the right side of the code, labeled "Problematic part", encompasses lines 581 through 615.

```
558 quat_left_ideal_init(&conj_ideal);
559 quat_ideal_conjugate_without_hnf(&conj_ideal, &right_order, &reduced_id, Bpoo);
560
561 // computing all the other connecting ideals and reducing them
562 for (int i = 1; i < num_alternate_order + 1; i++) {
563     quat_ideal_ideal_mul_reduced(&ideal[i], &conj_ideal, &ALTERNATE_CONNECTING_IDEALS[i - 1], Bpoo);
564     ibz_mat_4x4_copy(&reduced[i], &ideal[i].lattice.basis);
565     ibz_set(&adjusted_norm[i], 1);
566     ibz_mul(&adjusted_norm[i], &adjusted_norm[i], &ideal[i].lattice.denom);
567     ibz_mul(&adjusted_norm[i], &adjusted_norm[i], &ideal[i].lattice.denom);
568     post_llt_basis_treatment(&gram[i], &reduced[i], &ideal[i].norm, false);
569 }
570
571 // enumerating small vectors
572 // global parameters for the enumeration
573 int m = FINDUV_box_size;
574 int m4 = FINDUV_cube_size;
575
576 ibz_vec_4_t small_vecs[num_alternate_order + 1][m4];
577 ibz_t small_norms[num_alternate_order + 1][m4];
578 ibz_t alternate_small_norms[num_alternate_order + 1][m4];
579 ibz_t quotients[num_alternate_order + 1][m4];
580 int indices[num_alternate_order + 1];
581
582 for (int j = 0; j < num_alternate_order + 1; j++) {
583     for (int i = 0; i < m4; i++) {
584         ibz_init(&small_vecs[j][i]);
585         ibz_init(&alternate_small_norms[j][i]);
586         ibz_init(&quotients[j][i]);
587         ibz_vec_4_init(&alternate_small_vecs[j][i]);
588     }
589     // enumeration in the hypercube of norm m
590     indices[j] = enumerate_hypercube(small_vecs[j], small_norms[j], m, &gram[j], &adjusted_norm[j]);
591     // sorting the list
592     {
593         struct vec_and_norm small_vecs_and_norms[indices[j]];
594         for (int i = 0; i < indices[j]; ++i) {
595             memcpy(&small_vecs_and_norms[i].vec, &small_vecs[j][i], sizeof(ibz_vec_4_t));
596             memcpy(&small_vecs_and_norms[i].norm, &small_norms[j][i], sizeof(ibz_t));
597             small_vecs_and_norms[i].idx = i;
598         }
599         qsort(small_vecs_and_norms, indices[j], sizeof(*small_vecs_and_norms), compare_vec_by_norm);
600         for (int i = 0; i < indices[j]; ++i) {
601             memcpy(&small_vecs[j][i], &small_vecs_and_norms[i].vec, sizeof(ibz_vec_4_t));
602             memcpy(&small_norms[j][i], &small_vecs_and_norms[i].norm, sizeof(ibz_t));
603         }
604     }
605 #ifndef NDEBUG
606     for (int i = 1; i < indices[j]; ++i)
607         assert(ibz_cmp(&small_norms[j][i - 1], &small_norms[j][i]) <= 0);
608 #endif
609 }
610
611 for (int i = 0; i < indices[j]; ++i) {
612     ibz_div(&quotients[j][i], &remain, &n, &small_norms[j][i]);
613 }
614 }
615
616 int found = 0;
```

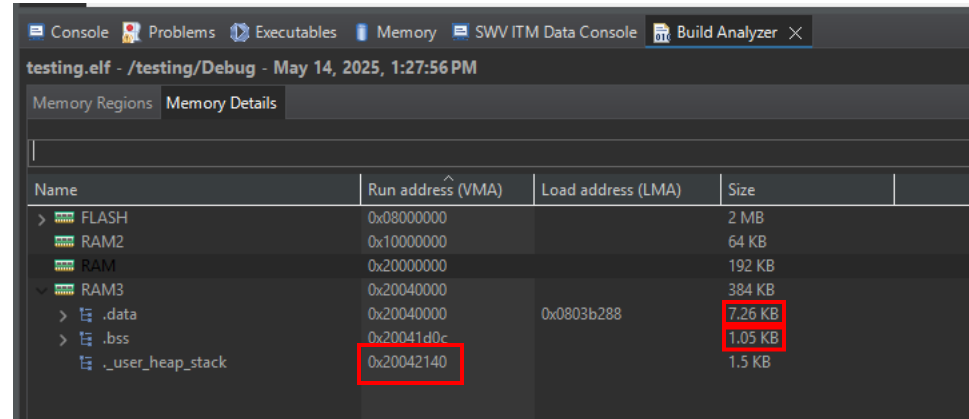
Registers window:

Name	Value	Descr
r0	7	
r1	7	
r2	224	
r3	536928920	
r4	7	
r5	0	
r6	0	
r7	537509848	
r8	7	
r9	7488	
r10	7488	
r11	29952	
r12	2146892541	
sp	0x2000e268	
lr	134244787	
pc	0x80072d6 <find_uv+4342>	
xpsr	16777216	
d0	0.78252218686748642	
d1	0	
d2	0	
d3	0	
d4	0	
d5	0	
d6	0	
d7	0.78252218686748642	
d8	0	
d9	0	
d10	0	
d11	0	
d12	0	
d13	0	
d14	0	
d15	0	

Register details for `sp`:

Name: sp  
Hex: 0x2000e268  
Decimal: 536928872  
Octal: 04000161150  
Binary: 100000000000001110001001101000  
Default: 0x2000e268

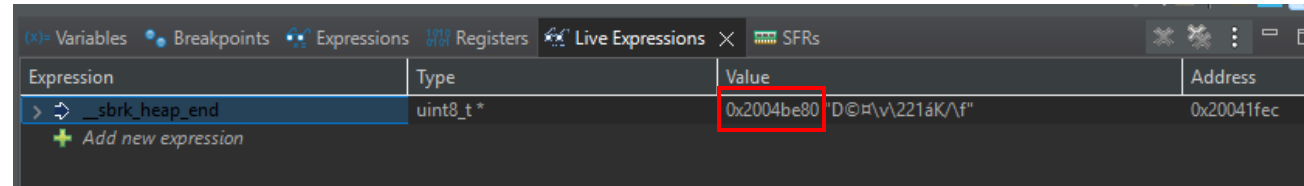
# What I did ?



testing.elf - /testing/Debug - May 14, 2025, 1:27:56 PM

Memory Regions Memory Details

Name	Run address (VMA)	Load address (LMA)	Size
> FLASH	0x08000000		2 MB
> RAM2	0x10000000		64 KB
> RAM	0x20000000		192 KB
> RAM3	0x20040000		384 KB
> .data	0x20040000	0x0803b288	7.26 KB
> .bss	0x20041d0c		1.05 KB
> ._user_heap_stack	0x20042140		1.5 KB



(x)- Variables Breakpoints Expressions Registers Live Expressions SFRs

Expression	Type	Value	Address
> __sbrk_heap_end	uint8_t *	0x2004be80	0x20041fec

+ Add new expression

# What I did ?

- Current SP= 0x2000e268
  - Start of SP (End of SRAM) = 0x200A0000
  - Current Heap pointer = 0x2004be80
  - Start of Heap = 0x20042140
- 
- STACK size = Start of SP - Current SP = 583.39 KB
  - HEAP “size” = Current Heap pointer - Start of Heap = 39.31 KB
  - .bss + .data + HEAP + STACK = ~ 631 KB ( out of 640 KB provide by board )

---

**Algorithm 4.1**  $\text{SQIsign.KeyGen}()$ 

---

**Output:** Secret key  $\text{sk}$  and public key  $\text{pk}$ .

```
1: while true do
2:    $I_{\text{sk}} \leftarrow \text{RandomIdealGivenNorm}(D_{\text{mix}}, \text{true})$ 
3:   try
4:      $I_{\text{sk}} \leftarrow \text{RandomEquivalentPrimeIdeal}(I_{\text{sk}})$ 
5:      $E_{\text{pk}}, \varphi_{\text{sk}}(P_0), \varphi_{\text{sk}}(Q_0) \leftarrow \text{IdealTolsogeny}(I_{\text{sk}})$ 
6:   except
7:     continue
8:    $P_{\text{pk}}, Q_{\text{pk}}, \text{hint}_{\text{pk}} \leftarrow \text{TorsionBasisToHint}(E_{\text{pk}})$ 
9:    $\mathbf{M}_{\text{sk}} \leftarrow \text{ChangeOfBasis}_{2^f}(E_{\text{pk}}, (\varphi_{\text{sk}}(P_0), \varphi_{\text{sk}}(Q_0)), (P_{\text{pk}}, Q_{\text{pk}}))$ 
10:   $\text{pk} \leftarrow (E_{\text{pk}}, \text{hint}_{\text{pk}})$ 
11:   $\text{sk} \leftarrow (E_{\text{pk}}, \text{hint}_{\text{pk}}, I_{\text{sk}}, \mathbf{M}_{\text{sk}})$ 
12: return  $\text{sk}, \text{pk}$ 
```

