Sliding window method with flexible window size for scalar multiplication on wireless sensor network nodes.

Pritam Gajkumar Shah., Xu Huang, Dharmendra Sharma, Department of ISE, University of Canberra, Australia

***Abstract*—Scalar multiplication is time consuming operation of ECC when implemented on wireless sensor nodes. The wireless sensor node consists of 8 bit micro controller and limited memory for the storage. The scalar multiplication process can be accelerated with the sliding window method which has two stages namely pre computation and an evaluation stage. Points for use in the evaluation stage are computed in the pre computation stage. The scalar multiplication is carried out in the evaluation stage with the addition of pre computed points. The number of pre computations depends on the window size of sliding window method. More is the window size, more are the pre computations and more is the memory required for the storage .This is the well-known draw-back of the sliding window method when implemented on WSN Nodes. This research paper proposes sliding window method with flexible window size for scalar multiplication on wireless sensor nodes. The flexible window size will prevent sensor node failures due to stack overflow.**

Keywords-Point multiplication, wireless sensor nodes, window size, ECC, precomputation, stack depth analysis, base segment, data segment.

WIRELESS SENSOR NETWORKS SECURITY

Compared to traditional networks, a wireless sensor network has many resource constraints. The MICA2 mote consists of an 8 bit (ATMega 128L) micro controller working on 7.3 MHz and consists of only 512Kbyte of RAM. As a result nodes of WSN have limited computational power and memory resources [1]. Due to the above limitations implementing ECDH, ECDSA schemes are expensive for sensor network due to heavy mathematical operation of scalar multiplication involved in these schemes. At the same time small and compact key sizes of ECC makes it desirable candidate for WSN security.[5]

ELLIPTIC CURVE CRYPTOGRAPHY PRELIMINARIES

An elliptic curve E over $GF(p)$ can be defined by $y^2 = x^3 + ax + b$ where $a, b \in GF(p)$ and $4a^3 + 27b^2 \neq 0$ in the $GF(p)$ [6],[7].

If there are two points on curve namely, $P(x_1, y_1)$, $Q(x_2, y_2)$ and their sum given by point $R(x_3, y_3)$ the algebraic formulas for point addition and point doubling are given by following equations:

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \quad \text{if } P \neq Q$$

$$\lambda = \frac{3x^2 + a}{2y_1}, \quad \text{if } P = Q$$

Where the addition, subtraction, multiplication and the inverse are the arithmetic operations over $GF(p)$, which are shown in Fig. 1.

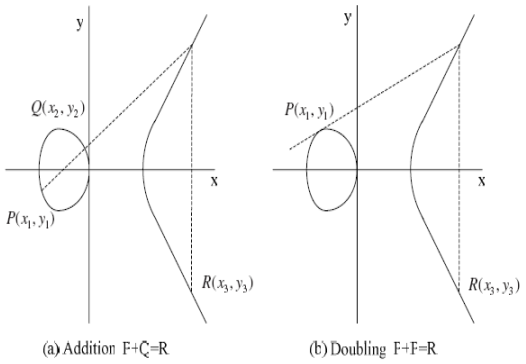


Fig. 1. Point addition and point doubling on elliptic curve

CHALLENGES INVOLVED IN THE SCALAR MULTIPLICATION

In ECC, scalar multiplication is the most time consuming operation and takes 85% of execution time [2]. Scalar multiplication is the operation of multiplying point P on an elliptic curve E defined over a field $GF(p)$ with positive integer k .

$$Q = kP = \underbrace{P + P + P + P}_{<-k-times\ -P->}$$

Scalar multiplication involves point addition and point doubling. Operational efficiency of this process is affected by the type of coordinate system used for point P, the algorithm used for recoding of integer k in scalar multiplication.

METHODS OF SCALAR MULTIPLICATION

As shown in algorithm 1 scalar multiplication is achieved by repeated point addition and doubling operations. In the binary method the integer k is represented in binary form:

$$k = \sum_{j=0}^{l-1} K_j 2^j, \; K_j \in \{0,1\}$$

The Binary method scans the bits of K either from left-to-right or right-to-left.

