

# A Galois Field Arithmetic Library

Pakize ŞANAL, MSc Candidate

Supervisor: Asst. Prof. Hüseyin HİŞİL

Yasar University  
Faculty of Engineering  
Department of Computer Engineering

June 5, 2017



# Outline

Content of the bachelor thesis

Studied assembly optimizations

Test results



# Content of the bachelor thesis

## A Galois Field Arithmetic Library

- ▶  $+, -, \cdot$ .
- ▶  $GF(2^w - c)$  where  $w = 127, 128, 255, 256$  and  $GF(2^{127} - 1)$ .
- ▶ Constant time AMD64 Assembly.
- ▶ Extensive validation and performance tests.



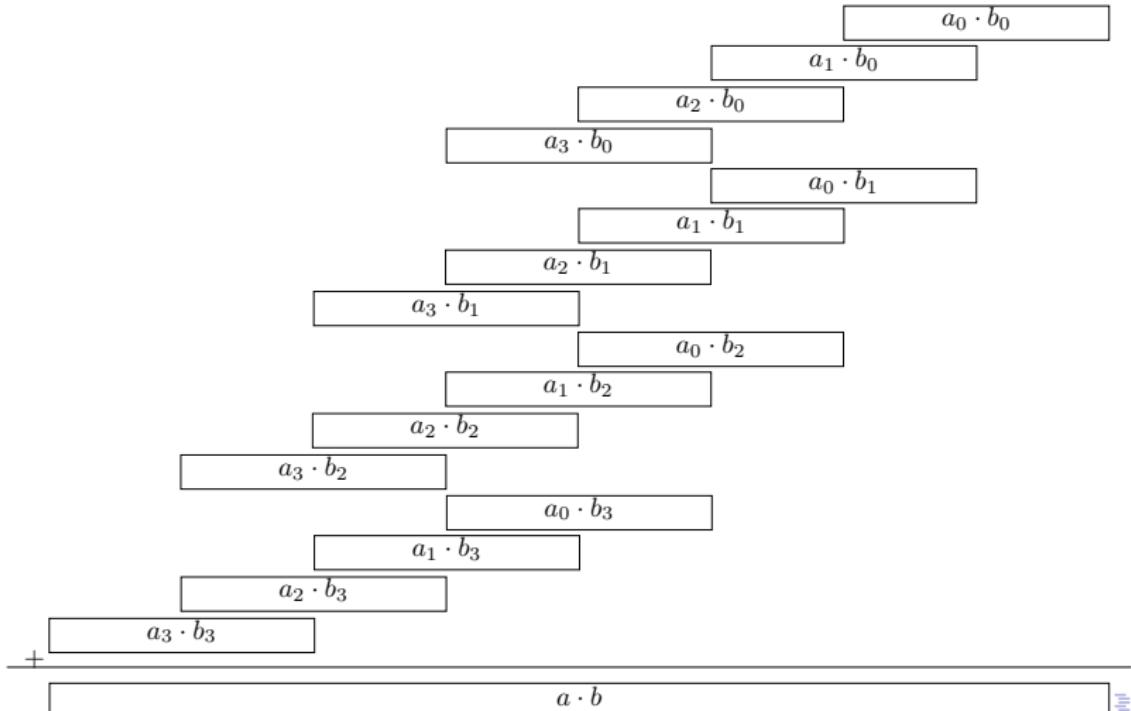
# 1. By scheduling of the operations

Four digits schoolbook vs. one level recursive schoolbook multiplication vs. ...

