Our LUT Technique

- Uses mullut a 128 kbyte LookUp Table
- Calculates output one byte/one word at a time instead of first calculating partial products and then adding them
- Two versions
  - B-LUT: generates output one byte at a time
  - F-LUT: generates output one word (32-bits) at a time

The LookUp Table (LUT)

The algorithm precomputes the product of all polynomials up to degree 7 with coefficients in GF(2).

Algorithm 3 Precomputation of LUT *mullut* for Lookup Table Based $GF(2^m)$ Multiplication

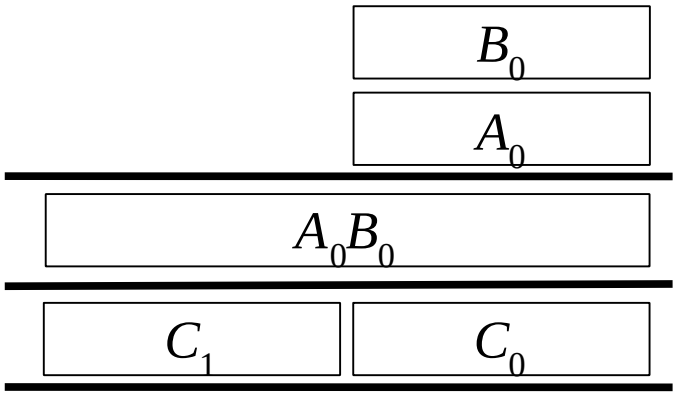
Input: none
Output: $mullut[256][256]$

```
 1: for i from 0 to 255 do
 2:     for j from 0 to 255 do
 3:         m ← i
 4:         mullut[i][j] ← 0
 5:         for k from 0 to 7 do
 6:             if k-th bit of j is 1 then
 7:                 mullut[i][j] ← mullut[i][j] ⊕ m
 8:             end if
 9:             m ← m ≪ 1
10:         end for
11:     end for
12: end for
13: output mullut[256][256]
```

Multiplication using *mullut[256][256]*

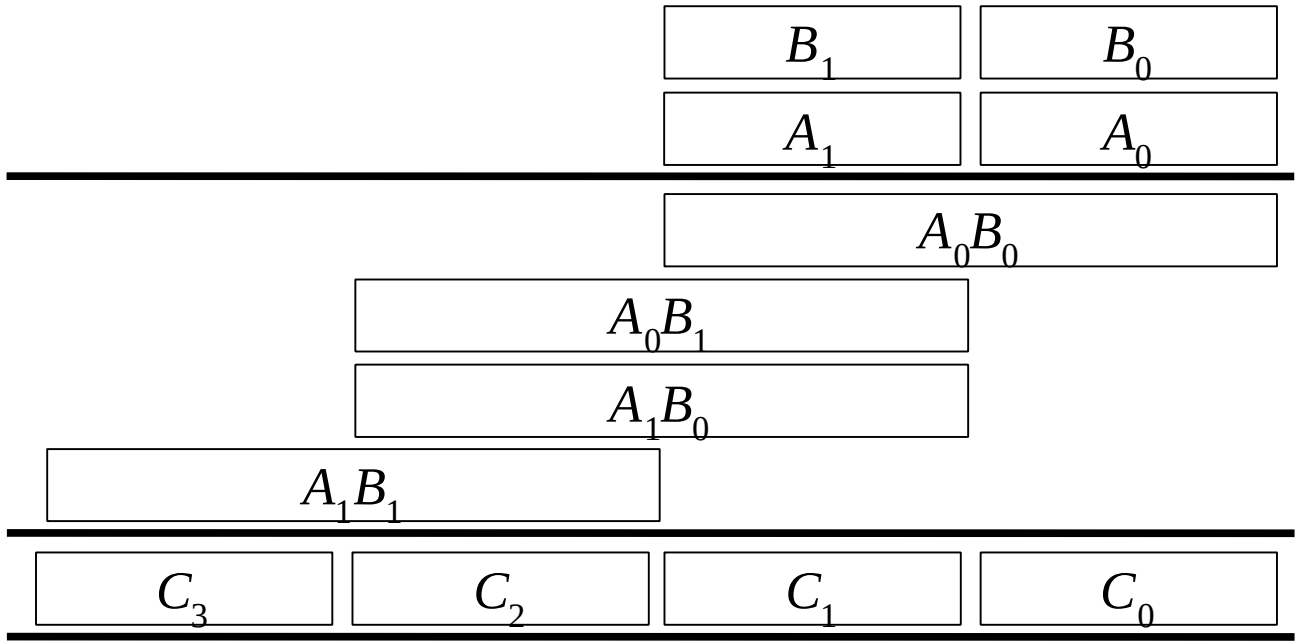
A_i and B_i are 8-bit values

8 × 8 multiplication



$C_0 = A_0B_0$ & *0x*FF
$C_1 = (A_0B_0$ & 8) & *0x*FF

16 × 16 multiplication



$C_0 = A_0B_0$ & *0x*FF
$C_1 = ((A_0B_0$&8) &*0x*FF)⊕(A_0B_1& *0x*FF)⊕(A_1B_0 & *0x*FF)
$C_2 = (($A_0B_1$ □8) &*0x*FF)⊕((A_1B_0 □8) & *0x*FF)⊕(A_1B_1& *0x*FF)
$C_3 = (A_1B_1$ □8) & *0x*FF

