Réunion flash

Point hebdomadaire

Duzés Florian

Sommaire

01
État des lieux

Point actuel

Fait → **En cours** → **Prévus**

Fait

- Activer chaîne de compilation
- Compilation en x86_64
- *Introduction* du mémoire

En cours

- Remplir les config
- Régler les problèmes d'analyse de Binsec
- Chaîne de bout en bout x86_64

Prévus

- Couverture des architectures différentes
- Couverture des compilateurs

Réalisation

Fait → **En cours** → **Prévus**

Fait

- Remplir les config
- Appel des fonctions voisines

En cours

- Affiner la génération des tests pour *Krmllib.h* et *Hacl_Hash_Blake2b.h*
- Chaîne de bout en bout x86_64
- *Introduction* du mémoire + section 1

Prévus

- Couverture des architectures différentes
- Couverture des compilateurs

02
Génération des tests

Point de vue général

Problèmes encore en cours la semaine dernière

- Remplissage des configs à 1/3
- Gestion des exceptions
- Appel aux fonctions voisines

02
Génération des tests

01 - *Fichiers de configuration*

Retour sur les fichiers de configuration

```
,"Hacl_Hash_Blake2b_malloc_with_params_and_key": {
    "*p":"Hacl_Hash_Blake2b_blake2_params"
  ,"last_node":"false"
  ,"*k":"BUF"
  ,"BUF":1
}

,"Hacl_Hash_Blake2b_malloc_with_key": {
    "*k":"BUF_KEY"
  ,"kk":"KEY"
  ,"BUF_KEY":1
  ,"KEY":1
}
```

Code – Hacl_Hash_Blake2b.h

```
 ,"Hacl_HPKE_Curve51_CP128_SHA512_setupBaseR": {
    "o_ctx":"Hacl_HPKE_Interface_Hacl_Impl_HPKE_Hacl_Meta_HPKE
            "
  ,"*enc":"BUFFER"
  ,"*skR":"BUFFER"
  ,"infolen":"BUFFER"
  ,"*info":"BUFFER"
  ,"BUFFER":1
}
 ,"Hacl_HPKE_Curve51_CP128_SHA512_sealBase": {
    "*skE":"BUFFER"
  ,"*pkR":"BUFFER"
  ,"infolen":"BUFFER"
  ,"*info":"BUFFER"
  ,"aadlen":"BUFFER"
  ,"*aad":"BUFFER"
  ,"plainlen":"BUFFER"
  ,"*plain":"BUFFER"
  ,"*o_enc":"BUFFER"
  ,"*o_ct":"BUFFER"
  ,"BUFFER":1
}
```

Code – Hacl_HPKE_Curve51_CP128_SHA512.h

02
Génération des tests

02 - *Construction d'un test de A à Z*

Production d'un test - 1

```c
//
// Made by
// ANDHRÍMNIR - 0.2.2
// 17-06-2025
//

#include <stdlib.h>
#include "Hacl_AEAD_Chacha20Poly1305.h"

#define tag TAG_SIZE
#define output BUF_SIZE
#define data AAD_SIZE
#define nonce NONCE_SIZE
#define key KEY_SIZE
#define input BUF_SIZE

#define BUF_SIZE 16384
#define AAD_SIZE 12
#define TAG_SIZE 16
#define NONCE_SIZE 12
#define KEY_SIZE 32

uint8_t output[BUF_SIZE];    uint8_t tag[TAG_SIZE];
uint8_t input[BUF_SIZE];     uint8_t data[AAD_SIZE];
uint8_t key[KEY_SIZE];       uint8_t nonce[NONCE_SIZE];

int main (int argc, char *argv[]){
Hacl_AEAD_Chacha20Poly1305_encrypt(output, tag, input, BUF_SIZE, data, AAD_SIZE, key, nonce);
  exit(0);
}
```

Code – Hacl_AEAD_Chacha20Poly1305_encrypt.c

Production d'un test - 2

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_EC_K256.h"

#define BUFFER_SIZE 5
uint64_t a[BUFFER_SIZE];
uint64_t out[BUFFER_SIZE];


int main (int argc, char *argv[]){
Hacl_EC_K256_felem_sqr(a, out);
  exit(0);
}
```

Code – Hacl_EC_K256_felem_sqr.c

Forme générique d'un test

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_EC_K256.h"

#define BUFFER_SIZE 5
uint64_t a[BUFFER_SIZE];
uint64_t out[BUFFER_SIZE];


int main (int argc, char *argv[]){
Hacl_EC_K256_felem_sqr(a, out);
  exit(0);
}
```