---

---

**Algorithm 3.16**  $\text{SuitableIdeals}(I)$ 

---

**Input:** A left  $\mathcal{O}_0$ -ideal  $I$ .

**Output:** Positive integers  $u, v, e$ , indices  $s, t$ , and  $\beta_1 \in \overline{J_s}I, \beta_2 \in \overline{J_t}I$  such that  $ud_1 + vd_2 = 2^e$ ,  $\gcd(ud_1, vd_2) = 1$ , and  $e \leq f$ , where  $d_1 = \text{nrd}(\beta_1) / \text{nrd}(\overline{J_s}I), d_2 = \text{nrd}(\beta_2) / \text{nrd}(\overline{J_t}I)$ .

```
1: for  $t$  from 0 up to  $n_{\text{orders}}$  do
2:   Let  $(b_1, \dots, b_4)$  be a basis of  $\overline{J_t}I$ 
3:   Compute the Gram matrix  $\mathbf{G} = (\langle b_r, b_s \rangle)_{1 \leq r, s \leq 4}$ 
4:    $(\alpha_1, \dots, \alpha_4), \_ \leftarrow \text{L2}_{\delta, \eta}((b_1, \dots, b_4), \mathbf{G})$ 
5:    $L_t \leftarrow \left[ \sum_{s=1}^4 x_s \alpha_s \mid (x_1, \dots, x_4) \in [-m, \dots, m] \right]$  for  $m = \text{FINDUV\_box\_size}$ 
6:   Sort  $L_t$  by increasing norm.
7:   for  $(s, t) \in [0, \dots, n_{\text{orders}}]^2$  do                                     // by increasing lexicographic order
8:     for  $(\beta_1, \beta_2) \in L_s \times L_t$  do                                     // by increasing lexicographic order
9:        $d_1 \leftarrow \text{nrd}(\beta_1) / \text{nrd}(\overline{J_s}I)$  and  $d_2 \leftarrow \text{nrd}(\beta_2) / \text{nrd}(\overline{J_t}I)$ 
10:      if  $d_1 \equiv 1 \pmod{2}$  and  $d_2 \equiv 1 \pmod{2}$  and  $\gcd(d_1, d_2) = 1$  then
11:         $u \leftarrow 2^f d_1^{-1} \pmod{d_2}$ 
12:         $v \leftarrow (2^f - ud_1) / d_2$ 
13:        if  $v > 0$  then
14:           $e \leftarrow \text{DyadicValuation}(u)$ 
15:           $u \leftarrow u / 2^e, v \leftarrow v / 2^e$ , and  $e \leftarrow f - e$ 
16:        return  $u, v, e, s, t, \beta_1, \beta_2$ 
17: raise Exception(“SuitableIdeals failed”)
```

---

---

**Algorithm 3.13**  $\text{IdealTolsogeny}(I)$ 

---

**Input:** A left  $\mathcal{O}_0$ -ideal  $I$ .

**Output:** The codomain  $E_I$  of  $\varphi_I, \varphi_I(P_0)$ , and  $\varphi_I(Q_0)$ .

```
1:  $u, v, e, s, t, \beta_1, \beta_2 \leftarrow \text{SuitableIdeals}(I)$ 
2:  $d_1 \leftarrow \text{nrd}(\beta_1) / \text{nrd}(\overline{J_s}I)$ 
3:  $d_2 \leftarrow \text{nrd}(\beta_2) / \text{nrd}(\overline{J_t}I)$                                      //  $ud_1 + vd_2 = 2^e$ , with  $\gcd(ud_1, vd_2) = 1$ 
4:  $E_u, \varphi_u(P_s), \varphi_u(Q_s) \leftarrow \text{FixedDegreelsogeny}(s, u)$            //  $\varphi_u : E_s \rightarrow E_u$  is an isogeny of degree  $u$ 
5:  $E_v, \varphi_v(P_t), \varphi_v(Q_t) \leftarrow \text{FixedDegreelsogeny}(t, v)$        //  $\varphi_v : E_t \rightarrow E_v$  is an isogeny of degree  $v$ 
6:  $[P, Q]^T \leftarrow \left[ \frac{1}{\text{nrd}(I) \text{nrd}(\overline{J_t}I)} \right] \mathbf{M}_{\overline{\beta_1}} \mathbf{M}_{\beta_2} [\varphi_v(P_t), \varphi_v(Q_t)]^T$ 
7:  $K_P \leftarrow [2^{f-e}]([d_1] \varphi_u(P_s), P)$ 
8:  $K_Q \leftarrow [2^{f-e}]([d_1] \varphi_u(Q_s), Q)$ 
9:  $E \times E', [(P, P'), (Q, Q')] \leftarrow \text{Isogeny22Chain}(K_P, K_Q, [(\varphi_u(P_s), 0_{E_v}), (\varphi_u(Q_s), 0_{E_v})])$ 
10: if  $e_{2^f}(P, Q) = e_{2^f}(P_s, Q_s)^{u^2 d_1}$  then                         // We identify the right curve  $E_I$  by ensuring it is  $d_1$ -isogeneous to  $E_0$ 
11:    $E_I \leftarrow E, P_I \leftarrow P$ , and  $Q_I \leftarrow Q$ 
12: else
13:    $E_I \leftarrow E', P_I \leftarrow P'$ , and  $Q_I \leftarrow Q'$ 
14:  $[P_I, Q_I]^T \leftarrow \left[ \frac{1}{ud_1} \right] \mathbf{M}_{\beta_1} [P_I, Q_I]^T$ 
15: return  $E_I, (P_I, Q_I)$ 
```

---



---

**Algorithm 4.1**  $\text{SQIsign.KeyGen}()$ 

---

**Output:** Secret key  $\text{sk}$  and public key  $\text{pk}$ .

```
1: while true do
2:    $I_{\text{sk}} \leftarrow \text{RandomIdealGivenNorm}(D_{\text{mix}}, \text{true})$ 
3:   try
4:      $I_{\text{sk}} \leftarrow \text{RandomEquivalentPrimeIdeal}(I_{\text{sk}})$ 
5:      $E_{\text{pk}}, \varphi_{\text{sk}}(P_0), \varphi_{\text{sk}}(Q_0) \leftarrow \text{IdealToIsogeny}(I_{\text{sk}})$ 
6:   except
7:     continue
8:    $P_{\text{pk}}, Q_{\text{pk}}, \text{hint}_{\text{pk}} \leftarrow \text{TorsionBasisToHint}(E_{\text{pk}})$ 
9:    $\mathbf{M}_{\text{sk}} \leftarrow \text{ChangeOfBasis}_{2^f}(E_{\text{pk}}, (\varphi_{\text{sk}}(P_0), \varphi_{\text{sk}}(Q_0)), (P_{\text{pk}}, Q_{\text{pk}}))$ 
10:   $\text{pk} \leftarrow (E_{\text{pk}}, \text{hint}_{\text{pk}})$ 
11:   $\text{sk} \leftarrow (E_{\text{pk}}, \text{hint}_{\text{pk}}, I_{\text{sk}}, \mathbf{M}_{\text{sk}})$ 
12:  return  $\text{sk}, \text{pk}$ 
```

---

---

**Algorithm 3.13**  $\text{IdealToIsogeny}(I)$ 

---

**Input:** A left  $\mathcal{O}_0$ -ideal  $I$ .

**Output:** The codomain  $E_I$  of  $\varphi_I, \varphi_I(P_0)$ , and  $\varphi_I(Q_0)$ .

```
1:  $u, v, e, s, t, \beta_1, \beta_2 \leftarrow \text{SuitableIdeals}(I)$ 
2:  $d_1 \leftarrow \text{nrd}(\beta_1) / \text{nrd}(\overline{J_s}I)$ 
3:  $d_2 \leftarrow \text{nrd}(\beta_2) / \text{nrd}(\overline{J_t}I)$ 
4:  $E_u, \varphi_u(P_s), \varphi_u(Q_s) \leftarrow \text{FixedDegreelsogeny}(s, u)$ 
5:  $E_v, \varphi_v(P_t), \varphi_v(Q_t) \leftarrow \text{FixedDegreelsogeny}(t, v)$ 
6:  $[P, Q]^T \leftarrow \left[ \frac{1}{\text{nrd}(I) \text{nrd}(\overline{J_t}I)} \right] \mathbf{M}_{\overline{\beta_1}} \mathbf{M}_{\beta_2} [\varphi_v(P_t), \varphi_v(Q_t)]^T$ 
7:  $K_P \leftarrow [2^{f-e}]([d_1]\varphi_u(P_s), P)$ 
8:  $K_Q \leftarrow [2^{f-e}]([d_1]\varphi_u(Q_s), Q)$ 
9:  $E \times E', [(P, P'), (Q, Q')] \leftarrow \text{Isogeny22Chain}(K_P, K_Q, [(\varphi_u(P_s), 0_{E_v}), (\varphi_u(Q_s), 0_{E_v})])$ 
10: if  $e_{2^f}(P, Q) = e_{2^f}(P_s, Q_s)^{u^2 d_1}$  then
11:    $E_I \leftarrow E, P_I \leftarrow P, \text{ and } Q_I \leftarrow Q$ 
12: else
13:    $E_I \leftarrow E', P_I \leftarrow P', \text{ and } Q_I \leftarrow Q'$ 
14:  $[P_I, Q_I]^T \leftarrow \left[ \frac{1}{u d_1} \right] \mathbf{M}_{\beta_1} [P_I, Q_I]^T$ 
15: return  $E_I, (P_I, Q_I)$ 
```