Algorithm 1: Left to right binary method for point multiplication [5]

Input: A point $P \in E(\mathcal{F}_q)$, an

l bits integer $k = \sum_{j=o}^{l-1} K_j 2^j, \quad K_j \in \{0,1\}$

Output: $Q = kP$

1. $Q \leftarrow \infty$
2. For $j = l - 1$ to 0 do:
   - 2.1 $Q \leftarrow 2Q$,
   - 2.2 if $k_j = 1$ the $Q \leftarrow Q + P$.

Return Q.

The cost of multiplication in binary method depends on the number of non zero elements and length of the binary representation of k. If the representation has $k_{l-1} \neq 0$ then binary method require $(l - 1)$ point doubling and $(W\text{-}1)$ where l is the length of the binary expansion of k and W is the Hamming weight of the k that is the number of non-zero elements in expansion of k.

For example if $k = 629 = (1001110101)_2$, it will require $(W\text{-}1) = 6 - 1 = 5$ point additions and $l - 1 = 10 - 1 = 9$ point doubling operations.

POINT MULTIPLICATION WITH THE SLIDING WINDOW METHOD

Binary method can be improved by scanning few bits at a time as with sliding window method. This method processes a window of length r where $r > 1$ disregarding fixed digit boundaries, and skips runs of zeros between them as shown in Algorithm 2.[4] These runs are taken care by point doubling.

Algorithm 2: Sliding window Method for Scalar Multiplication

Input : A point P, an integer $k = \sum_{j=0}^{L-1} k_j 2^j, k_j \in \{0,1\}$.

Output : $Q = [k]P$.

Precomputations.

1. $P_1 \leftarrow P, P_2 \leftarrow [2]P$.

2. For $i = 1$ to $2^{r-1} - 1$ do $P_{2i+1} \leftarrow P_{2i-1} + P_2$.

3. $j \leftarrow L - 1, Q \leftarrow \vartheta$.

MainLoop.

4 While $j \geq 0$ do :

5. if $k_j = 0$ then $Q \leftarrow [2]Q, j \leftarrow j - 1$.

6. Else Do :

7. Let t be the least integer such that

$j - t + 1 \leq r$ and $k_t = 1$,

8. $h_j \leftarrow (k_j k_{j-1} \ldots\ldots k_t)_2$,

9. $Q \leftarrow [2^{j-t+1}]Q + P_{h_j}$,

10. $j \leftarrow t - 1$.

11. Return Q

For the above algorithm the number of pre computations required for unsigned binary number is given by 2^{r-1} and for signed binary number are given by $2^{r-1} - 1$ where r is the size of window and is greater than 1.

Let us compute Q=[763] P with sliding window algorithm with K recoded in Binary form with different window sizes ranging from 2 to 10.

It is observed that as the window size increases the number of pre computations also increases geometrically. At the same time number of additions and doubling operations decreases.

- **Let us compute Q= [763] P with sliding window with window sizes ranging from 2 to 10**
- **763 = $[1011111011]_2$**

- Window Size $w = 2$
- No of precomputation = 2
- 2P,3P
- 763 = 10 11 11 10 11 [The fonts indicated with brown colour shows window boundary]
- The intermediate values of Q are ,
- P, 2P 4P, 8P, 11P, 22P, 44P, 47P, 94P, 95P, 190P, 380P, 760P, 763P
- Computational cost = 9 doubling, 4 additions, and 2 precomputation.

- Window Size $w = 3$
- No of precomputation = 4
- 2P, 3P, 5P, 7P
- 763 = 101 111 101 1
- = 5P 7P 5P 1P
- The intermediate values of Q are ,
- 5P, 10P, 20P, 40P, 47P, 94P, 188P, 376P, 381P, 762P, 763P
- Computational cost = 7 doublings, 3 additions, and 4 precomputation.

- Window Size $w = 4$
- No of precomputation = 8
- 2P, 3P, 5P, 7P, 9P, 11P, 13P, 15P
- 763 = 1011 111 0 11
- = 11P 7P 3P
- The intermediate values of Q are
- 11P, 22P, 44P, 88P, 95P, 190P, 380P, 760P, 763P
- Computational cost = 6 doublings, 2 additions, and 8 precomputation.

