CUDA programing

Kyoung Bae Jang

https://youtu.be/nRmNMzJLjoU

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

CryptoCraft LAB

SIMECK

- A family of lightweight block ciphers (SPECK + SIMON)

Table 1. Parameters of SIMECK.

Cipher	Block size (bits)	Key size (bits)	Word size (bits)	Keywords m	Rounds r
SIMECK-32/64	32	64	16	4	32
SIMECK-48/96	48	96	24	4	36
SIMECK-64/128	64	128	32	4	44

- The round function and the key schedule are based on the Feistel architecture

SIMECK

- Round function
  - Similar with SIMON
  - $$R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i)$$
  - $$f(x) = x \,\&\, \mathrm{ROL}_5(x) \oplus \mathrm{ROL}_1(x).$$

Table 2. Notation.

Notations	Meaning
l_0	Plaintext to be encrypted
r_0	Plaintext to be encrypted
k_i	Round key
$\oplus$	XOR operation
&	AND operation
ROL_i	Rotation left operation (i-bit)

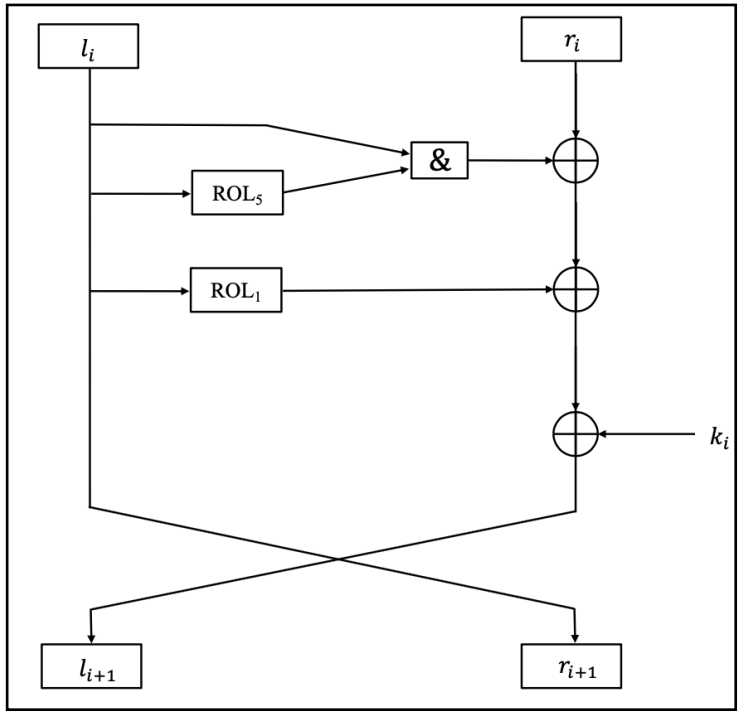


Fig. 1. Round function of SIMECK

SIMECK

- Key schedule
  - Similar with SPECK
  - $$k_{i+1} = t_i$$
    $$t_{i+3} = k_i \oplus f(t_i) \oplus C \oplus (z_j)_i$$
  - C (constant) and z_j (sequence) of SIMECK 32/64

```
uint16_t constant = 0xFFFC;
uint32_t sequence = 0x9A42BB1F;
```

  - Initial key

    $(t2, t1, t0, k0)$

Table 2. Notation.

Notations	Meaning
l_0	Plaintext to be encrypted
r_0	Plaintext to be encrypted
k_i	Round key
$\oplus$	XOR operation
&	AND operation
ROL_i	Rotation left operation (i-bit)

SIMECK - C

- SIMECK 32/64
  - Round function and Rotation Left

```
#define LROT16(x, r) (((x) << (r)) | ((x) >> (16 - (r))))

#define ROUND32(key, lft, rgt, tmp)  { \
    tmp = (lft); \
    lft = ((lft) & LROT16((lft), 5)) ^ LROT16((lft), 1) ^ (rgt) ^ (key); \
    rgt = (tmp); \
}
```

$$R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i)$$

$$f(x) = x \,\&\, \mathrm{ROL}_5(x) \oplus \mathrm{ROL}_1(x).$$

SIMECK - C

- SIMECK 32/64
  - Key schedule

```
constant &= 0xFFFC;
constant |= sequence & 1;
sequence >>= 1;
```