---

---

**Algorithm 3.16**  $\text{SuitableIdeals}(I)$ 

---

**Input:** A left  $\mathcal{O}_0$ -ideal  $I$ .

**Output:** Positive integers  $u, v, e$ , indices  $s, t$ , and  $\beta_1 \in \overline{J_s}I, \beta_2 \in \overline{J_t}I$  such that  $u d_1 + v d_2 = 2^e$ ,  $\gcd(u d_1, v d_2) = 1$ , and  $e \leq f$ , where  $d_1 = \text{nrd}(\beta_1) / \text{nrd}(\overline{J_s}I), d_2 = \text{nrd}(\beta_2) / \text{nrd}(\overline{J_t}I)$ .

```
1: for  $t$  from 0 up to  $n_{\text{orders}}$  do
2:   Let  $(b_1, \dots, b_4)$  be a basis of  $\overline{J_t}I$ 
3:   Compute the Gram matrix  $\mathbf{G} = (\langle b_r, b_s \rangle)_{1 \leq r, s \leq 4}$ 
4:    $(\alpha_1, \dots, \alpha_4), - \leftarrow \text{L2}_{\delta, \eta}((b_1, \dots, b_4), \mathbf{G})$ 
5:    $L_t \leftarrow \left[ \sum_{s=1}^4 x_s \alpha_s \mid (x_1, \dots, x_4) \in [-m, \dots, m] \right]$  for  $m = \text{FINDUV\_box\_size}$ 
6:   Sort  $L_t$  by increasing norm.
7:   for  $(s, t) \in [0, \dots, n_{\text{orders}}]^2$  do
8:     for  $(\beta_1, \beta_2) \in L_s \times L_t$  do
9:        $d_1 \leftarrow \text{nrd}(\beta_1) / \text{nrd}(\overline{J_s}I)$  and  $d_2 \leftarrow \text{nrd}(\beta_2) / \text{nrd}(\overline{J_t}I)$ 
10:      if  $d_1 \equiv 1 \pmod{2}$  and  $d_2 \equiv 1 \pmod{2}$  and  $\gcd(d_1, d_2) = 1$  then
11:         $u \leftarrow 2^f d_1^{-1} \pmod{d_2}$ 
12:         $v \leftarrow (2^f - u d_1) / d_2$ 
13:        if  $v > 0$  then
14:           $e \leftarrow \text{DyadicValuation}(u)$ 
15:           $u \leftarrow u / 2^e, v \leftarrow v / 2^e, \text{ and } e \leftarrow f - e$ 
16:        return  $u, v, e, s, t, \beta_1, \beta_2$ 
17: raise Exception("SuitableIdeals failed")
```

---

# Difficulties

- Have enough skills thanks to my studies in cybersecurity with heap, stack, ... but :
  - New tool ( => debugger STM32CubeIDE)
  - Code run really slowly, approximative 5 minutes to reach the point ( => run multiple time to understand the situation)
  - Make work Printf

There is another way to evaluate  
SQLsign and potentially see if all  
my adaptation works ?

# Phase 3


Careers Sample & buy Support & community 日本語 中文 English

ST Products Tools & software Applications Solutions **ST Developer Zone** About us


STM32 MCU Developer Zone Jump start MCU Portfolio Boards and hardware tools Software development tools Embedded software Solutions Developer resources MCU Community

STM32 Developer Zone > STM32 MCU Developer Zone > Start a project with an MCU


## Start a project with an STM32 microcontroller




Step 1: Select your board Evaluation board, expansion boards



**Nucleo boards**  
Essential evaluation and prototyping



**STM32 Discovery kits**  
Feature-rich evolution and prototyping



**STM32 Eval boards**  
Full-feature evaluation

Feedback

[https://www.st.com/content/st\\_com/en/stm32-mcu-developer-zone/start-a-project-with-an-mcu.html](https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/start-a-project-with-an-mcu.html)

# Phase 3

## NUCLEO-L4R5ZI



<https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html>

## STM32F429I-DISC1



<https://www.st.com/en/evaluation-tools/32f429idiscovery.html>

# Phase 3

## **NUCLEO-L4R5ZI**

- 2 MB of flash memory
- 640 KB of SRAM

## **STM32F429I-DISC1**

- 2 MB of flash memory
- 256 KB of SRAM
- 8 MB of SDRAM  
(=> frame buffer for LCD screen)

Why not use the SDRAM as base  
for my stack ?

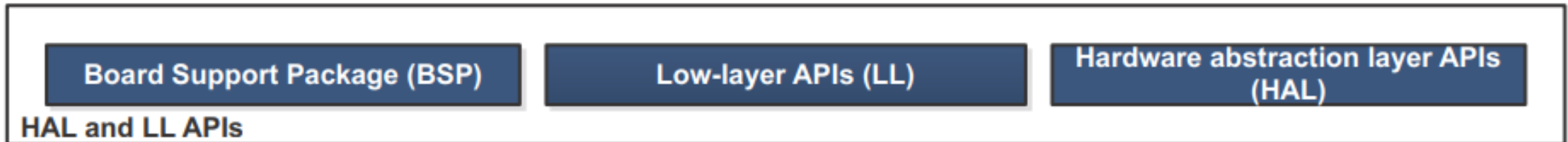
# Phase 3

- Step 1 : initialize the SDRAM
- Step 2 : move the stack



# Initialization of the SDRAM

Not so simple, not like « plug and play » when we have OS

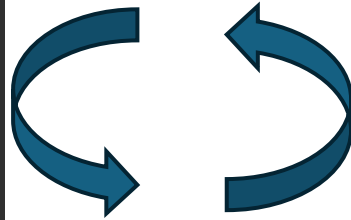



[https://www.st.com/resource/en/user\\_manual/um2298-stm32cube-bsp-drivers-development-guidelines-stmicroelectronics.pdf](https://www.st.com/resource/en/user_manual/um2298-stm32cube-bsp-drivers-development-guidelines-stmicroelectronics.pdf)

# Move the stack

```
/* Highest address of the user mode stack */
_estack = ORIGIN(RAM) + LENGTH(RAM);

/* Memories definition */
MEMORY
{
  CCMRAM (xrw) : ORIGIN = 0x10000000, LENGTH = 64K
  RAM      (xrw) : ORIGIN = 0x20000000, LENGTH = 192K
  FLASH   (rx)  : ORIGIN = 0x8000000, LENGTH = 2048K
  SDRAM   (xrw) : ORIGIN = 0xD0000000, LENGTH = 8192K
}
```



```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration-----*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */


  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  //MX_FMC_Init();
  BSP_SDRAM_Init();
  MX_TIM1_Init();
  MX_USART1_UART_Init();
  MX_RNG_Init();
  MX_I2C3_Init();
  MX_SPI5_Init();
}
```

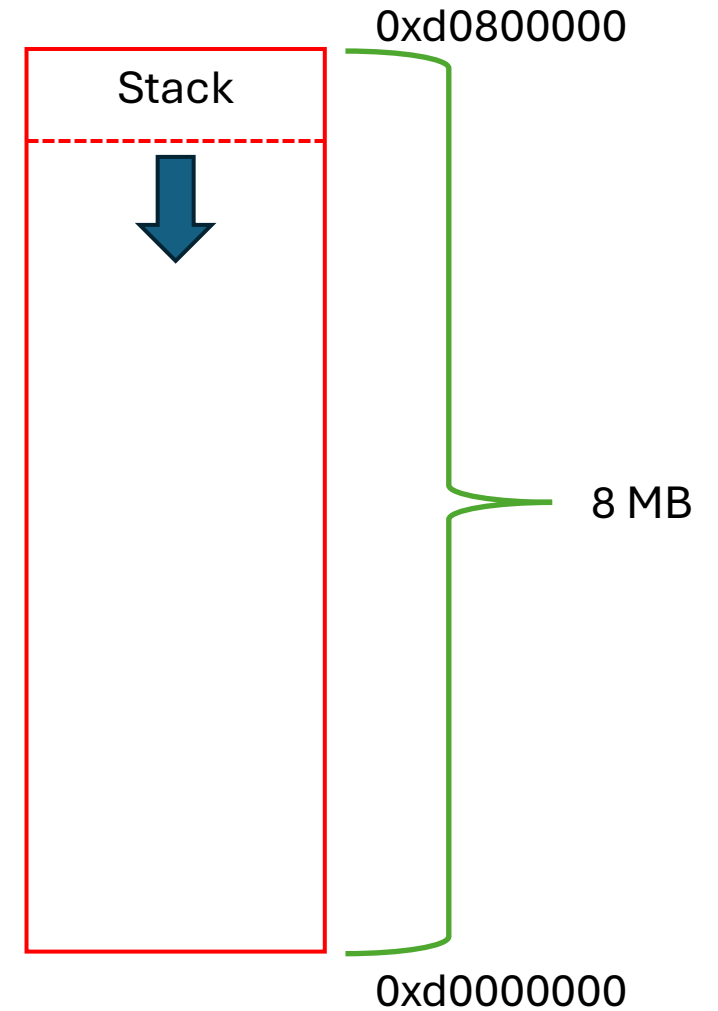
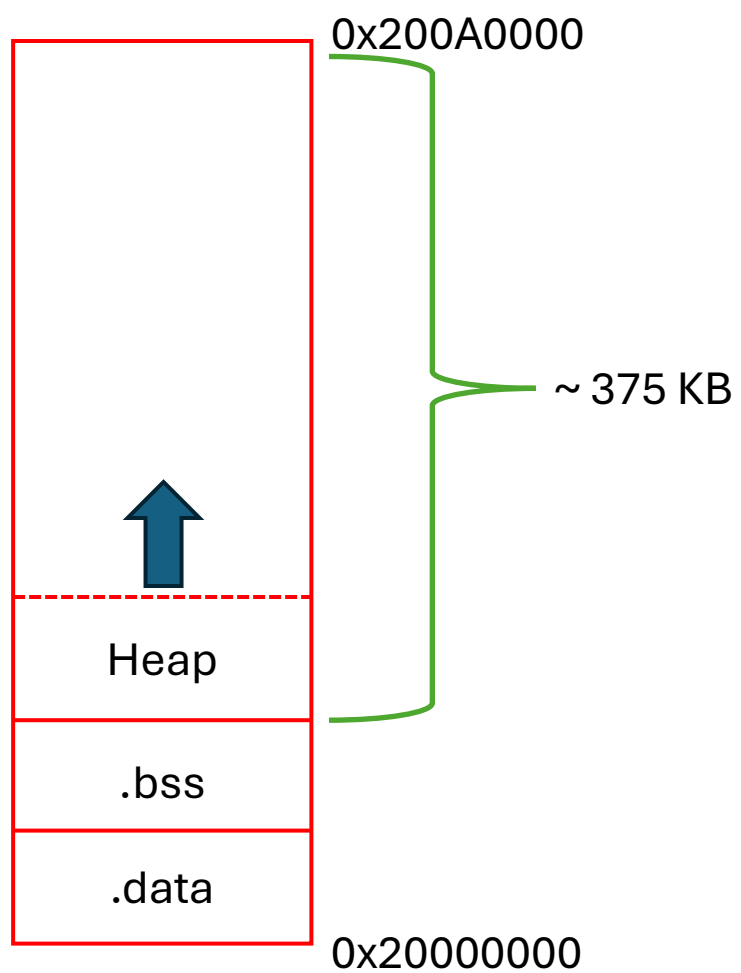