- Window Size $w = 5$
- No of precomputation = 16
- 2P, 3P, 5P, 7P, 9P, 11P, 13P, 15P, 17P, 19P, 21P, 23P, 25P, 27P, 29P, 31P
- 763 = 10111 11011
- = 23P 27P
- The intermediate values of Q are
- 23P, 46P, 92P, 184P, 368P, 736P, 763P
- Computational cost = 5 doublings, 1 additions, and 16 precomputation

- Window Size $w = 6$
- No of precomputation = 32.
- 2P,3P, 5P, 7P, 9P,11P,13P,15P,17P,19P, 21P, 23P, 25P, 27P, 29P, 31P, 33P, 35P, 37P, 39P, 41P, 43P, 45P, 47P, 49P, 51P, 53P, 55P, 57P, 59P, 61P
- 763 = 101111 1011
- The intermediate values of Q are
- 47P, 94P, 188P, 376P, 752P, 763P

- Computational cost = 4 doublings, 1 additions, and 32 precomputation

- Window Size $w = 7$
- No of precomputation = 64
- 2P,3P, 5P, 7P, 9P,11P,13P,15P,17P, 19P, 21P, 23P, 25P, 27P, 29P, 31P, 33P, 35P, 37P, 39P, 41P, 43P, 45P, 47P, 49P, 51P, 53P, 55P, 57P, 59P, 61P, 63P, 65P, 67P, 69P, 71P, 73P, 75P, 77P, 79P, 81P, 83P, 85P, 87P, 89P, 91P, 93P, 95P, 97P, 99P, 101P, 103P, 105P, 107P, 109P, 111P, 113P, 115P, 117P, 119P, 121P
- 763 = 1011111 011
- The intermediate values of Q are
- 95P, 190P, 380P, 760P, 763P
- Computational cost = 3 doublings, 1 additions, and 64 precomputation.

- Window Size $w = 8$
- No of precomputation = 128
- 2P,3P, 5P, 7P, 9P, 11P,13P,15P, 17P, 19P, 21P, 23P, 25P, 27P, 29P, 31P, 33P, 35P, 37P, 39P, 41P, 43P, 45P, 47P, 49P, 51P, 53P, 55P, 57P, 59P, 61P, 63P, 65P, 67P, 69P, 71P, 73P, 75P, 77P, 79P, 81P, 83P, 85P, 87P, 89P, 91P, 93P, 95P, 97P, 99P, 101P, 103P, 105P, 107P, 109P, 111P, 113P, 115P, 117P, 119P, 121P, 123P, 125P, 127P, 129P, 131P, 133P, 135P, 137P, 139P, 141P, 143P, 145P, 147P, 151P, 153P, 155P, 157P, 159P, 161P, 163P, 165P, 167P, 169P, 171P, 173P, 175P, 177P, 179P, 181P, 183P, 185P, 187P, 189P, 191P, 193P, 195P, 197P, 199P, 201P, 203P, 205P, 207P, 209P, 211P, 213P, 215P, 217P, 219P, 221P, 223P, 225P, 227P, 229P, 231P, 233P, 235P, 237P, 241P, 243P, 245P, 247P, 249P, 251P, 253P, 255P
- 763 = 1011111 0 11
- The intermediate values of Q are
- 95P, 190P, 380P, 760P, 763P
- Computational cost = 3 doublings, 1 additions, and 128 precomputation.

- Window Size $w = 9$
- No of precomputation = 256
- 763 = 101111101 1
- The intermediate values of Q are
- 381P, 762P, 763P
- Computational cost = 1 doublings, 1 additions, and 256 precomputation.

- Window Size $w = 10$
- No of precomputation = 512
- = 1011111011
- The intermediate values of Q are
- 763P
- Computational cost = 0 doublings, 0 additions, and 512 precomputation.

EFFECT OF WINDOW SIZE ON THE MEMORY UTILIZATION OF MICA MOTES:

It is easy to see that the sliding window method will increase both the ROM [for additional code size]and RAM [for storing the pre-computed points] consumptions of wireless sensor nodes .The Table 1 and 2 gives performance results which were measured Tiny ECC[1] group for sliding window method for window size of 2 and 4 ..

