

HW/SW co-design of a hyperelliptic curve cryptosystem using a microcode instruction set coprocessor

Alireza Hodjat^a, Lejla Batina^b, David Hwang^{a,*}, Ingrid Verbauwhede^{a,b}

^a*Electrical Engineering Department, University of California, Los Angeles, California, USA*

^b*Katholieke Universiteit Leuven, ESAT/SCD-COSIC, Belgium*

Abstract

Hardware/software co-design of computationally intensive cryptosystems is the preferred solution to achieve the required speed for resource-limited embedded applications. This paper presents a microcode instruction set coprocessor which is designed to work with 8-bit microcontrollers to implement a hyperelliptic curve cryptosystem. The microcode coprocessor is capable of performing a range of Galois Field operations using a dual-multiplier/dual-adder datapath and storing the intermediate results in the external RAM unit of the coprocessor. This coprocessor is programmed using the software routines of the 8-bit microcontroller which implement the HECC divisor's doubling and addition operations. The Jacobian scalar multiplication was computed in a 656 ms (7.87M cycles) on 8051 microcontroller running at 12 MHz clock frequency which is 228 times faster than the pure software implementation. This number is 78 ms (1M cycles) on the Atmel AVR microcontroller running at 12 MHz clock which is 106 times faster than the pure software implementation. Both HW/SW co-design implementations are comparable to existing HECC implementations on the 32-bit ARM7 at 80 MHz.

© 2006 Published by Elsevier B.V.

Keywords: Hardware/software co-design; Hyperelliptic curve cryptography; Galois field $GF(2^m)$; Genus 2 curves; Microcode coprocessor

1. Introduction

Security of communications or in general of digital data is achieved by various cryptographic algorithms. In particular, implementations of public key cryptography (PKC) present a challenge in a vast majority of application platforms ranging from software to hardware. Software platforms are more flexible but hardware acceleration is usually necessary for computationally intensive operations as required for PKC applications. Emerging areas such as radio frequency identification (RFID) tags and sensor networks put new requirements on implementations of PKC algorithms with firm constraints in terms of number of gates, power consumption, bandwidth, etc. The best-known and most commonly used public-key cryptosystem is RSA [1]. This is not a feasible solution for low-power and low foot-print devices. A promising candidate appears to

be a hyper/elliptic curve cryptosystem (H/ECC), but the previously mentioned requirements can probably be achieved only with the synergy of hardware and software. ECC has already proven its potential as it offers shorter certificates, lower power consumption and better performance on some platforms. In addition, ECC offers more “security per bit” than RSA, as no sub-exponential algorithm is known that solves the discrete logarithm problem in this group. However, HECC maintains all those advantages with even shorter bit-lengths. More precisely, the operand size for HECC is at least a factor of two smaller than the one of ECC, with the same level of security. This fact makes HECC a very good choice for platforms with limited resources.

This article describes a microcode instruction set coprocessor which implements a hyperelliptic curve cryptosystem. More precisely, we have implemented the HECC divisor multiplication operation on 8-bit microprocessors, which uses a small hardware module to optimize the performance. This extra hardware is a coprocessor with a dual-multiplier/dual-adder datapath, which allows for a

*Corresponding author.

E-mail addresses: ahodjat@ee.ucla.edu (A. Hodjat), davidhwang@berkeley.edu (D. Hwang).

speed-up of factor 228 and 106 when compared with the software-only solution on 8051 and AVR microcontrollers, respectively. We have re-written the formulae of Byramjee and Duquesne [2] to facilitate the divisor operations in this special case. In this way we achieved optimized divisor doubling and addition. The remainder of this paper is organized as follows. Section 2 lists some related previous work. In Section 3 some background information on HECC is given. Details of our implementation are specified in Sections 4 and 5. Section 6 presents the coprocessor's instruction set. Results are given in Section 7 and conclusions are in Section 8.