# Move the stack

```
BSP_SDRAM_Init();
MX_TIM1_Init();
MX_USART1_UART_Init();
MX_RNG_Init();
MX_I2C3_Init();
MX_SPI5_Init();
/* USER CODE BEGIN 2 */
SDRAM_Clear();
__asm volatile (
    "msr msp, %0\n"
    :
    : "r"(&__stack_end__)
    :
);
/* USER CODE END 2 */
```

Move manually SP to  
the end of the SDRAM

# State of RAM



# First victory

=> Key generation works

# Difficulties

- Begin to understand the meaning of bare metal environment in term of hardware
- Lack of documentation over internet to move the stack

# Phase 4 : The bug

- Code randomly crash at any routine :
  - Key generation
  - Signature
  - Verification
- Never see a bug like that before
- 3 Hypothesis :
  - Stack overflow
  - Heap overflow
  - Buffer overflow with file FP or use of GMP

# Phase 4 : The bug

- 3 Hypothesis :

- Stack overflow (?) 

STM32   
CubeProgrammer

- Heap overflow (?) 

STM32   
CubeMonitor

- Buffer overflow with file FP or use of GMP (?) 

Reading code + ChatGPT



# Real problem : The BSP

- Adapted be normally used as frame buffer for LCD
  - Must remove LTDC + DMA2D ( hardware for LCD )
  - Clean BSP
  - Increase the clock system => the BSD adapted for a clock system of 180 MHz

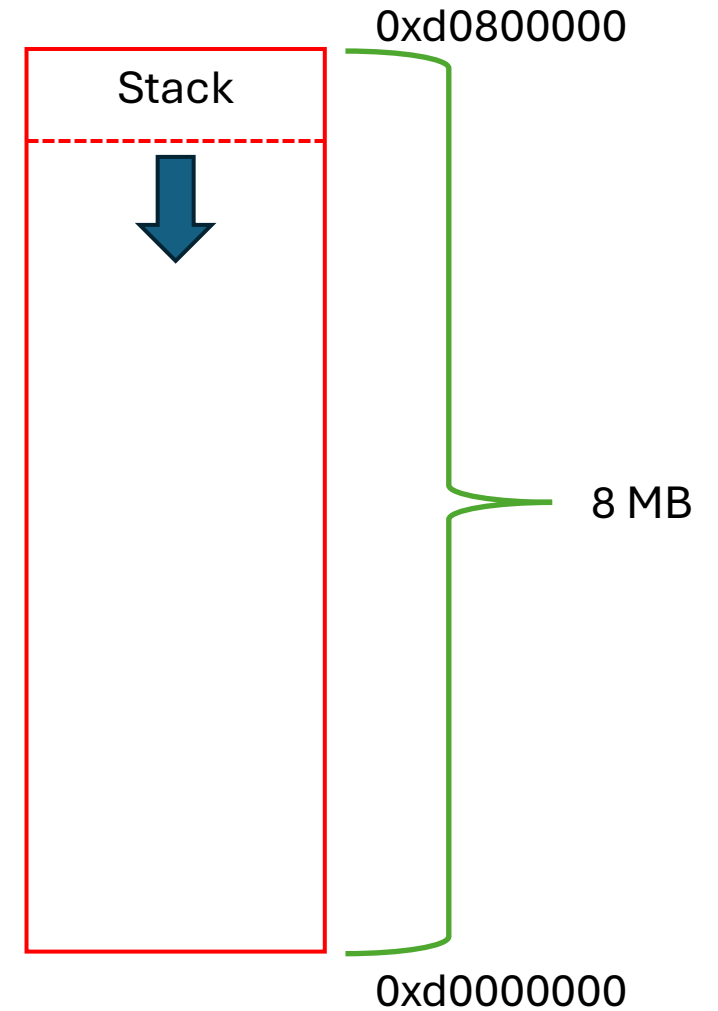
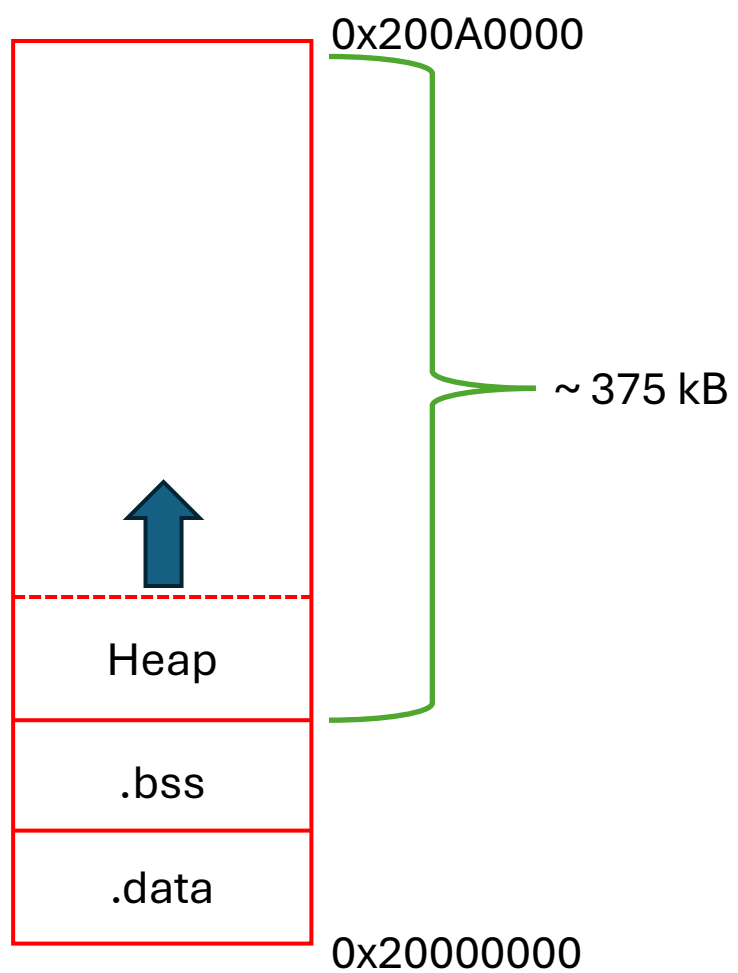
# Difficulties

- Take a very long time to solve ( time to run the code a least once ~ 33 minutes )
- Lot of new notion that I never see during my studies (hardware/electronics)
- New tools ( STM32 programmer and monitor)