Parameters	MICAz Max RAM 4K		TelosB Max RAM 10K	
	ROM bytes	RAM bytes	ROM bytes	RAM bytes
secp128r1	13,120	762	12,528	830
secp128r2	13,102	762	12,494	830
secp160k1	13,912	938	12,566	1,006
secp160r1	13,880	938	12,560	1,006
secp160r2	13,988	938	12,564	1,182
secp192k1	13,510	1,114	12,628	1,182
secp192r1	13,204	1,114	12,718	1,182

Table1 Memory Utilization by Tiny ECC for Window Size=2. [3]

Parameters	MICAz Max RAM 4K		TelosB Max RAM 10K	
	ROM bytes	RAM bytes	ROM bytes	RAM bytes
secp128r1	13,094	1,168	12,532	1,254
secp128r2	13,076	1,168	12,498	1,254
secp160k1	13,890	1,440	12,570	1,526
secp160r1	13,858	1,440	12,564	1,526
secp160r2	13,966	1,440	12,568	1,526
secp192k1	13,488	1,712	12,632	1,798
secp192r1	13,182	1,712	12,722	1,798

Table 2 Memory Utilization by Tiny ECC for Window Size=4. [3]

**** Secp128r1, secp128r2, secp160k1, secp160r1, secp160r2, secp192k1, secp192r1 are elliptic curve domain parameters over F_p recommended by Standards for Efficient Cryptography Group.***

Sliding window method can make signature generation and verification 1.2 times faster at the cost of dramatic RAM increase (1,262, 1,328 and 1,472 bytes for MICAz, TelosB/Tmote Sky, and Imote2, respectively.

Since MICAz and TelosB/Tmote Sky are low-end sensor platforms, they have much smaller RAM (4kB, 10kB) compared with Imote2 (256kB). Before using sliding window method, we should be very careful if the sensing application has large RAM consumption

WHAT HAPPEN IF THE REQUIRED RAM IS LARGER THAN AVAILABLE ONE?

Calling a function or handling an interrupt in WSN nodes causes stack memory to be allocated. In our case it is the function written for *scalar multiplication* by using sliding window method. When this interrupt occurs, the micro controller allocates stack memory to store precompuataional values for scalar multiplication. If the stack memory region is not large enough to hold these values, a stack overflow occurs. Stack overflow leads to corrupted RAM and subsequently to non-deterministic sensor node failures. .If the window size is it may end up in node failure due to above reasons.

PROPOSED NEW ALGORITHM OF SLIDING WINDOW WITH FLEXIBLE WINDOW SIZE

This new algorithm makes sure that the chosen window size **r** of the sliding window method for scalar muliptlication is *stack safe* and also make sure that memory is sufficient to hold all the precomputaional values.The above concept is called as *stack depth analysis*.

WHAT IS STACK DEPTH ANALYSIS?

A TinyOS application has a single stack and its size can be computed like this:

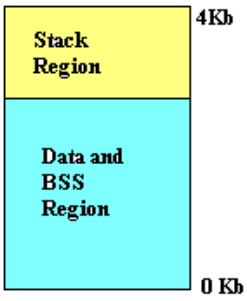


Figure2. Memroy model for TinyOS on MICA mote

Stack region size = RAM size – [data segment size + BSS segment size]

In the most basic case , the memory model for a TinyOS application on a MICA 2 or MICAz platform with 4 KB of RAM looks like in figure1.The size of each region is immediately apparent. The only remaining question is: Is the stack memory region large enough to contain the actual stack and to hold precomputed values of sliding window method?

STACK DEPTH ANALYSIS TOOL FOR TINYOS

Stack Depth Analysis can be done with following command-

```
[regehr@babel BaseStation]$ tos-ramsize micaz
./build/micaz/main.exe
```

The result should be something like:

```
BSS segment size is 1708, data segment size is 16
The upper bound on stack size is 538
The upper bound on RAM usage is 2262
There are 1834 unused bytes of RAM
```

Here output is telling us that for application , approximately 1.8 KB of RAM is free and could have been allocated for precomputations.