	SCB	RSCB	OSCB
$2^{256} - c$	38	-	-

$a_3$	$a_2$	$a_1$	$a_0$
$b_3$	$b_2$	$b_1$	$b_0$

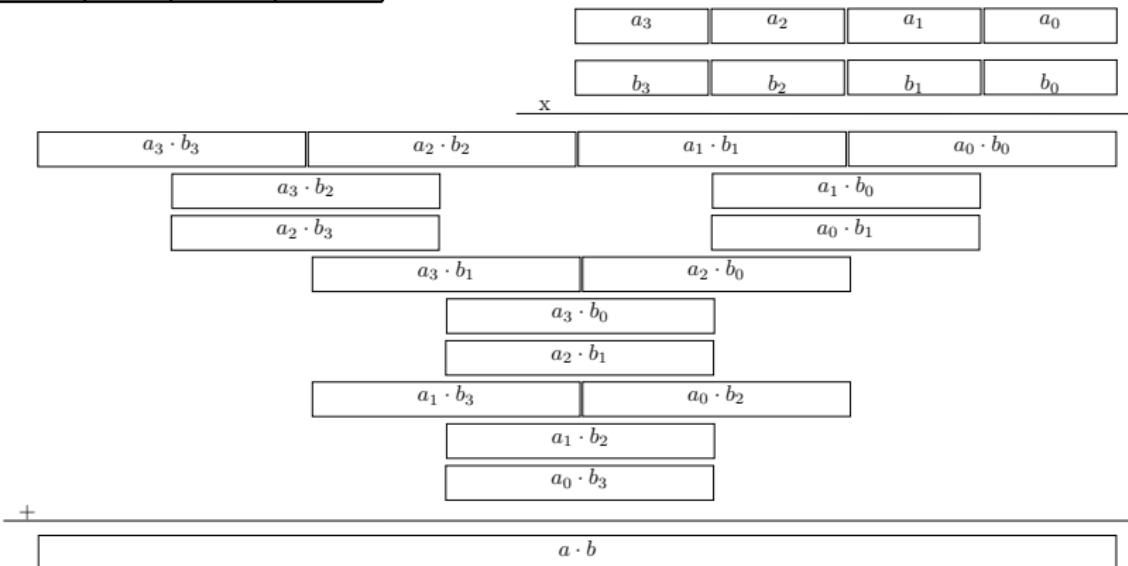
x



# 1. By scheduling of the operations

Four digits schoolbook vs. one level recursive schoolbook multiplication vs. ...

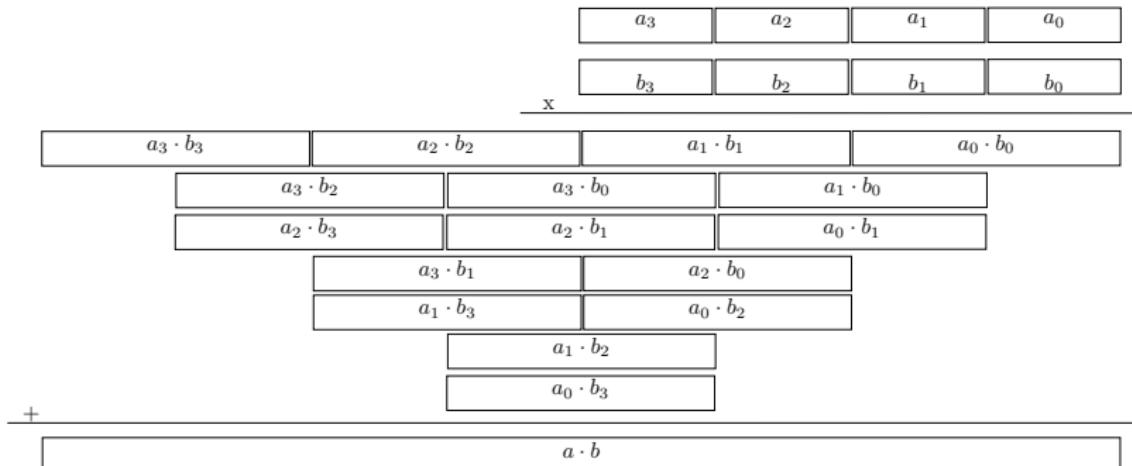
	SCB	RSCB	OSCB
$2^{256} - c$	38	35	-



# 1. By scheduling of the operations

Four digits schoolbook vs. one level recursive schoolbook multiplication vs. ...