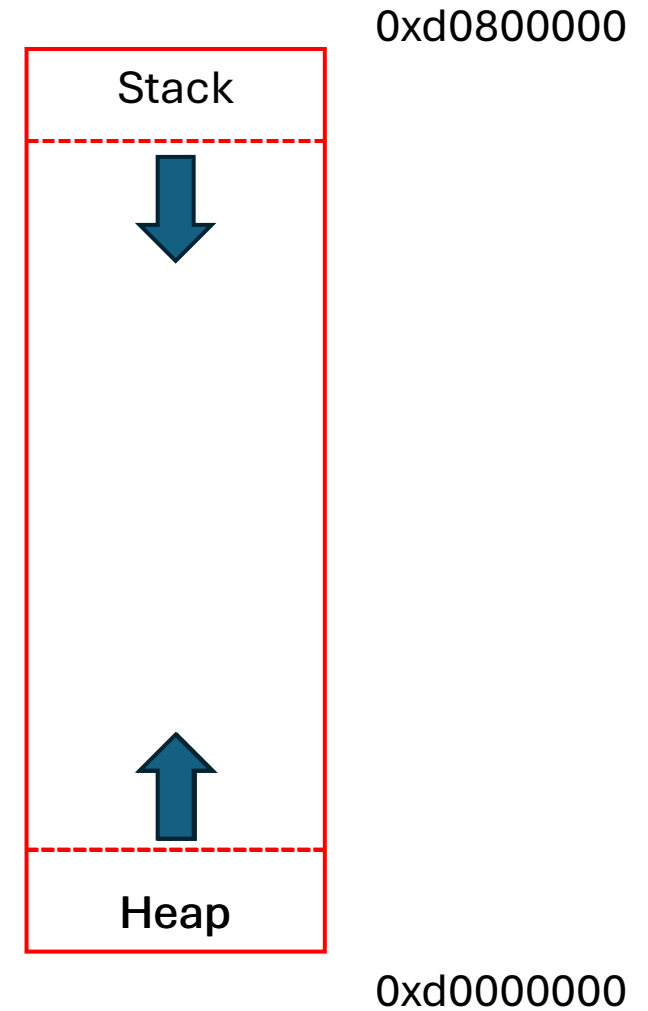
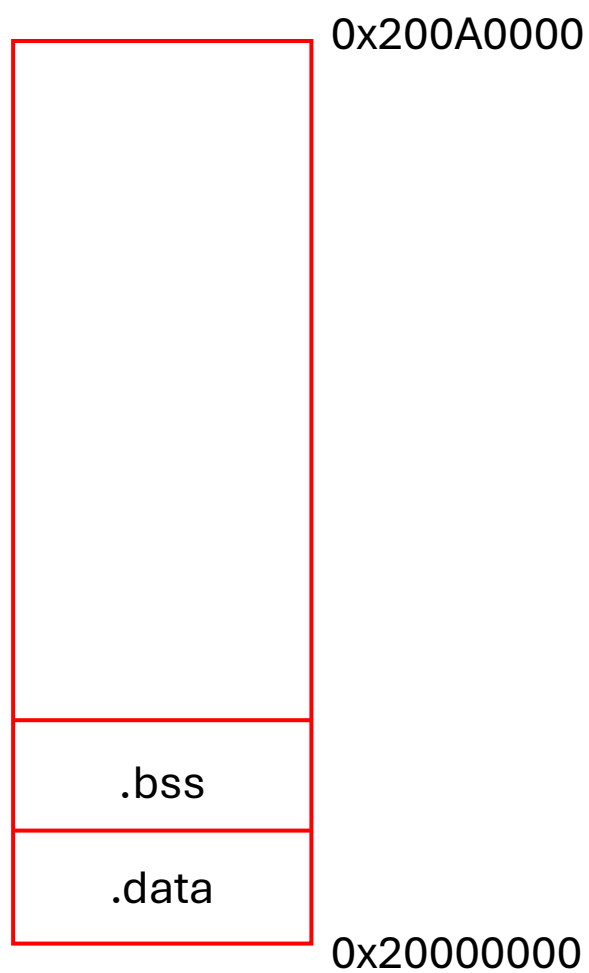
## Phase 5 : Evaluate SQLsign with all level of difficulties

- Level 1 
- Level 3 
- Level 5 




# State of RAM



# State of RAM



## Phase 5 : Evaluate SQLsign with all level of difficulties

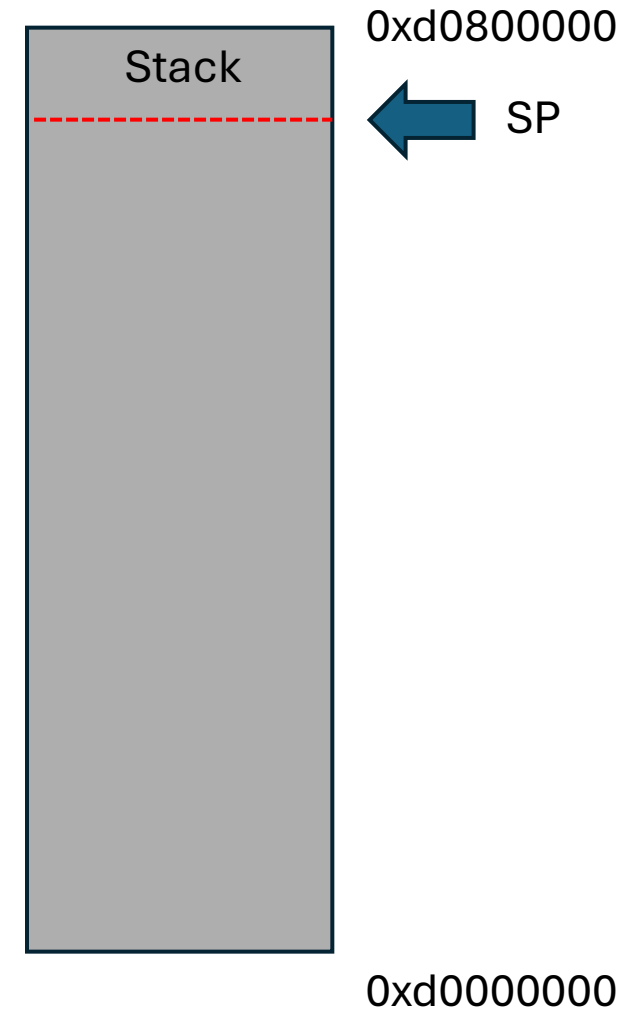
- Level 1 
- Level 3 
- Level 5 

# Phase 6 : Data gathering

- First time after cycle
- Heap only grow, never shrink
- Stack => technique of « stack painting »

# Stack painting

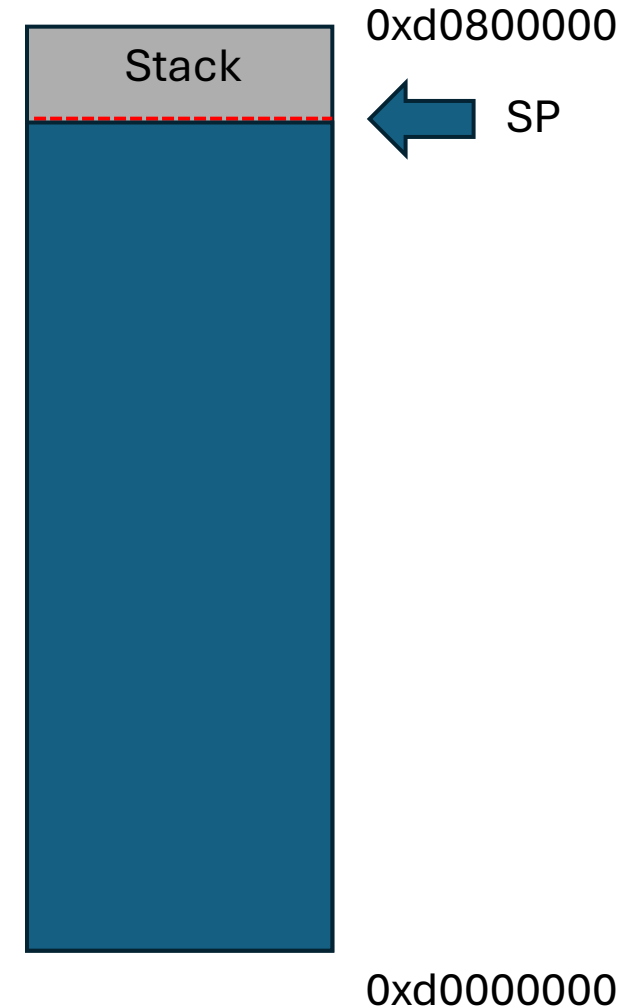
- 1) Fill the stack from lower address to Stack Point with a specific value ( example : 0xAB )





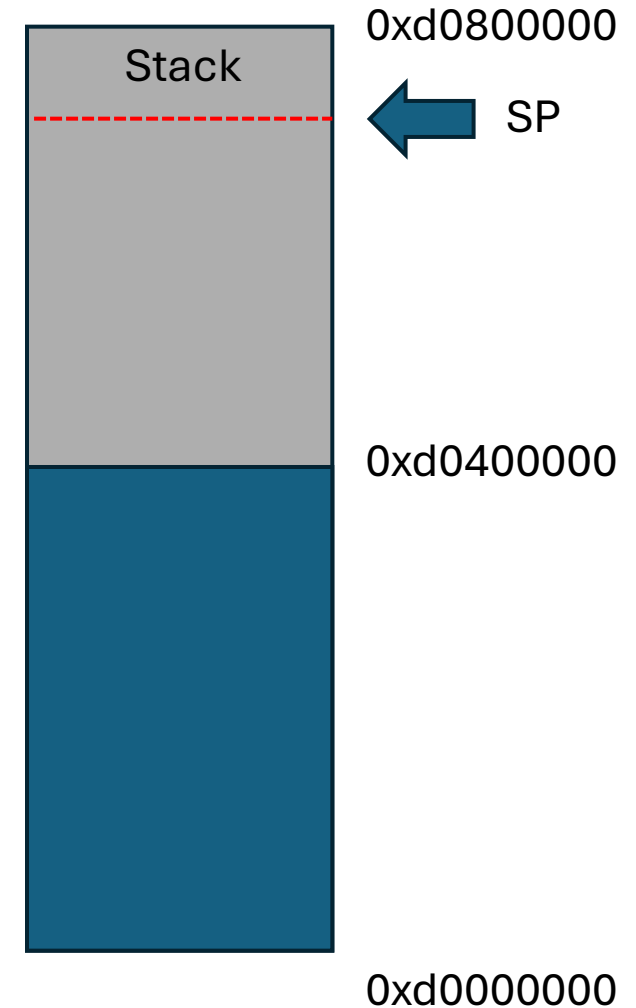
# Stack painting (simplified)