$$k_{i+1} = t_i$$
$$t_{i+3} = k_i \oplus f(t_i) \oplus C \oplus (z_j)_i$$

* Initial key : $(t2, t1, t0, k0)$

  - Similar with round function

$$R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i)$$

SIMECK - C

- SIMECK 32/64

```c
void simeck_32_64(uint16_t master_key[], uint16_t plaintext[],uint16_t ciphertext[]) {
    int NUM_ROUNDS = 32;

    uint16_t keys[4] = {
        master_key[0],
        master_key[1],
        master_key[2],
        master_key[3],
    };
    ciphertext[0] = plaintext[0];
    ciphertext[1] = plaintext[1];
    uint16_t temp;

    uint16_t constant = 0xFFFC;
    uint32_t sequence = 0x9A42BB1F;

    for (int i = 0; i < NUM_ROUNDS; i++) {

        ROUND32(keys[0],ciphertext[1],ciphertext[0],temp);

        constant &= 0xFFFC;
        constant |= sequence & 1;
        sequence >>= 1;

        ROUND32(constant, keys[1], keys[0], temp);

        // rotate the LFSR of keys
        temp = keys[1];
        keys[1] = keys[2];
        keys[2] = keys[3];
        keys[3] = temp;
    }
}
```

$$R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i)$$

$$k_{i+1} = t_i$$

$$t_{i+3} = k_i \oplus f(t_i) \oplus C \oplus (z_j)_i$$

SIMECK – CUDA C(1)

- SIMECK 32/64
  - Round function + key schedule

```
for (int i = 0; i < NUM_ROUNDS; i++) {

    ROUND32(keys[0],ciphertext[1],ciphertext[0],temp);

    constant &= 0xFFFC;
    constant |= sequence & 1;
    sequence >>= 1;

                            lft       rgt
    ROUND32(constant, keys[1], keys[0], temp);

    // rotate the LFSR of keys
    temp = keys[1];
    keys[1] = keys[2];
    keys[2] = keys[3];
    keys[3] = temp;
}
```

Value update

```
#define ROUND32(key, lft, rgt, tmp)  { \
    tmp = (lft); \
    lft = ((lft) & LROT16((lft), 5)) ^ LROT16((lft), 1) ^ (rgt) ^ (key); \
    rgt = (tmp); \
}
```

Key use in round fuction

Key update in key schedule

$$R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i)$$

$$k_{i+1} = t_i$$
$$t_{i+3} = k_i \oplus f(t_i) \oplus C \oplus (z_j)_i$$

- Frequent memory access GPU & CPU → Low performance

SIMECK – CUDA C(2)

- SIMECK 32/64
  - Encrypt multiple messages(1024 x 76) with multiple keys (1024 x 76)

```
int main() {
    int blobknum = 76;
    int number = 1024 * blobknum;
    uint16_t text32[1024 * 76][2];
    uint16_t key64[1024 * 76][4];
    for (int i = 0; i < number; i++) {
        text32[i][0] = 0xffff;
        text32[i][1] = 0xffff;
        key64[i][0] = 0xffff;
        key64[i][1] = 0xffff;
        key64[i][2] = 0xffff;
        key64[i][3] = 0xffff;
    }

    uint16_t* d_text, * d_key;

    cudaMalloc((void**)&d_text, sizeof(uint16_t) * number * 2);
    cudaMalloc((void**)&d_key, sizeof(uint16_t) * number * 4);


    cudaMemcpy(d_text, text32, sizeof(uint16_t) * number * 2, cudaMemcpyHostToDevice);
    cudaMemcpy(d_key, key64, sizeof(uint16_t) * number * 4, cudaMemcpyHostToDevice);
```

allocate GPU memory

CPU → GPU

SIMECK – CUDA C(2)

```
clock_t start, end;

start = clock();

Round << <blobknum, number / blobknum >> > (d_key, d_text);

cudaMemcpy(text32, d_text, sizeof(uint16_t) * number * 2, cudaMemcpyDeviceToHost);

end = clock();

int i = 1024 * 76 - 1;
printf("Simeck32/64 Cipher : %x %x \n", text32[i][0], text32[i][1]);

printf("elapsed time : %f", (double)(end - start)/CLOCKS_PER_SEC);
```

Parallel Encryption (76 block , block per thread : 1024)

GPU → CPU

```
__global__ void Round(uint16_t* keys, uint16_t* ciphertext)
{
    int k = blockDim.x * blockIdx.x + threadIdx.x;
    uint16_t temp;
    uint16_t constant = 0xFFFC;
    uint32_t sequence = 0x9A42BB1F;

    for (int i = 0; i < 32; i++) {

        ROUND32( keys[4*k], ciphertext[2*k + 1], ciphertext[2*k], temp);

        constant &= 0xFFFC;
        constant |= sequence & 1;
        sequence >>= 1;

        ROUND32(constant, keys[4 * k + 1], keys[4 * k], temp);

        //printf("%x % x \n", ciphertext[2*k], ciphertext[2 * k + 1]);

        temp = keys[4 * k + 1];
        keys[4 * k + 1] = keys[4 * k + 2];
        keys[4 * k + 2] = keys[4 * k + 3];
        keys[4 * k + 3] = temp;
    }
}
```