2. Related work

Algorithms for HECC and their implementations have been studied intensively in the past years. A significant amount of work has been performed on optimizing the formulae for the group operation [3,2,4]. Explicit formulae for genus 2 curves are given by Lange [4] for arbitrary fields and for various types of coordinates. There exist practical results for both software platforms (general purpose or embedded processor) and hardware devices, such as FPGAs. For embedded processors, a large amount of work is performed for the ARM platform [5]. Pelzl et al. [5] implemented the group operation of genus 2 and 3 for HECC on an ARM7 processor. They compared the results with ECC implementation (with corresponding security) and showed that HECC performance is comparable to the one of ECC. The performance for divisor scalar multiplication on the ARM microprocessor for genus 2 was further optimized in [5] and compared to genres 3 and 4. They proved that genus 3 is the fastest, requiring less than 70 ms on an ARM7 running at 80 MHz. Gura et al. [6] compared ECC and RSA on 8-bit CPUs and proved that public-key cryptography is viable on small devices, with the results favoring ECC substantially. The first complete hardware implementation of HECC was given by Boston [7]. They used Cantor's algorithm [8] to implement HECC on the VirtexII FPGA. In [9] three different architectures on an FPGA have been examined for a vast area of applications. Most of the published work dealt with binary fields. The only exception is work of Baktir et al. [10] which investigated implementation over an extension field of odd characteristic i.e. over optimal tower fields (OTF) on an ARM7. For software/hardware co-design the only relevant work that we can compare with is the one of Kumar and Paar [11]. They implemented ECC on an 8-bit AVR microcontroller with some extra hardware for field multiplications.

3. Hyperelliptic curve cryptography

3.1. Hyperelliptic curves

Hyperelliptic curve cryptography was proposed in 1988 by Kobitz [12] as a generalization of elliptic curve

cryptography. In particular, elliptic curves can be viewed as a special case of hyperelliptic curves i.e. an EC is an HEC with genus $g = 1$. Here we consider a hyperelliptic curve C of genus $g = 2$ over $\text{GF}(2^m)$, which is defined by an equation of the form:

$$C : y^2 + h(x)y = f(x) \quad \text{in } \text{GF}(2^m)[x, y], \quad (1)$$

where $h(x) \in \text{GF}(2^m)[x]$ is polynomial of degree at most g ($\text{deg}(h) \leq g$) and $f(x)$ is a monic polynomial of degree $2g + 1$ ($\text{deg}(f) = 2g + 1$). There are some more conditions that have to be fulfilled. For genus 2 curves, in the general case the following equation is used: $y^2 + (h_2x^2 + h_1x + h_0)y = x^5 + f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0$. For our implementation we used so-called type II curves [2], which are defined by $h_2 = 0, h_1 \neq 1$. In particular, the authors of [2] recommend curves of the form: $y^2 + xy = x^5 + f_3x^3 + x^2 + f_0$, since they combine simpler arithmetic with a good security level. More precisely, those curves allow for much faster divisor doubling while addition stays the same as for a general curve.

Now we introduce a group structure for specific objects created on a hyperelliptic curve. A divisor D is a formal sum of points on the hyperelliptic curve C i.e. $D = \sum m_P P$ and its degree is $\text{deg } D = \sum m_P$. Let Div denotes the group of all divisors on C and Div_0 the subgroup of Div of all divisors with degree zero. The Jacobian J of the curve C is defined as quotient group $J = \text{Div}_0/P$. Here P is the set of all principal divisors, where a divisor D is called principal if $D = \text{div}(f)$, for some element f of the function field of C ($\text{div}(f) = \sum_{P \in C} \text{ord}_P(f)P$). The discrete logarithm problem in the Jacobian is the basis of security for HECC. In practice, the Mumford representation according to which each divisor is represented as a pair of polynomials $[u, v]$ is usually used. Here, u is monic of degree 2, $\text{deg } v < \text{deg } u$ and $u|f - hv - v^2$ (so-called reduced divisors). For implementations of HECC, we need to implement the multiplication of elements of the Jacobian i.e. divisors with some scalar.