32 × 32 multiplication using *mullut[256][256]*

A_i and B_i are 8-bit values

Each C_i depends on A_i and B_i whose indices add up to i

Shaded portions indicate something that effects neighbours

Effects of processing in word-size is that bytes of the boundaries of words have to result in some sort of “carry”

The Pattern Emerges

$C_0 = A_0B_0 \,\&\, 0x\text{FF}$

$C_1 = ((A_0B_0 \,\&\, 8) \,\&\, 0x\text{FF}) \oplus (A_0B_1 \,\&\, 0x\text{FF}) \oplus (A_1B_0 \,\&\, 0x\text{FF})$

$C_2 = ((A_0B_1 \square 8) \,\&\, 0x\text{FF}) \oplus ((A_1B_0 \square 8) \,\&\, 0x\text{FF}) \oplus (A_0B_2 \,\&\, 0x\text{FF}) \oplus (A_1B_1 \,\&\, 0x\text{FF}) \oplus (A_2B_0 \,\&\, 0x\text{FF})$

$C_3 = ((A_0B_2 \,\&\, 0x\text{FF}) \square 8) \oplus ((A_1B_1 \,\&\, 0x\text{FF}) \square 8) \oplus ((A_2B_0 \,\&\, 0x\text{FF})) \square 8) \oplus (A_0B_3 \,\&\, 0x\text{FF})$
$\oplus (A_1B_2 \,\&\, 0x\text{FF}) \oplus (A_2B_1 \,\&\, 0x\text{FF}) \oplus (A_3B_0 \,\&\, 0x\text{FF})$

$C_4 = ((A_0B_3 \square 8) \,\&\, 0x\text{FF}) \oplus ((A_1B_2 \square 8) \,\&\, 0x\text{FF}) \oplus ((A_2B_1 \square 8) \,\&\, 0x\text{FF}) \oplus ((A_3B_0 \square 8) \,\&\, 0x\text{FF})$
$\oplus (A_1B_3 \,\&\, 0x\text{FF}) \oplus (A_2B_2 \,\&\, 0x\text{FF}) \oplus (A_3B_1 \,\&\, 0x\text{FF})$

$C_5 = ((A_1B_3 \square 8) \,\&\, 0x\text{FF}) \oplus ((A_2B_2 \square 8) \,\&\, 0x\text{FF}) \oplus ((A_3B_1 \square 8) \,\&\, 0x\text{FF})$
$\oplus (A_2B_3 \,\&\, 0x\text{FF}) \oplus (A_3B_2 \,\&\, 0x\text{FF})$

$C_6 = ((A_2B_3 \square 8) \,\&\, 0x\text{FF}) \oplus (A_3B_2 \square 8) \,\&\, 0x\text{FF}) \oplus (A_3B_3 \,\&\, 0x\text{FF})$

$C_7 = (A_3B_3 \square 8) \,\&\, 0x\text{FF}$

B-LUT

Algorithm 4 Basic Lookup Table Based $GF(2^m)$ Multiplication (B-LUT)

Input: $a, b \in GF(2^m)$ stored as array of machine words $a(x) = A_{s-1}A_{s-2}\dots A_0$ and $b(x) = B_{s-1}B_{s-2}\dots B_0$