Code – Hacl_EC_K256_felem_sqr.c

Forme générique d'un test

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_EC_K256.h"

#define BUFFER_SIZE 5
uint64_t a[BUFFER_SIZE];
uint64_t out[BUFFER_SIZE];

int main (int argc, char *argv[]){
Hacl_EC_K256_felem_sqr(a, out);
  exit(0);
}
```

Code – Hacl_EC_K256_felem_sqr.c

Forme générique d'un test

```
//
// Made by
// ANDHRÍMNIR - 0.5.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_EC_K256.h"

#define BUFFER_SIZE 5
uint64_t a[BUFFER_SIZE];
uint64_t out[BUFFER_SIZE];

int main (int argc, char *argv[]){
Hacl_EC_K256_felem_sqr(a, out);
  exit(0);
}
```

Phase de déclaration : 8 lignes

Phase variables

Phase principales : 4 lignes

Code – Hacl_EC_K256_felem_sqr.c

02
Génération des tests

03 - *Gestion des appels de fonctions par collage*

Appel aux fonctions voisines

```
,"Hacl_MAC_Poly1305_Simd256_reset": {
   "*state":"Hacl_MAC_Poly1305_Simd256_malloc"
  ,"*key":"BUF_KEY"
  ,"BUF_KEY":1
}
```

Code – Hacl_MAC_Poly1305_Simd256.json

Schéma de construction d'un test

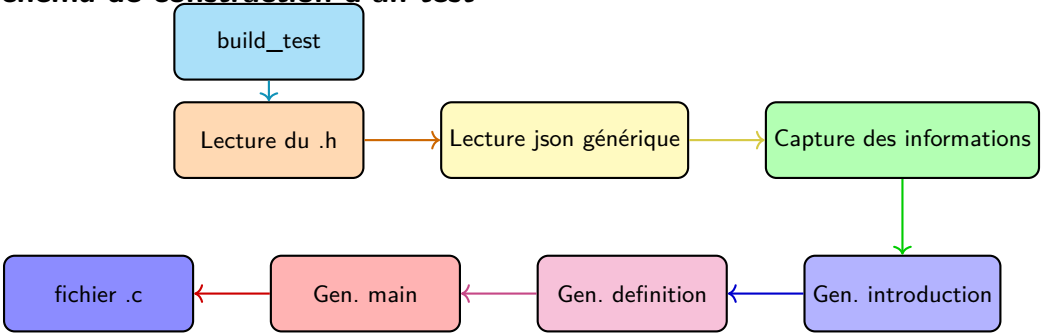


Schéma de construction d'un test

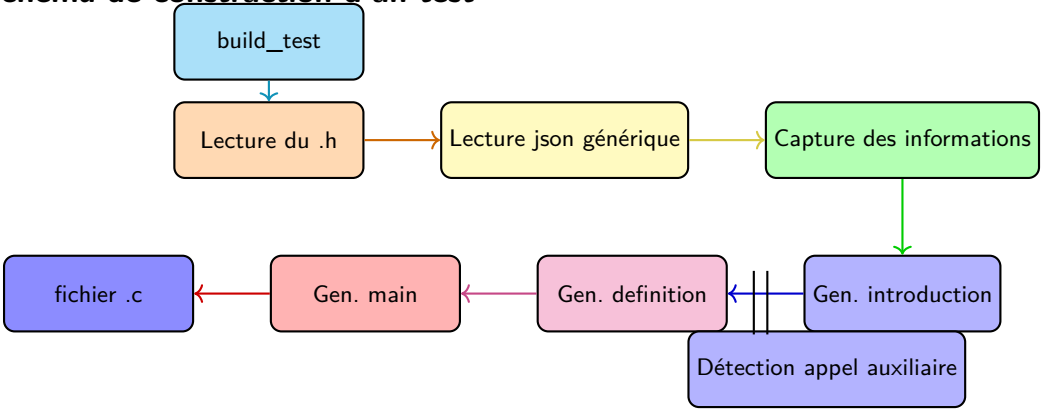
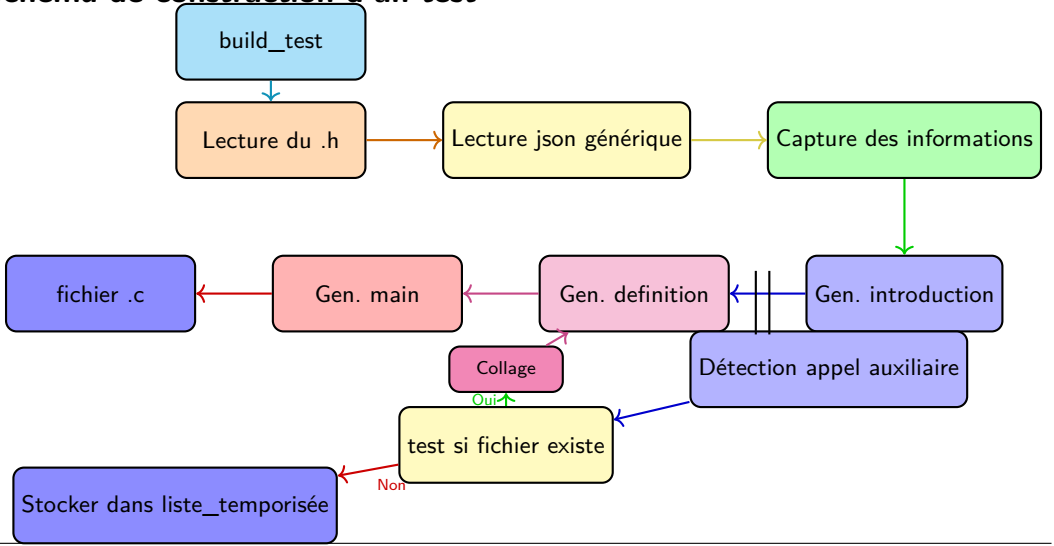


Schéma de construction d'un test



Résultat final

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_MAC_Poly1305_Simd256.h"
```