3.2. HECC algorithms

Following a top-down approach, the highest-level operation is the divisor scalar multiplication. It is implemented by the use of the so-called "non-adjacent form" i.e. as the NAF algorithm [1], which has the lowest weight among all other signed digit representations. The fact that the subtraction of divisors is as expensive as the divisor addition makes this representation beneficial. In this way, the scalar multiplication is implemented as a sequence of divisor additions/subtractions and doublings. We use projective coordinates which allow us to complete all divisor operations without inversion. Only one inversion and four multiplications are required at the end to convert back from projective to affine coordinates. We have re-written the formulae from [2] for the doubling to achieve almost full parallelism for field multiplications. We also used the same approach to get the formulae for the addition in the case of mixed coordinates. The formulae for

both, the parallelized doubling and addition are given as Tables 5 and 6.

3.3. Inversion in GF(2⁸³)

In order to convert the projective coordinates to the affine coordinates one inversion has to be done on one of the coordinates and the result has to be multiplied with the other four. Inversion in binary fields can be replaced by a chain of multiplications (and squarings). First by means of Fermat’s little theorem we have $a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2$, for all $a \in GF(2^m)$. The technique to compute this in optimal way is the basis for the idea of Itoh and Tsujii [13]. Their method is especially suited for normal basis but can be applied on polynomial basis as well. Here we consider the case for m odd, so $m - 1$ is even. Then we can write

$$\begin{aligned}
 a^{2^{m-1}-1} &= a^{(2^{m-1/2}-1)(2^{m-1/2}+1)} \\
 &= (a^{2^{m-1/2}-1})^{2^{m-1/2}} a^{2^{m-1/2}} - 1.
 \end{aligned}
 \tag{2}$$

This formula is used recursively until $m - 1$ is odd. In this case $a^{2^{m-1}-1} = aa^{2^{m-1}-2} = a(a^{2^{m-2}-1})^2$. For the case of GF(2⁸³), by repeated use of these formulae we can compute the inverse by only 90 multiplications. The total number of multiplications (or squarings) required to compute an inverse in GF(2^m) is given with $\lfloor \log_2(m - 1) \rfloor + w(m - 1) - 1$. Here $w(k)$ denotes the Hamming weight of some positive integer k .

4. System architecture

4.1. Hardware/software partitioning

In Fig. 1, the pyramid shows the hierarchy of operations in HECC. Each level is mapped onto a different hardware or software structure. At the lowest level there are GF(2⁸³) operators which are implemented in the hardware datapath. The combinations of these operators are designed using the microcode instructions. Each line of the algorithms shown in Tables 5 and 6 can be represented

by one of these microcode instructions. The next level is the divisor operations (doubling and addition) that are developed in software routines using the coprocessor’s microcode assembly instructions. This is the HW/SW partition boundary. On top of this level the scalar multiplication algorithm is implemented using the C code on the CPU.

4.2. HECC hardware architecture

Fig. 2 shows the block diagram of the hardware architecture. The coprocessor includes a datapath, local storage, and the top controller which will be explained in detail in the next section. There are four 8-bit ports that are used for communication between the microcontroller and the coprocessor. Two of them are for the input and output data and the other two are for coprocessor instructions and the address to access the local storage. Every data transfer to the local storage (RAM) is through the *input-word* and the *output-word* registers that are 84 bits wide (the word length of the operation in the coprocessor datapath).

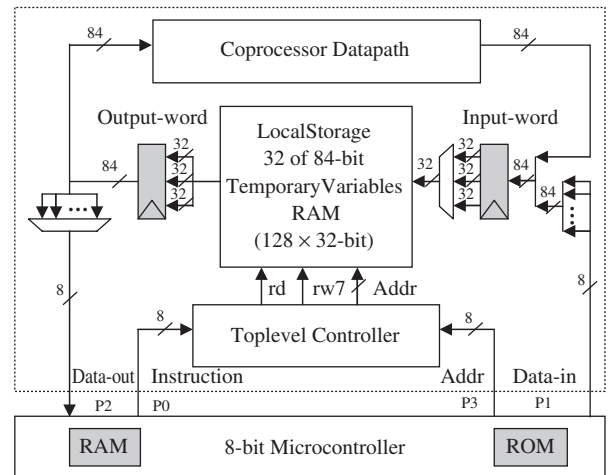


Fig. 2. Microcode coprocessor connected to 8-bit microcontroller.