Output: $c(x) = a(x) \cdot b(x) \bmod p(x)$

1: $d \leftarrow \lceil m/8 \rceil, c \leftarrow 0$
2: **for** k from 0 to $2 \cdot d$ by 1 **do**
3: $\quad j \leftarrow k, carry \leftarrow 0$
4: $\quad$ **for** l from 0 to k by 1 **do**
5: $\quad\quad$ **if** $l < d$ and $j < d$ **then**
6: $\quad\quad\quad$ **if** $k \bmod 4 = 0$ **then**
7: $\quad\quad\quad\quad C_{k/4} \leftarrow C_{k/4} \oplus (mullut[(A_{l/4} \gg ((l \bmod 4)8))\&0xFF][(A_{m/4} \gg ((j \bmod 4)8))\&0xFF]) \ll (((l+j) \bmod 4)8) \oplus carry$
8: $\quad\quad\quad\quad carry \leftarrow 0$
9: $\quad\quad\quad$ **else**
10: $\quad\quad\quad\quad C_{k/4} \leftarrow C_{k/4} \oplus (mullut[(A_{l/4} \gg ((l \bmod 4)8))\&0xFF][(A_{m/4} \gg ((j \bmod 4)8))\&0xFF]) \ll (((l+j) \bmod 4)8)$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad\quad$ **if** $(((l+j) \bmod 4)8 = 24)$ **then**
13: $\quad\quad\quad\quad carry \leftarrow carry \oplus (mullut[(A_{l/4} \gg ((l \bmod 4)8))\&0xFF][(A_{m/4} \gg ((j \bmod 4)8))0xFF]) \gg 8$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: $\quad j \leftarrow j - 1$
18: **end for**
19: **return** $c(x) \bmod p(x), c(x) = C_{s-1}C_{s-2}\dots C_0$

- We process 8 bits of the inputs at a time.
- The number of 8-bit units present in the inputs a and b is given as d.
- Output c before reduction would consist of 2·d bytes.
- Each byte of the output is calculated by performing lookups into the table mullut and adding (XOR) looked up values after they have been shifted by necessary amounts.
- The symbol & represents the bit-wise AND operation.

GF(2^{16}) Multiply Using mullut

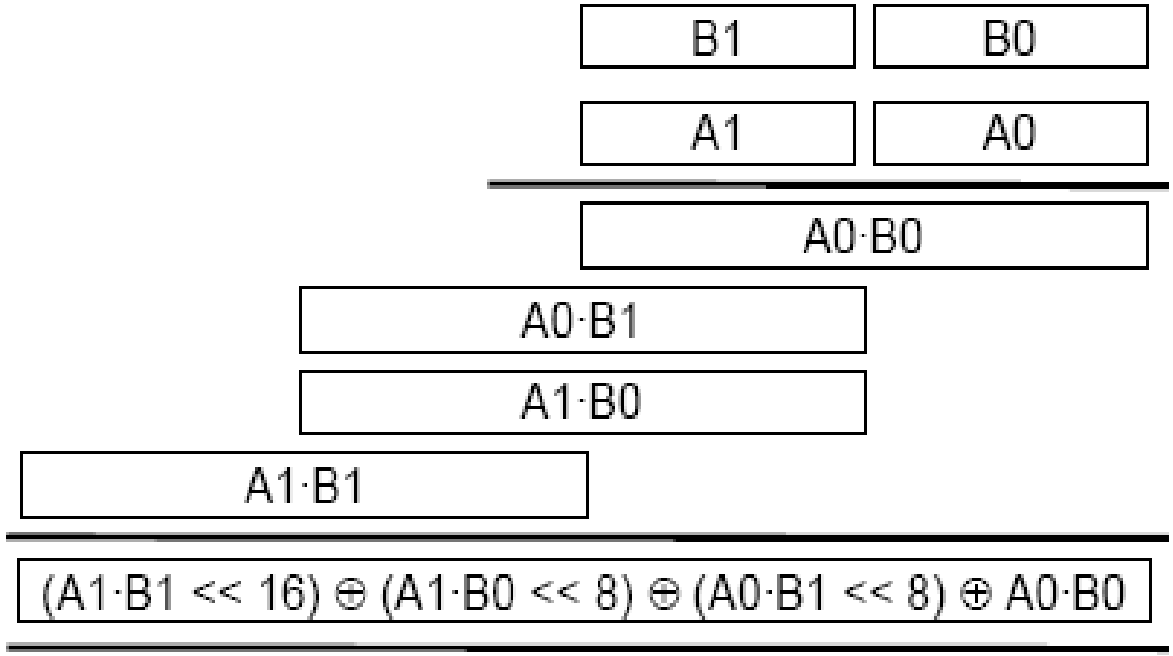


Fig. 1. A 16 by 16 Multiply Using mullut

F-LUT

Algorithm 5 Fast Lookup Table Based $GF(2^m)$ Multiplication (F-LUT)

Input: $a, b \in GF(2^m)$ stored as array of machine words $a(x) = A_{s-1}A_{s-2}\dots A_0$ and $b(x) = B_{s-1}B_{s-2}\dots B_0$ and $mullut[256][256]$