- 1) Fill the stack from lower address to Stack Point with a specific value ( example : 0xAB )
- 2) Let the code run



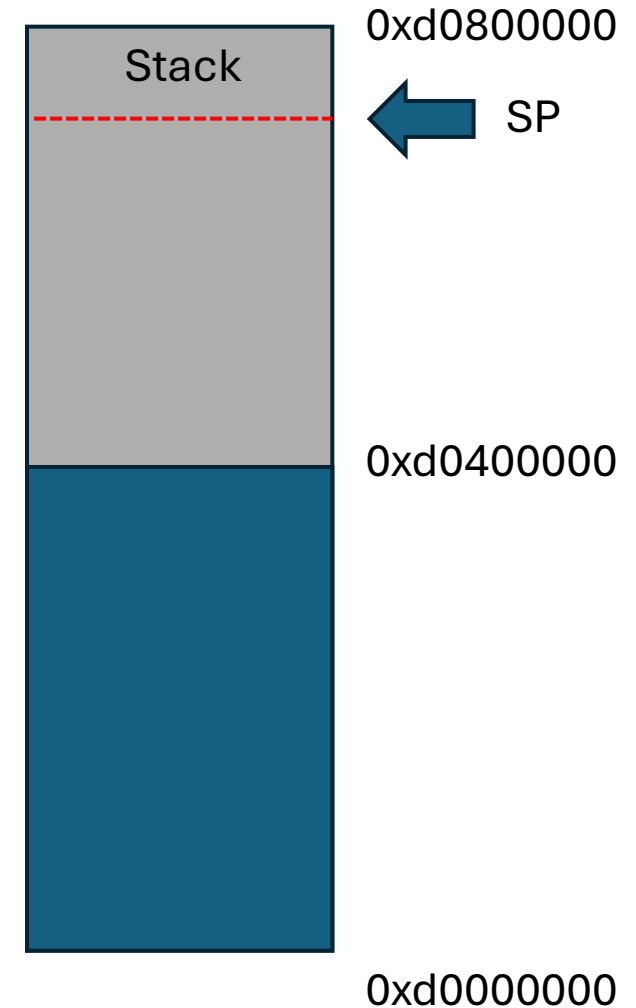
# Stack painting (simplified)

- 1) Fill the stack from lower address to Stack Point with a specific value ( example : 0xAB )
- 2) Let the code run



# Stack painting (simplified)

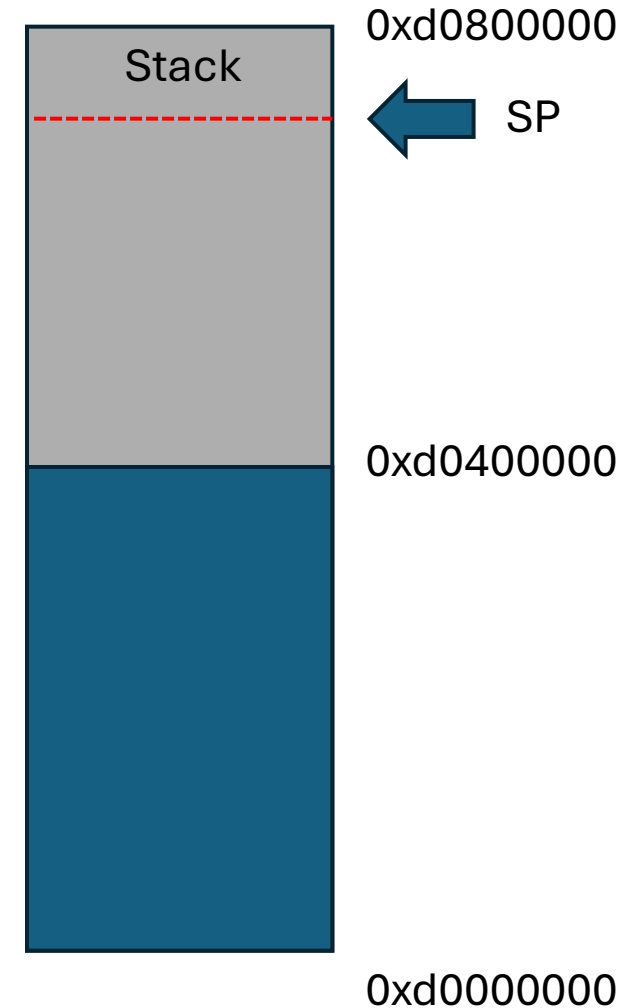
- 1) Fill the stack from lower address to Stack Point with a specific value ( example : 0xAB )
- 2) Let the code run
- 3) From lower address begin to observe a which address we do not find anymore our specific value



# Stack painting (simplified)

- 1) Fill the stack from lower address to Stack Point with a specific value ( example : 0xAB )
- 2) Let the code run
- 3) From lower address begin to observe a which address we do not find anymore our specific value
- 4) “Begin of the stack” - “Address found” = stack usage

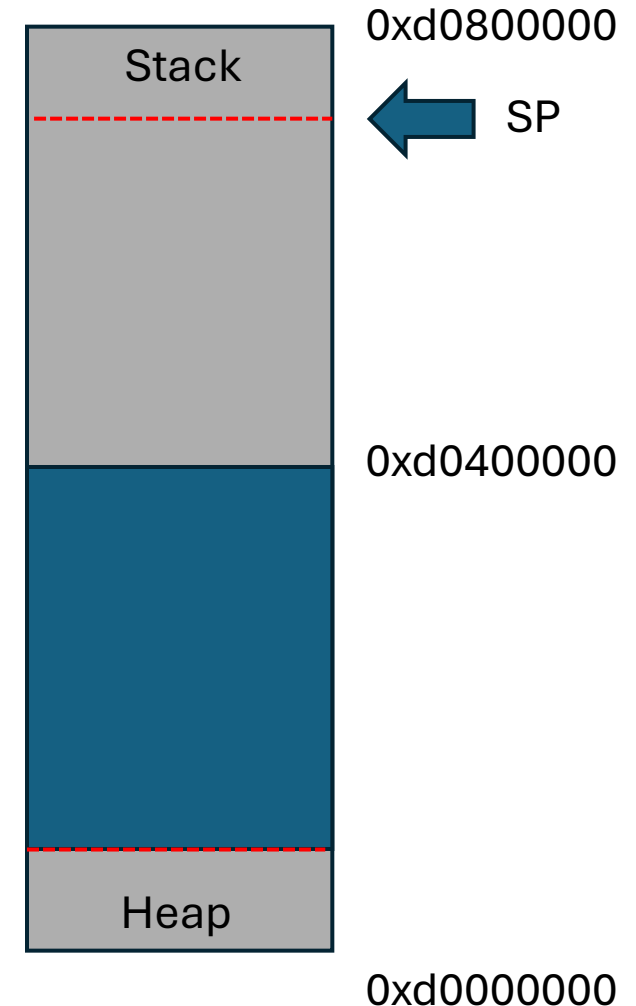
$$0xd0800000 - 0xd0400000 = 4 \text{ MB}$$



# Stack painting (With heap)

- 1) Fill the stack from **address of higher point of heap** to Stack Point with a specific value ( example : 0xAB )
- 2) Let the code run
- 3) From **address of higher point of heap** begin to observe a which address we do not find anymore our specific value
- 4) “Begin of the stack” - “Address found” = stack usage

$$0xd0800000 - 0xd0400000 = 4 \text{ MB}$$



# Data Analysis

- Size code :
  - Level 1 : 251.02 KB | .bss : 1.62 KB | .data : 7.27 KB
  - Level 3 : 302.76 KB | .bss : 1.71 KB | .data : 12.3 KB
  - Level 5 : 363.30 KB | .bss : 1.61 KB | .data : 14.23 KB

# Level 1

	Key Generation			Sign			Verification				
	Time	Cycle	Stack size	Time	Cycle	Stack size	Time	Cycle	Stack size		Heap usage
	m	MC	KB	m	MC	KB	m	MC	B		KB
Average	11.8767048	2511.77192	591.055158	21.0894416	2022.48404	595.661096	0.852116	612.951765	37324		91.48578
STD	5.08582841	1380.35171	0.88579772	3.63281444	1319.18558	0.84102802	0.01648659	178.072344	0		1.35058852
MAX	31.04008	4239.47333	592.33984	27.6488	4121.19041	596.56641	0.87365	845.48542	37324		92.47656
MIN	7.45138	15.74904	590.41797	15.07465	2.04906	594.66797	0.82792	351.63237	37324		86.48047