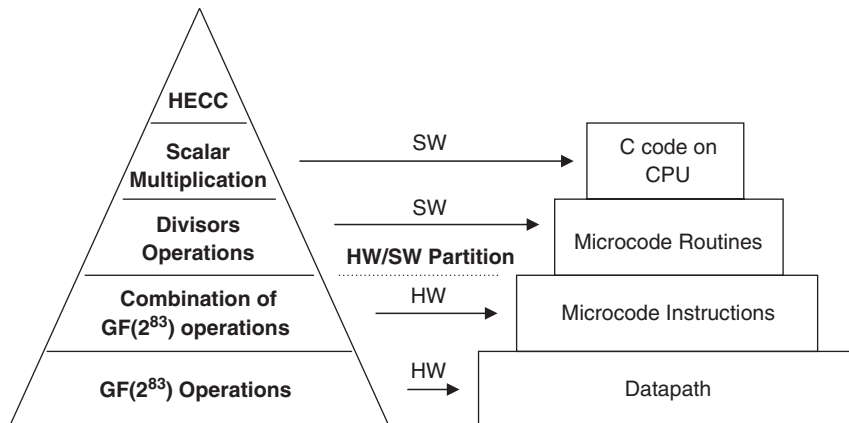


Fig. 1. HW/SW partitioning of the HECC.

4.3. Software architecture

The software is divided into two main sections. One is the routines of microcode instructions that are used to program the coprocessor. These routines implement the divisor’s operations (doubling, addition, and subtraction) based on the algorithms of Tables 5 and 6. The second section of the software is the C code which is finally compiled to microcontroller’s assembly code. The scalar multiplication algorithm based on the NAF algorithm is used to reduce the number of additions. This partition of the software is implemented in C and calls the routines of divisor’s operations described above. Two different microcontrollers are used in our implementations. One is 8051 which is an 8-bit microcontroller originally designed by Intel. Second microcontroller is the Atmel 8-bit AVR. In our architecture we have used the Dalton 8051 core from UC Riverside [14] and the AVRORA [15] design environment for 8051 and AVR implementations, respectively. In both case the microcontroller interfaces to the microcode coprocessor via IO ports.

5. Coprocessor architecture

5.1. Coprocessor datapath

Fig. 3 shows the coprocessor’s datapath that is designed based on the dual-multiplier/dual-adder in $GF(2^{83})$. The main reason for this implementation is that the divisor’s operations in Tables 5 and 6 are scheduled in such a way that two multiplications or additions can be performed concurrently in order to increase the overall performance. This means that the datapath has to be capable of performing every line of the schedules in Tables 5 and 6.

As shown before, the multiplier and the adder units of Fig. 3 are $GF(2^{83})$ operators. The bit-serial $GF(2^{83})$ multiplier is used that performs multiplication in 84 cycles. This is mainly for the reason of area compactness.

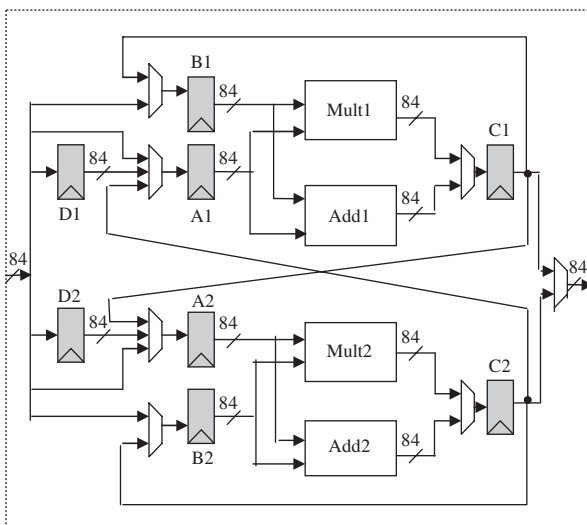


Fig. 3. Coprocessor’s datapath.