Output: $c(x) = a(x) \cdot b(x) \bmod p(x)$

1: $carry \leftarrow 0, carry1 \leftarrow 0, carry2 \leftarrow 0, h \leftarrow 0, temps[2s^2]$ is an array of machine words of size w each.
2: **for** i from 0 to $2s-1$ **do**
3: $C_i \leftarrow 0$
4: $j \leftarrow i$
5: **for** k from 0 to i **do**
6: **if** $k < s$ and $j < s$ **then**
7: $carry \leftarrow 0$
8: $temps[2h] \leftarrow A_k \cdot B_j[0]$ and $temps[2h+1] \leftarrow A_k \cdot B_j[1]$
9: $C_i \leftarrow C_i \oplus temps[2h] \oplus carry1$
10: $carry1 \leftarrow 0$
11: $carry2 \leftarrow carry2 \oplus temps[2h+1]$
12: $h \leftarrow h+1$
13: **end if**
14: $j \leftarrow j-1$
15: **end for**
16: $carry1 \leftarrow carry2$
17: $carry2 \leftarrow 0$
18: **end for**
19: $C_{2s-1} \leftarrow A_{s-1} \cdot B_{s-1}[1]$
20: **return** $c(x) \bmod p(x), c(x) = C_{s-1}C_{s-2}\dots C_0$

Table 2. Number of Operations for Base Case of $GF(2^{32})$ multiply using *mullut*

XOR X_B	AND A_B	SHIFTS S_B	Table Lookups T_B
19	8	22	16

Comparison with Other Techniques

Table 3. Number of Word Level Operations Required for Different $GF(2^m)$ Multiplication Techniques

Method	XORs	Shifts	ANDs	Table Lookups
Shift-Add	$3 \cdot w \cdot s^2$	$4 \cdot w \cdot s^2 + w \cdot s$	0	0
Shift-Add with Window Size u	$4 \cdot \frac{w}{u} \cdot s^2 + 3 \cdot 2^u \cdot u \cdot s$	$6 \cdot \frac{w}{u} \cdot s^2 + 2^{u+2} \cdot u \cdot s + \frac{w}{u} \cdot s$	$\frac{w}{u} \cdot s$	$2 \cdot \frac{w}{u} \cdot s^2$
Basic LUT	$\frac{3}{2}d^2$	$\frac{15}{4}d^2$	$\frac{10}{4}d^2$	$\frac{5}{4}d^2$
Fast LUT	$X_B s^2$	$S_B s^2$	$A_B s^2$	$T_B s^2$

Table 4. Number of Word Level Operations Required for Various Binary Finite Fields Multiplication Techniques for $m = 163$ ($w = 32$ bits)

Technique	XOR	Shift	AND	Table Lookups
Shift and Add	3456	4800	0	0
Shift and Add with Window $u = 2$	2448	3744	96	1152
Shift and Add with Window $u = 4$	2448	3312	48	576
B-LUT	662	1654	1103	551
F-LUT	684	792	288	576

Multiplier Performance Data

Table 5. Timings (in μ sec) for Different Finite Field Multiplication Methods (Pentium IV 2.0 GHz with 512 kB L2 Cache)

m	Shift-Add	Shift-Add ($u = 2$)	Shift-Add ($u = 4$)	Shift-Add ($u = 8$)	Comb	KA	B-LUT	F-LUT
113	5.7	2.4	5.9	167	7.4	13.1	9.4	1.4
131	9.9	2.8	7.9	215	14.8	46.1	11.7	1.7
163	13.6	3.6	9.1	238	23.7	52.1	17.2	2.4
193	17.3	4.9	10.2	253	33.1	57.1	24.6	3.6
223	22.0	6.2	11.3	267	55.1	60.1	33.3	4.5
283	29.0	8.5	13.5	322	73.5	204	47.2	6.4
409	98.1	20.8	32.9	713	511	449	95.6	12.7
571	141	30.5	38.9	756	1004	1588	180	24.3

Impact on ECC Performance

Table 7. Timings (in msec) for Elliptic Curve Scalar Multiplication Using Affine Coordinates Except as Noted (Pentium IV 2.0 GHz with 512 kB L2 Cache)

m	Curve	Shift and Add ($u = 2$) and EEA Inversion	Shift and Add ($u = 2$) and FLT Inversion	F-LUT and FLT Inversion	F-LUT Projective Coordinates
113	sect113r1	6.9	4.3	2.9	2.5
131	sect131r1	12.6	6.6	5.3	4.1
163	sect163r1	20.8	11.0	8.9	6.6
193	sect193r1	34.8	19.3	16.5	13.3
233	sect233r1	49.5	27.3	22.2	14.9
283	sect283r1	83.7	48.7	41.8	27.8
571	sect571r1	746	445	382	248

Conclusions and Future Work

- **We presented:**
  - **Two new algorithms for GF(2^m) multiplication using Lookup Table and results of an implementation of the new LUT based finite field multiplication techniques.**
  - **We provided the results of our ECC implementation showing performance impact of our new GF(2^m) multiplication technique.**
- **Future ideas:**
  - **Combine Karatsuba's algorithm with our LUT based technique to cut down the base number of XORs, Shifts and Table Lookups.**
  - **We also intend to explore more efficient EC scalar multiplication techniques and use of our multiplication technique for HECC.**