# Level 3

	Key Generation			Sign			Verification				
	Time	Cycle	Stack size	Time	Cycle	Stack size	Time	Cycle	Stack size		Heap usage
	m	MC	MB	m	MC	MB	m	MC	KB		KB
Average	36.07258	2768.35932	2.47372188	69.6412738	2117.096	2.48011188	2.85425375	761.150262	60.45703		421.982909
STD	11.5108994	924.153136	0.00103319	13.7229398	1443.0889	0.00076418	0.05187097	560.234755	2.2015E-14		20.6294427
MAX	64.19558	4115.71065	2.47533	96.6169	4080.44468	2.48066	2.92553	1531.00593	60.45703		437.46875
MIN	24.5804	1375.67845	2.47295	48.64852	23.12968	2.47862	2.7955	126.56757	60.45703		384.20703



# Level 5

	Key Generation			Sign			Verification				
	Time	Cycle	Stack size	Time	Cycle	Stack size	Time	Cycle	Stack size		Heap usage
	m	MC	MB	m	MC	MB	m	MC	KB		KB
Average	78.8275517	2365.68384	2.176985	154.731542	2505.88312	2.18348167	6.07347667	1884.90245	79.71484		402.822915
STD	13.8266964	1293.73328	0.00113885	27.7205353	1272.88002	0.00110465	0.12399968	1534.92315	0		7.30196233
MAX	97.77285	3679.82585	2.17791	191.66892	4145.14787	2.1845	6.20428	4001.89791	79.71484		405.86328
MIN	56.96618	716.90664	2.17553	112.77808	676.30501	2.18246	5.9381	67.51296	79.71484		387.91797

# Data criticism

- Technologie : SDRAM slower than SRAM ( x10 to x100)
  - Increase time
  - Increase number cycle
  - ( does not change the stack and heap)
  - Very bad optimization
- Compare to the work of team SQIsign :
  - data small change in term of stack usage
  - Not the same code for verification

Performance data for verification only using a pure C implementation, running on an ARM Cortex-M4 CPU:

parameter set	clock cycles	stack usage	code size
NIST - I	123 megacycles	31 kilobytes	40 kilobytes
NIST - III	375 megacycles	50 kilobytes	44 kilobytes
NIST - V	751 megacycles	64 kilobytes	46 kilobytes

# Future work

- Compare/evaluate with and without FPU
- Adapt the work to PQM4
- Improve SQIsign to use less RAM resource
- Improve hardware configuration
- Evaluate if the code work between M4 and 64 bits architecture
- Update verification with the recent work of SQIsign team

**END**

# Printf

- 2 Solutions
  - use Serial debugger ( output on cube ide)
  - Use UART ( output on external program Tera Term 5)
- Serial debugger => initial solution
- UART => final solution to gather datas

**STM32**   
**CubeIDE**

---

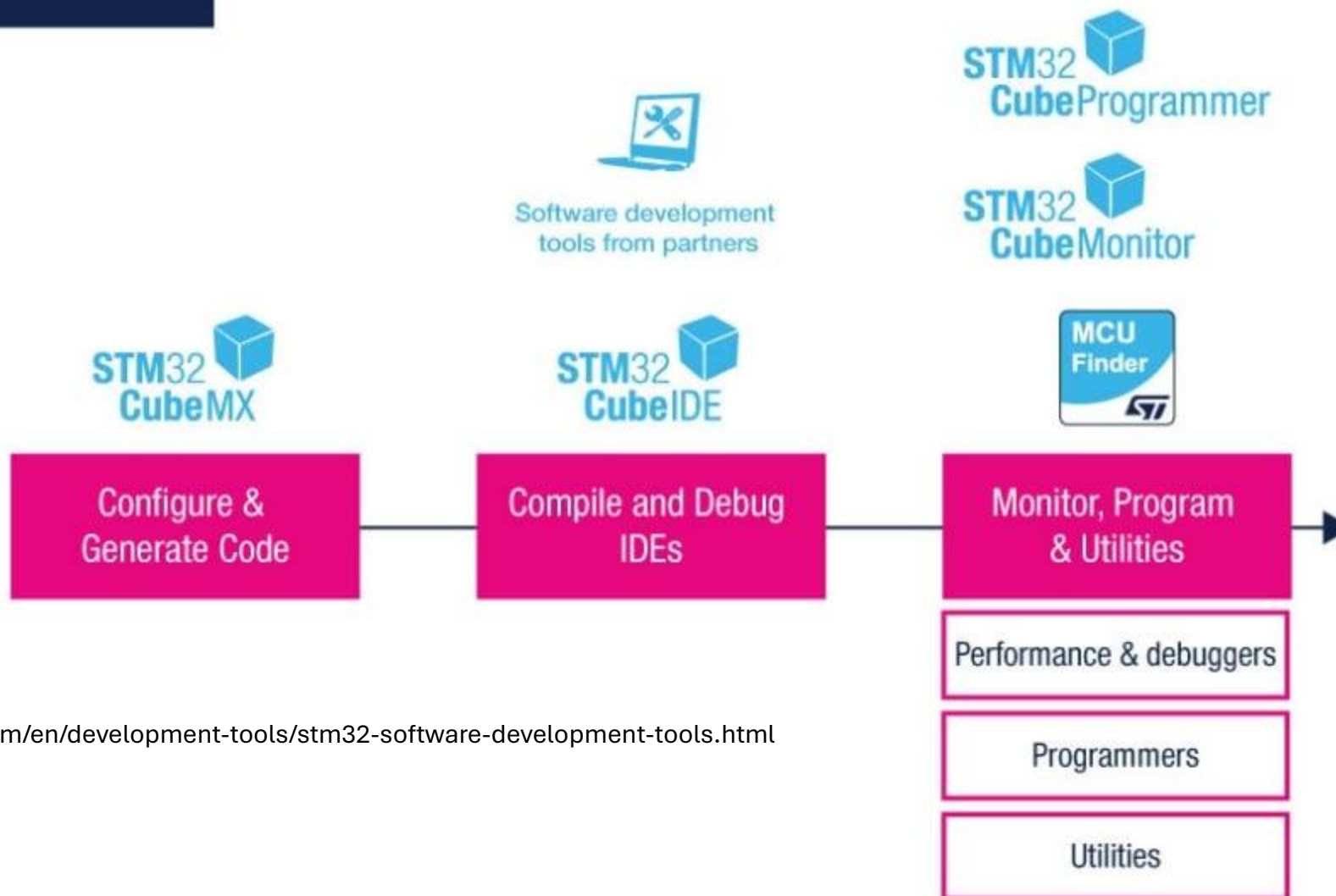
**STM32**   
**CubeMonitor**

**STM32**   
**CubeProgrammer**

---

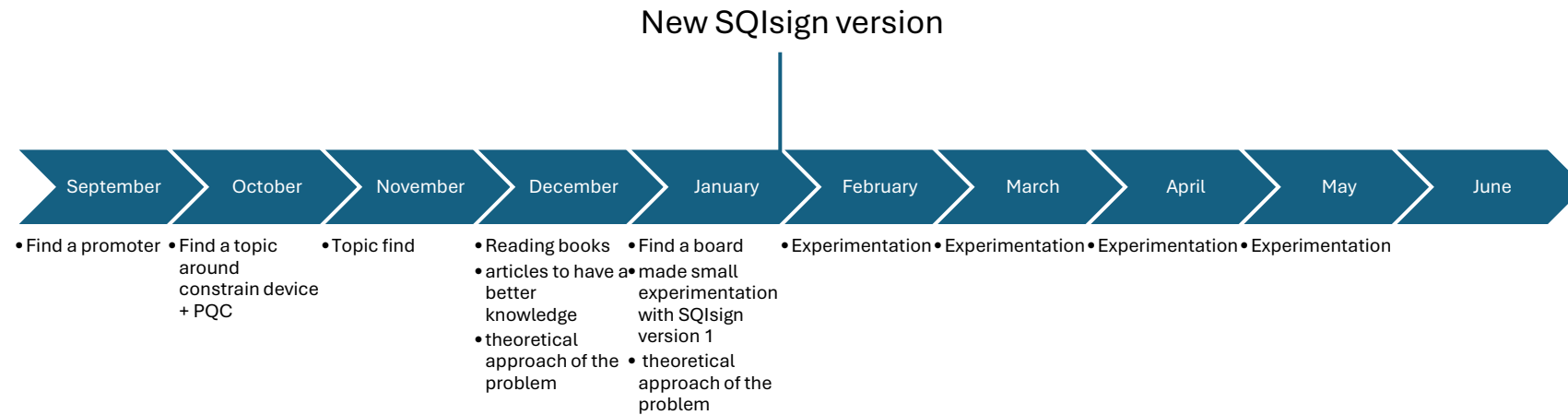
**STM32**   
**CubeMX**

## Software development tools



<https://www.st.com/en/development-tools/stm32-software-development-tools.html>

# Timeline (personal work)





# Timeline (NIST)



# Go to 32bits to 64bits

- Modifie the Fp file
- Modifie inside macroM4.h, radix 32 to radix 64

# Evaluate the good work between M4 and 64 bits architecture

**ARM CORTEX M4 datasheet**

**GMP**

Floating-Point Unit

Optional single precision floating-point unit  
IEEE 754 compliant

<https://www.st.com/en/evaluation-tools/nucleo-l4r5zi.html>

## Floating Point Mode

On some systems, the hardware floating point has a control mode which can set all operations to be done in a particular precision, for instance single, double or extended on x86 systems (x87 floating point). The GMP functions involving a **double** cannot be expected to operate to their full precision when the hardware is in single precision mode. Of course this affects all code, including application code, not just GMP.