Moreover, the adder is a bit-parallel adder that can produce the output in a single clock cycle. It will be shown in the results section that because of the overhead of programming the microcode coprocessor from the main microcontroller, the bit-serial multiplier generates its output before the next instruction is ready to be processed by the coprocessor. In the case of 8051 microcontroller, this programming overhead is mainly because of the one over twelve clock division unit of the 8051 which make the coprocessor to perform 12 time faster than the 8051 microcontroller. Therefore, the processor will never wait for the bit-serial multiplier to finish its operation.

5.2. Coprocessor storage unit

The local storage unit consists of 128 memory locations of 32-bit width. Every four locations are used to store each temporary variable of the $GF(2^{83})$ field. Therefore, there are total of 32 memory locations that can store the elements of $GF(2^{83})$. The memory address bus is 7 bits wide to cover the 128 locations (32 variables) and the coprocessor controller asserts the required values for the memory read (rd) and write (wr) signal. Also notice that the input into and out of the local RAM has to go through the *input-word* and *output-word* registers.

6. Coprocessor instruction set

6.1. Coprocessor single instructions

These instructions are used to transfer data between the coprocessor and the microcontroller, load and store data to the RAM through the *input-word* and *output-word* registers, perform single operations using each of the adders and multipliers, and moving the content of the registers in the coprocessor datapath. Some examples of the single instructions and their definitions are shown in Table 1.

Table 1
Coprocessor single instructions

Instruction	Definition
Load-data-in	Loads 8 bits of data from Data-in port to the Input-word
Get-data-out	Returns 8 bits of data from Output-word to the Data-out port
Load-to-RAM	Loads the Input-word into the RAM from addr to addr + 2
Read-from-RAM	Returns content of RAM from addr to addr + 2 to Output-word
Do-mult1	Runs the first Multiplier (Mult1) $C1 = A1 * B1$
Do-add1	Runs the first Adder (Add1) $C1 = A1 + B1$
Outword-to-A1	Moves the content of output-word to A1
C1-to-inword	Moves the content of C1 to input-word
C1-to-B1	Moves the content of C1 to B1
D1-to-A1	Moves the content of D1 to A1

6.2. Coprocessor microcode instructions

The microcode instructions are the main instructions that are used to implement the divisor's addition and doubling algorithms. These instructions implement a combination of Galois Field additions and multiplications with multiple input operands. Table 2 lists some examples of these instructions and their definitions. Any line of the divisor's doubling or addition schedules can be implemented by one of these microcode instructions. It should be noticed that each of the microcode instructions are implemented using a sequence of other instructions. For example the *Two-MultAdd* instruction is represented by *Domult1* and *Domult2*, followed by *CITOB1* and *DITOA1*, followed by *C2TOB2* and *D2TOA2*, followed by *Doadd1* and *Doadd2* instructions.

7. Results

The proposed HW/SW co-design of the HECC system was implemented and co-simulated using GEZEL [16].

GEZEL is a design environment for the exploration of domain-specific coprocessor and multiprocessor micro architectures, which can provide cycle true HW/SW co-simulation with various embedded core instruction set simulators (ISS). In our application, we used the Dalton 8051 ISS and AVRORA simulator to perform cycle-accurate co-simulation for 8051 and AVR microcontrollers, respectively. The microcode coprocessor is designed in GEZEL hardware description language which is a finite state machine plus datapath (FSMD) system model. After timing and functional verification is performed, the GEZEL code was automatically converted to RTL VHDL and synthesized for FPGA. The detailed timings of different parts of the HECC co-design implementation are presented in Table 3. The delay is given in terms of number of cycles and msec at the 12 MHz clock frequency.