Code – Hacl_MAC_Poly1305_Simd256_reset.c

Résultat final

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_MAC_Poly1305_Simd256.h"

#define BUF_KEY 1
uint8_t key[BUF_KEY];
```

Code – Hacl_MAC_Poly1305_Simd256_reset.c

Résultat final

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_MAC_Poly1305_Simd256.h"

#define BUF_KEY 1
uint8_t key[BUF_KEY];

#define BUFFER 1
uint8_t key[BUFFER];
Hacl_MAC_Poly1305_Simd256_state_t state = Hacl_MAC_Poly1305_Simd256_malloc(key
    );
```

Code – Hacl_MAC_Poly1305_Simd256_reset.c

Résultat final

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_MAC_Poly1305_Simd256.h"

#define BUF_KEY 1
uint8_t key[BUF_KEY];

#define BUFFER 1
uint8_t key[BUFFER];
Hacl_MAC_Poly1305_Simd256_state_t state = Hacl_MAC_Poly1305_Simd256_malloc(key
    );
```

Code – Hacl_MAC_Poly1305_Simd256_reset.c

Résultat final

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_MAC_Poly1305_Simd256.h"

#define BUF_KEY 1
uint8_t key[BUF_KEY];

#define BUFFER 1
uint8_t key[BUFFER];
Hacl_MAC_Poly1305_Simd256_state_t state = Hacl_MAC_Poly1305_Simd256_malloc(key
    );

int main (int argc, char *argv[]){
Hacl_MAC_Poly1305_Simd256_reset(state, key);
  exit(0);
}
```

Résultat final

```c
//
// Made by
// ANDHRÍMNIR - 0.3.0
// 09-07-2025
//

#include <stdlib.h>
#include "Hacl_MAC_Poly1305_Simd256.h"

#define BUF_KEY 1
uint8_t key[BUF_KEY];

#define BUFFER 1
uint8_t key[BUFFER];
Hacl_MAC_Poly1305_Simd256_state_t state = Hacl_MAC_Poly1305_Simd256_malloc(key);

int main (int argc, char *argv[]){
Hacl_MAC_Poly1305_Simd256_reset(state, key);
  exit(0);
}
```

Code – Hacl_MAC_Poly1305_Simd256_reset.c

03
Binsec en x86_64

Génération des scripts

Problème d'analyse

- Temps de compilation ++
- Crash de binsec // crash machine
- Trop d'information ?

Démo

Test ensemble ?

04
Conclusion

Conclusion

Objectif

Finir le module x86_64.

- ☑ ~~Remplir les configurations~~
- ☑ ~~Générer les tests~~* [1]
- ☑ ~~Compiler les tests~~*
- ☑ Analyser les tests

1. * : encore quelques fonctions/tests qui résistent -> ficher de config pas idéal

Merci.