	SCB	RSCB	OSCB
$2^{256} - c$	38	35	37



# 1. By scheduling of the operations

One level Karatsuba multiplication vs. one level schoolbook multiplication

	Karatsuba	SCB
$2^{127} - 1$	12	6
$2^{127} - c$	17	13
$2^{128} - c$	12	10

$$\begin{array}{c} a_1 \quad a_0 \\ \hline b_1 \quad b_0 \end{array}$$

$$\begin{array}{c} a_1 \cdot b_1 \quad a_0 \cdot b_0 \\ \hline (a_1 + a_0) \cdot (b_1 + b_0) \\ a_1 \cdot b_1 \\ a_0 \cdot b_0 \\ \hline a \cdot b \end{array}$$

Diagram illustrating the Karatsuba multiplication algorithm:

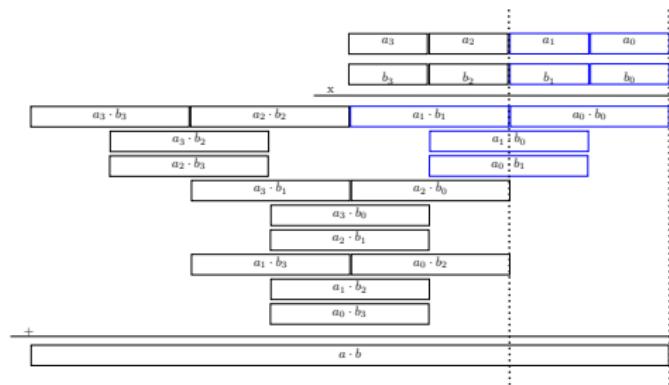
- The input numbers  $a$  and  $b$  are split into  $a_1, a_0$  and  $b_1, b_0$  respectively.
- The multiplication is computed using the formula:  $a \cdot b = (a_1 + a_0) \cdot (b_1 + b_0) - a_1 \cdot b_1 - a_0 \cdot b_0 + a_1 \cdot b_1$ .
- Arrows indicate the intermediate steps: a double-headed arrow between  $a_1 + a_0$  and  $b_1 + b_0$ , and a single arrow pointing from  $a_1 \cdot b_1$  to the subtraction step.



## 2. By making optimization

### Register optimization

```
1 //...
2 movq 8*0(%r8), %rax
3 mulq 8*0(%r9)
4 movq %rax, %rbx
5 movq %rdx, %rsi
6 movq 8*1(%r8), %rax
7 mulq 8*1(%r9)
8 movq %rax, %r10
9 movq %rdx, %r11
10 movq 8*1(%r8), %rax
11 mulq 8*0(%r9)
12 addq %rax, %rsi
13 adcq %rdx, %r10
14 adcq $0, %r11
15 movq 8*0(%r8), %rax
16 mulq 8*1(%r9)
17 addq %rax, %rsi
18 adcq %rdx, %r10
19 adcq $0, %r11
20 movq %rbx, 8*0(%rdi)
21 movq %rsi, 8*1(%rdi)
22 //...
```



Listing 1 :  $< GF(2^{255} - c), * >$



### 3. By using special instructions

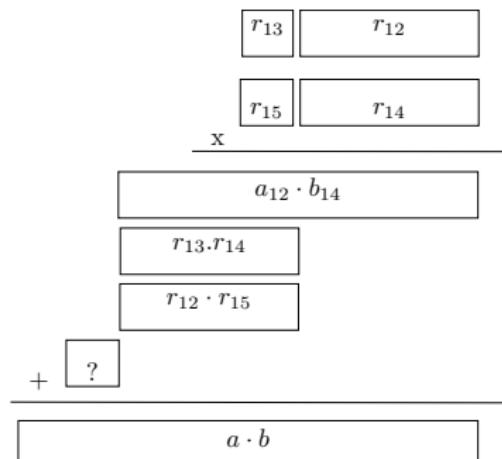
The instruction `cmove`

#### Conditional Move

```
1 //...
2     movq %r12, %rax
3     mulq %r14
4     movq $0, %rbp
5     cmp $0, %r13
6     cmovz %rbp, %r14
7     cmp $0, %r15
8     cmovz %rbp, %r12
9     andq %r13, %r15
10    addq %r12, %rdx
11    adcq $0, %rbp
12    addq %r14, %rdx
13    adcq %r15, %rbp
14 //...
```

```
if r13 = 0 then
| Return 0.
else
| Return r14.
end
```

```
if r12 = 0 then
| Return 0.
else
| Return r15.
end
```