If you are running tos-ramsize from the build system, the result should be something like:

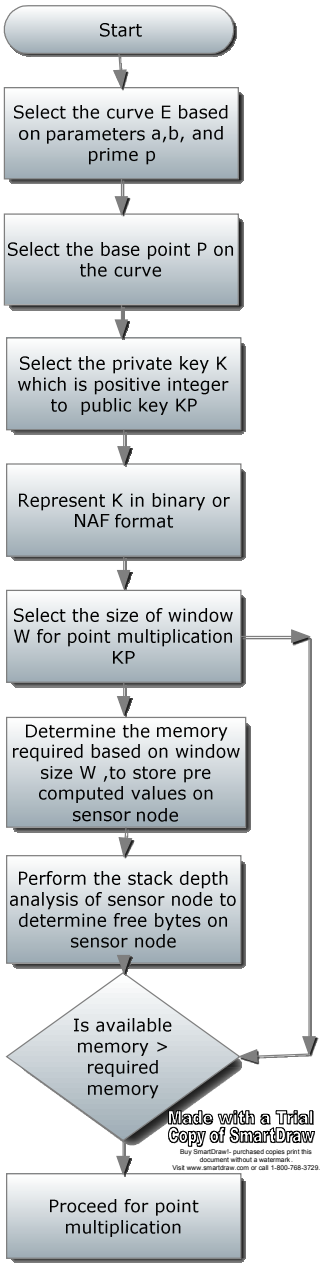
```
mkdir -p build/micaz
    compiling MultihopOscilloscopeAppC to a micaz
binary
ncc -o build/micaz/main.exe -Os -fnesc-
separator=__ -Wall
-Wshadow -Wnesc-all -target=micaz -fnesc-
cfile=build/micaz/app.c
-board=micasb -DDEFINED_TOS_AM_GROUP=0x22 --param
max-inline-insns-single=100000
-I/home/regehr/z/tinyos-2.x/tos/lib/net/
-I/home/regehr/z/tinyos-2.x/tos/lib/net/ctp
-I/home/regehr/z/tinyos-2.x/tos/lib/net/4bitle
-DIDENT_APPNAME=\"MultihopOscillo\" -
DIDENT_USERNAME=\"regehr\"
-DIDENT_HOSTNAME=\"babel\" -
DIDENT_USERHASH=0xaa57ee96L
-DIDENT_TIMESTAMP=0x49e3a884L -
DIDENT_UIDHASH=0xb4560dc8L
-fnesc-dump=wiring -fnesc-
dump='interfaces(!abstract())'
-fnesc-dump='referenced(interfacedefs,
components)'
-fnesc-dumpfile=build/micaz/wiring-check.xml
MultihopOscilloscopeAppC.nc -lm

BSS segment size is 3421, data segment size is 24
The upper bound on stack size is 578
The upper bound on RAM usage is 4023
There are 73 unused bytes of RAM

    compiled MultihopOscilloscopeAppC to
build/micaz/main.exe
           25458 bytes in ROM
            3445 bytes in RAM
avr-objcopy --output-target=srec
build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex
build/micaz/main.exe build/micaz/main.ihex
    writing TOS image
```

Here, tos-ramsize is telling that only 73 bytes of memory is available. In this case window size of 2 for ECC parameters secp128r1, in which RAM of 762 bytes will cause node failure as required memory is larger than available one i.e. 73 bytes.

CONCLUSION:

Flexible window size of sliding window method on wireless sensor network platform can considerably remove the risk of node failure. The window size and available memory trade off can be done automatically with the new proposed algorithm. The proposed algorithm can be implemented on sensor node platform with very few simple instructions and will not occupy much code size in ROM.

Figure 3: A new algorithm of sliding window method with flexible window size for scalar multiplication on WSN platform



REFERENCES:

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks,* vol. 38, pp. 393-422, 2002.

[2] N. Gura, A. Patel, and A. Wander, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES)* August 2004.

[3] An Liu, Peng Ning , "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks" in the proceedings of International Conference on Information Processing of sensor networks, 2008.

[4] Ian Blake, Gadiel Seroussi, and Nigel Smart, "Elliptical Curves in Cryptography", London Mathematical Society Lecture Notes Series 265, Cambridge University Press, 1999.

[5] Lauter, K.: The Advantages of Elliptic Curve Cryptography for Wireless Security. J. IEEE Wireless Communications 11(1), 62– 67 (2004).

[6] Koblitz, N.: Elliptic Curve Cryptosystems. J. Math. Compute. 48, 203–209 (1987).

[7] Menezes, A.: Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers, Dordrecht (1993).