Table 4 compares the performance of the scalar multiplication of the proposed HW/SW co-design of the HECC system with related work. Our 83-bit HECC system takes 7.8 M cycles of the 8051 microcontroller which results in the total delay of 656 ms at a 12 MHz clock frequency. This

Table 2
Coprocessor microcode instructions

Instruction	Definition	
Multi-Mult	$C1 = (A1 * B1)$	$C2 = (A2 * B2)$
Add-Mult	$C1 = (A1 + B1)$	$C2 = (A2 * B2)$
TwoAdd-Mult	$C1 = (A1 + B1) * (A2 + B2)$	
TwoMult-Add	$C1 = (A1 * B1) * (A2 * B2)$	
MultiAdd-Mult	$C1 = (A1 * B1) + D1$	$C2 = (A2 * B2)$
Two-MultAdd	$C1 = (A1 * B1) + D1$	$C2 = (A2 * B2) + D2$

Table 3
HW/SW cryptosystem performance

HECC operations	8051 Performance [# clocks (Mcycles)]	AVR Performance [# clocks (Mcycles)]	8051 Performance [Delay (ms) At 12 MHz]	AVR Performance [Delay (ms) At 12 MHz]
I/O data transfer	0.089	0.010	7.4	0.88
Divisor's double	0.11	0.013	9.9	1.18
Divisor's addition	0.13	0.015	11.1	1.32
Coordinate conversion	0.20	0.023	17.2	2.05
Scalar multiplication	7.87	0.938	656	78

Table 4
Comparison with related work

Design	PKC	CPU	Frequency (MHz)	Delay (ms)	# Cycles (Mega)
[11]	163-bit ECC	AVR	4	115	
[10]	80-bit HECC	ARM7	80	374	29.9
[5]	83-bit HECC	ARM7	80	71.56	5.72
[6]	160-bit ECC	8051	12	4580	54.9
Software only	83-bit HECC	8051	12	149800	1800
Software only	83-bit HECC	AVR	12	8310	100
HW/SW co-design	83-bit HECC	8051	12	656	7.87
HW/SW co-design	83-bit HECC	AVR	12	78	0.938

Table 5
Formulae for parallelized doubling

Step	Calculations	# mult.
1	<i>Precomputation and resultant r:</i> $t_0 = Z^2$ $t_1 = U_1^2$ $r = U_0 \cdot Z$ $a = Z + V_1$	2M
2	<i>Compute k:</i> $k_1 = f_3 \cdot t_0 + t_1$ $b = V_1 \cdot a + t_0$ $k_0 = U_1 \cdot k_1 + Z \cdot b$	2M
3	<i>Compute s:</i> $t_2 = k_0 \cdot U_1$ $s_1 = k_0 \cdot Z$ $s_0 = k_1 \cdot r + t_2$	2M
4	<i>Compute l:</i> $t_0 = t_0 \cdot r$ $t_1 = s_1 \cdot k_0$ $r = t_0 \cdot s_1$ $t_3 = U_0 \cdot k_0$ $l_2 = s_1 \cdot t_2$ $l_0 = s_0 \cdot t_3$ $l_1 = (t_2 + t_3) \cdot (s_0 + s_1)$ $l_1 = l_1 + l_2 + l_0$	4M
5	<i>Compute U':</i> $U'_0 = s_0^2 + r$ $U'_1 = t_0^2$	1M
6	<i>Precomputation:</i> $a = s_0 \cdot s_1 + U'_1$; $s_1 = s_1^2$ $l_2 = l_2 + a$ $b = U'_0 + l_1$ $Z' = s_1 \cdot r$ $t_2 = r \cdot t_1$ $t_0 = U'_0 \cdot l_2 + l_0 \cdot s_1$ $t_1 = U'_1 \cdot l_2 + s_1 \cdot b$	4M
7	<i>Adjust:</i> $U'_1 = U'_1 \cdot r$ $U'_0 = U'_0 \cdot r$	1M
8	<i>Compute V':</i> $V'_0 = t_0 + t_2 \cdot V_0$ $V'_1 = t_1 + t_2 \cdot V_1 + Z'$	1M
Total		17M

implementation is more than 228 times faster than the pure software implementation of HECC on 8051 and it is 7 times faster than 160-bit ECC implementation on 8051 as reported by [6]. On AVR, our 83-bit HECC system takes 938 k cycles which is equivalent to the total delay of 78 ms at a 12 MHz clock frequency. This number is 106 times faster than pure software implementation on AVR. The main reason that AVR is much faster than 8051 even at the same clock frequency (12 MHz) is due to the clock division unit in 8051 microcontroller. Basically every instruction in 8051 takes 12 times more cycles because the internal clock of the 8051 is divided by 12. Compared to 80-bit HECC implementation of ARM7, the number of clock cycles is a better metric because ARM7 is clocked at 80 MHz. In terms of number of clock cycles, our design on 8051 is at the same order with [5] and around 4 times faster than [10]. On AVR, our HECC implementation is 6 and 58 times faster than [5] and [6], respectively (Tables 5 and 6).