- CPU Performance

```
Simeck32/64 Cipher : a92b 3527
elapsed time : 0.022000
```

- CUDA C Performance

```
Simeck32/64 Cipher : a92b 3527
elapsed time : 0.001000
```

SIMECK – CUDA PTX

- __global__ void Round part (PTX), key schedule part (omitted)

```
// Round Function
asm("mov.u16 %0, %1;"
    :"=h"(temp) : "h"(ciphertext[2 * k + 1])
);
//ROTL16 (x,5)
asm("shl.b16 %0, %1, 5;"
    :"=h"(temp_result[0]) : "h"(ciphertext[2 * k + 1])
);
asm("shr.b16 %0, %1, 11;"
    :"=h"(temp_result[1]) : "h"(ciphertext[2 * k + 1])
);
asm("or.b16 %0, %1, %2;"
    :"=h"(result[0]) : "h"(temp_result[0]), "h"(temp_result[1])
);
//ROTL16 (x,1)
asm("shl.b16 %0, %1, 1;"
    :"=h"(temp_result[0]) : "h"(ciphertext[2 * k + 1])
);
asm("shr.b16 %0, %1, 15;"
    :"=h"(temp_result[1]) : "h"(ciphertext[2 * k + 1])
);
asm("or.b16 %0, %1, %2;"
    :"=h"(result[1]) : "h"(temp_result[0]), "h"(temp_result[1])
);
asm("and.b16 %0, %1, %2;"
    :"=h"(ciphertext[2 * k + 1]) : "h"(ciphertext[2 * k + 1]), "h"(result[0])
);
asm("xor.b16 %0, %1, %2;"
    :"=h"(ciphertext[2 * k + 1]) : "h"(ciphertext[2 * k + 1]), "h"(result[1])
);
asm("xor.b16 %0, %1, %2;"
    :"=h"(ciphertext[2 * k + 1]) : "h"(ciphertext[2 * k + 1]), "h"(ciphertext[2 * k])
);
asm("xor.b16 %0, %1, %2;"
    :"=h"(ciphertext[2 * k + 1]) : "h"(ciphertext[2 * k + 1]), "h"(keys[4 * k])
);
asm("mov.u16 %0, %1;"
    :"=h"(ciphertext[2 * k]) : "h"(temp)
);
```

```
#define LROT16(x, r) (((x) << (r)) | ((x) >> (16 - (r))))

#define ROUND32(key, lft, rgt, tmp)  { \
    tmp = (lft); \
    lft = ((lft) & LROT16((lft), 5)) ^ LROT16((lft), 1) ^ (rgt) ^ (key); \
    rgt = (tmp); \
}
```

- CUDA C Performance

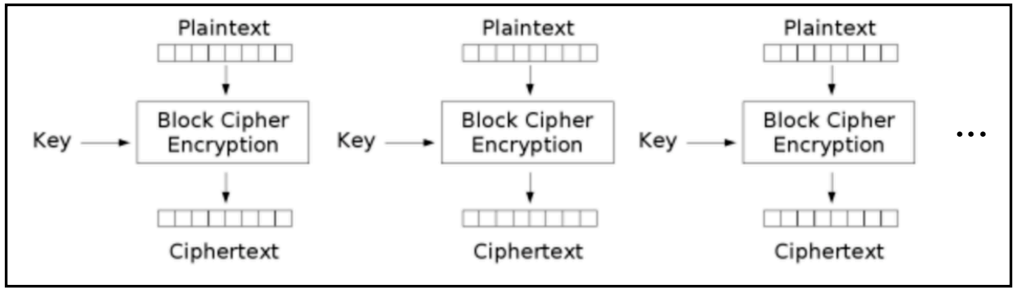
```
Simeck32/64 Cipher : a92b 3527
elapsed time : 0.001000
```
Faster

- CUDA My PTX performance

```
Simeck32 / 64 Cipher a92b 3527
elapsed time : 0.001000
```

Performance

- Parallel encryption (1024 x 76)



  - CPU Performance

```
Simeck32/64 Cipher : a92b 3527
elapsed time : 0.022000
```

  - CUDA C Performance

```
Simeck32/64 Cipher : a92b 3527
elapsed time : 0.001000
```
Faster

  - CUDA My PTX performance

```
Simeck32 / 64 Cipher a92b 3527
elapsed time : 0.001000
```

Thank you