Listing 2 :  $< GF(2^{128} - c), * >$



### 3. By using special instructions

The instruction `btqx`

Bit Test and Reset

```
1 //...
2 /*r11, r10, r9, r8*/
3 shlq $1, %r11
4 btrq $63, %r10
5 adcq $0, %r11
6 shlq $1, %r10
7 btrq $63, %r9
8 adcq $0, %r10
9
10 addq %r8, %r10
11 adcq %r9, %r11
12
13 btrq $63, %r11
14 adcq $0, %r10
15 adcq $0, %r11
16 //...
```



Listing 3 :

$\langle GF(2^{127} - 1), * \rangle$

*Faster compact Diffie-Hellman: Endomorphisms on the  $x$ -line*

C. Costello, H. Hisil, and B. Smith

### 3. By using special instructions

Comparing with the MPFQ library <  $GF(2^{127} - 1)$ , \* >

45 instructions, 9 clock cycles

33 instructions, 6 clock cycles

```
1  //...
2  /*r11, r10, r9, r8*/
3  shlq $1, %r11
4  btrq $63, %r10
5  adcq $0, %r11
6  shlq $1, %r10
7  btrq $63, %r9
8  adcq $0, %r10
9
10 addq %r8, %r10
11 adcq %r9, %r11
12
13 btrq $63, %r11
14 adcq $0, %r10
15 adcq $0, %r11
16 //...
```

Listing 4 : My schoolbook's code  
reduction part

```
1  //... /*r11, r10, r9, r8*/
2  movq $9223372036854775807, %rax
3  movq %r9, %r12
4  andq %rax, %r9
5  shrq $63, %r12
6  movq %r10, %rdx
7  shlq $1, %r10
8  orq %r10, %r12
9  shlq $1, %r11
10 shrq $63, %rdx
11 orq %r11, %rdx
12 addq %r12, %r8
13 adcq %rdx, %r9
14 movq %r9, %r12
15 andq %rax, %r9
16 shrq $1, %r12
17 adcq $0, %r8
18 adcq $0, %r9
19 //...
```

Listing 5 : MPFQ schoolbook's code  
reduction part

<https://www.imsa.res.in/~ecc14/slides/hisil.pdf>

## Test Results

Timing benchmarks were taken on an Intel Core i7-6500U processor running Ubuntu 14.04.5 LTS with TurboBoost disabled and all cores but one are switched-off (i.e. hyperthreading is disabled). To obtain the executables, we used GNU-gcc version 4.8.4 with the `-O2` flag set and GNU assembler version 2.24.

	Karatsuba	Schoolbook (SCB)	Recursive SCB
$2^{127} - 1$	12	6	-
$2^{127} - c$	17	13	-
$2^{128} - c$	12	10	-
$2^{255} - c$	-	46	40
$2^{256} - c$	-	38	34



```

1  /* libraries */
2  #define TRIAL 1000000000000
3  int main() {
4      long long st, fn;
5      st = cpucycles();
6      unsigned long an[2], bn[2], cn[2];
7      an[0] = (unsigned long) rand() * (unsigned long) rand();
8      an[1] = (unsigned long) rand() * (unsigned long) rand();
9      bn[0] = (unsigned long) rand() * (unsigned long) rand();
10     bn[1] = (unsigned long) rand() * (unsigned long) rand();
11     cn[0] = (unsigned long) rand() * (unsigned long) rand();
12     cn[1] = (unsigned long) rand() * (unsigned long) rand();
13     unsigned long int i;
14     for (i = 0; i < TRIAL; i++) {
15         mul127_scb_v01(an, bn, cn);
16         an[0] = bn[1];
17         an[1] = cn[0];
18         bn[0] = an[1];
19         bn[1] = cn[1];
20         cn[0] = an[1];
21         cn[1] = bn[0];
22     }
23     fn = cpucycles();
24     double first = ((double) fn - st) / TRIAL;
25     st = cpucycles();
26     for (i = 0; i < TRIAL; i++) {
27         mul127_scb_test(an, bn, cn);
28         an[0] = bn[1];
29         an[1] = cn[0];
30         bn[0] = an[1];
31         bn[1] = cn[1];
32         cn[0] = an[1];
33         cn[1] = bn[0];
34     }
35     fn = cpucycles();
36     double second = ((double) fn - st) / TRIAL;
37     printf("net clock cycle : %lf\n\n", first - second);
38     return 1;
39 }

```



Listing 6 : A performance test