<https://gmplib.org/gmp-man-6.3.0.pdf>

# Evaluate the good work between M4 and 64 bits architecture

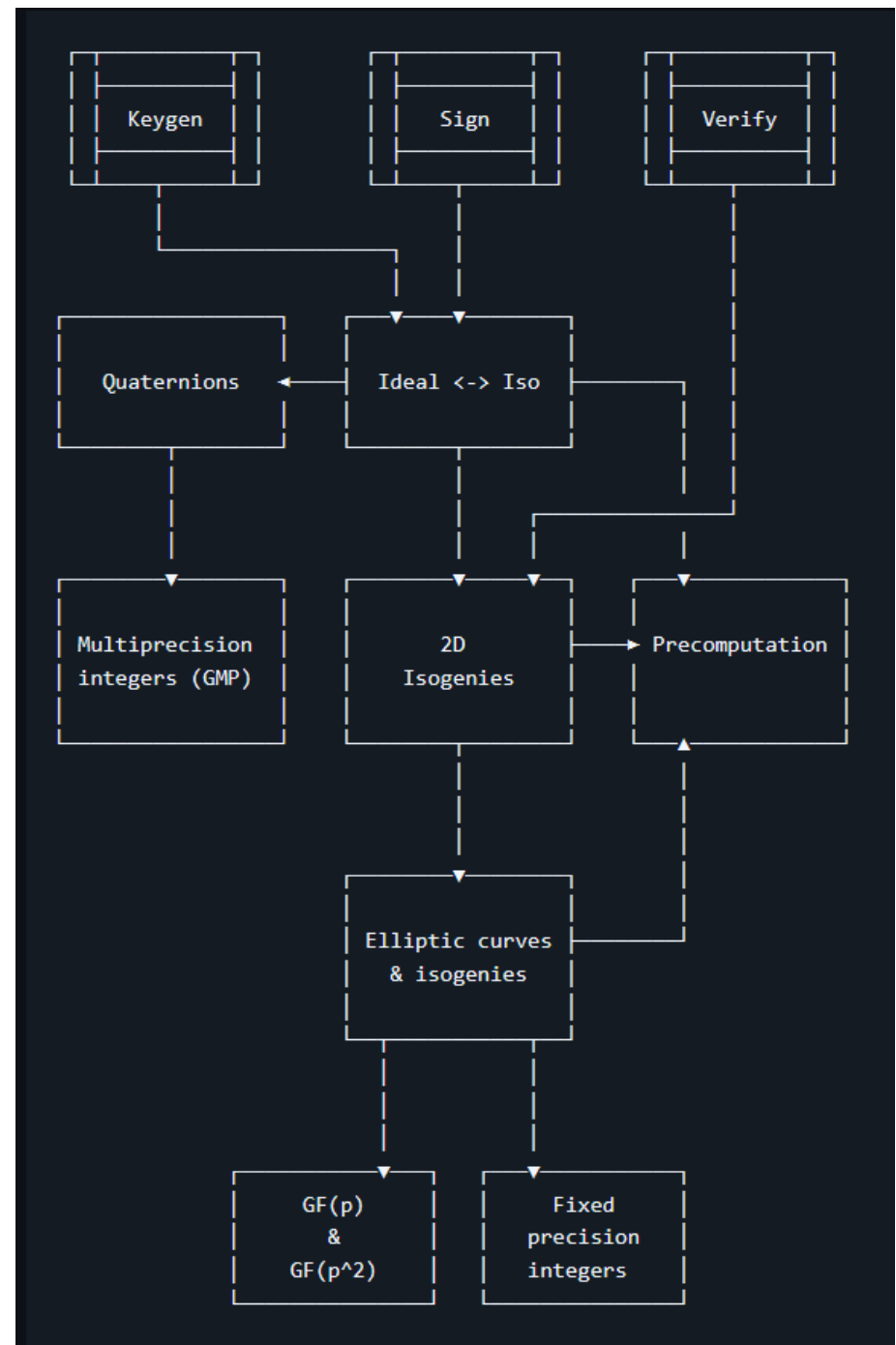
## Mini-gmp.h

```
int mpz_cmp_d (const mpz_t, double);  
int mpz_cmpabs_d (const mpz_t, double);  
double mpz_get_d (const mpz_t);  
void mpz_set_d (mpz_t, double);  
void mpz_init_set_d (mpz_t, double);
```

## Mini-gmp-extra.h

```
double mpz_get_d_2exp(signed long int *exp,  
const mpz_t op);  
  
double mini_mpz_get_d_2exp(signed long int  
*exp, const mpz_t op);
```

# SQIsign Dependencies





# STM32 MCUs

## 32-bit Arm® Cortex®-M



<div>★</div> <div>High Performance</div>			<div>STM32F7</div> <div>1082 CoreMark 216 MHz Cortex-M7</div>	<div>STM32H7</div> <div>Up to 3224 CoreMark Up to 600 MHz Cortex-M7 240 MHz Cortex-M4</div>	<div>STM32N6</div> <div>3360 CoreMark 800 MHz Cortex-M55</div>
	<div>STM32F2</div> <div>398 CoreMark 120 MHz Cortex-M3</div>	<div>STM32F4</div> <div>608 CoreMark 180 MHz Cortex-M4</div>	<div>STM32H5</div> <div>Up to 1023 CoreMark 250 MHz Cortex-M33</div>		
<div>➤➤</div> <div>Mainstream</div>	<div>STM32C0</div> <div>114 CoreMark 48 MHz Cortex-M0+</div>	<div>STM32G0</div> <div>142 CoreMark 64 MHz Cortex-M0+</div>	<div>STM32G4</div> <div>569 CoreMark 170 MHz Cortex-M4</div>		<div>●</div> Optimized for mixed-signal applications
			<div>STM32F0</div> <div>106 CoreMark 48 MHz Cortex-M0</div>	<div>STM32F1</div> <div>177 CoreMark 72 MHz Cortex-M3</div>	
<div>🔋</div> <div>Ultra-low-power</div>			<div>STM32L4+</div> <div>409 CoreMark 120 MHz Cortex-M4</div>	<div>STM32U5</div> <div>651 CoreMark 160 MHz Cortex-M33</div>	<div>STM32U3</div> <div>393 CoreMark 96 MHz Cortex-M33</div>
	<div>STM32L0</div> <div>75 CoreMark 32 MHz Cortex-M0+</div>	<div>STM32U0</div> <div>140 CoreMark 56 MHz Cortex-M0+</div>	<div>STM32L4</div> <div>273 CoreMark 80 MHz Cortex-M4</div>	<div>STM32L5</div> <div>443 CoreMark 110 MHz Cortex-M33</div>	
<div>📶</div> <div>Wireless</div>			<div>STM32WL</div> <div>162 CoreMark 48 MHz Cortex-M4 48 MHz Cortex-M0+</div>	<div>STM32WB0</div> <div>156 CoreMark 64 MHz Cortex-M0+</div>	<div>STM32WB</div> <div>219 CoreMark 64 MHz Cortex-M4 32 MHz Cortex-M0+</div>
					<div>STM32WBA</div> <div>407 CoreMark 100 MHz Cortex-M33</div>
<div>●</div> Cortex-M0+ Radio co-processor					



# STM32 MCUs 32-bit Arm® Cortex®-M



<div>★</div> <div>High Performance</div>	STM32F2 398 CoreMark 120 MHz Cortex-M3	STM32F4 608 CoreMark 180 MHz Cortex-M4	STM32H5 Up to 1023 CoreMark 250 MHz Cortex-M33	STM32F7 1082 CoreMark 216 MHz Cortex-M7	STM32H7 Up to 3224 CoreMark Up to 600 MHz Cortex-M7 240 MHz Cortex-M4	STM32N6 3360 CoreMark 800 MHz Cortex-M55
<div>➤➤</div> <div>Mainstream</div>	STM32C0 114 CoreMark 48 MHz Cortex-M0+	STM32G0 142 CoreMark 64 MHz Cortex-M0+	STM32F0 106 CoreMark 48 MHz Cortex-M0	STM32F1 177 CoreMark 72 MHz Cortex-M3	STM32G4 569 CoreMark 170 MHz Cortex-M4	<div>●</div> Optimized for mixed-signal applications
<div>🔋</div> <div>Ultra-low-power</div>	STM32L0 75 CoreMark 32 MHz Cortex-M0+	STM32U0 140 CoreMark 56 MHz Cortex-M0+	STM32L4 273 CoreMark 80 MHz Cortex-M4	STM32L4+ 409 CoreMark 120 MHz Cortex-M4	STM32U5 651 CoreMark 160 MHz Cortex-M33	STM32U3 393 CoreMark 96 MHz Cortex-M33
<div>📶</div> <div>Wireless</div>						
	STM32WL 162 CoreMark 48 MHz Cortex-M4 48 MHz Cortex-M0+		STM32WB0 156 CoreMark 64 MHz Cortex-M0+		STM32WB 219 CoreMark 64 MHz Cortex-M4 32 MHz Cortex-M0+	STM32WBA 407 CoreMark 100 MHz Cortex-M33
<div>●</div> Cortex-M0+ Radio co-processor						

[https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html?ecmp=tt21056\\_gl\\_link\\_may2021](https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html?ecmp=tt21056_gl_link_may2021)