8. Conclusion

A HW/SW co-design of a hyperelliptic curve cryptosystem was presented using a microcode crypto coprocessor

Table 6
Formulae for parallelized addition

Step	Calculations	# mult.
1	<i>Precomputation and resultant r:</i> $t_1 = U_{11} \cdot Z_2 + U_{21}$ $t_2 = U_{10} \cdot Z_2 + U_{20}$ $t_0 = U_{11} \cdot t_1 + t_2$ $a = t_1^2$ $r = t_0 \cdot t_2 + a \cdot U_{10}$	3M
2	<i>Compute almost s:</i> $t_4 = V_{10} \cdot Z_2 + V_{20}$ $t_5 = V_{11} \cdot Z_2 + V_{21}$ $w_2 = t_0 \cdot t_4$, $b = t_0 + t_1$ $w_3 = t_1 \cdot t_5$ $b = b \cdot (t_4 + t_5)$ $a = w_3 \cdot (1 + U_{11})$ $b = w_2 + b$, $s_1 = a + b$ $s_0 = w_2 + U_{10} \cdot w_3$	4M
3	<i>Precomputations:</i> $R = Z_2 \cdot r$ $s_3 = s_1 \cdot Z_2$ $\tilde{R} = R \cdot s_3$ $s_0 = s_0 \cdot Z_2$ $S = s_0 \cdot s_1$ $S_3 = s_3^2$ $\tilde{S} = s_3 \cdot s_1$ $\tilde{\tilde{S}} = s_0 \cdot s_3$ $\tilde{\tilde{R}} = \tilde{R} \cdot \tilde{S}$	5M
4	<i>Compute l:</i> $l_2 = \tilde{S} \cdot U_{21}$ $l_0 = S \cdot U_{20}$ $l_1 = (\tilde{\tilde{S}} + S) \cdot (U_{21} + U_{20})$, $l_1 = l_0 + l_1 + l_2$ $l_2 = l_2 + \tilde{\tilde{S}}$	2M
5	<i>Compute U':</i> $a = t_1 \cdot r$ $b = s_1^2$ $c = s_1 \cdot Z_2$ $b = b \cdot t_1$ $a = R \cdot (a + c)$ $b = b \cdot (t_1 + U_{21})$ $d = t_2 \cdot \tilde{S} + s_0^2$, $U'_0 = b + d + a$ $U'_1 = \tilde{S} \cdot t_1 + R^2$ $b = S_3 \cdot (U'_0 + l_1)$, $l_2 = l_2 + U'_1$ $t_4 = U'_0 \cdot l_2 + S_3 \cdot l_0$ $Z' = \tilde{R} \cdot S_3$ $t_5 = U'_1 \cdot l_2 + b$ $U'_1 = \tilde{R} \cdot U'_1$ $U'_0 = \tilde{R} \cdot U'_0$	9M
6	<i>Compute V':</i> $V'_0 = \tilde{\tilde{R}} \cdot \tilde{V}_{20}$, $V'_1 = \tilde{\tilde{R}} \cdot \tilde{V}_{21} + Z'$, $V'_0 = t_4 + V'_0$ $V'_1 = t_5 + V'_1$	1M
Total		24M

attached to the ports of the 8-bit microcontroller. The proposed crypto coprocessor is capable of performing a range of $GF(2^{83})$ operations. The divisor's addition and doubling operations are implemented using SW routines based on the coprocessor's microcode instructions. The scalar multiplication is developed in C and compiled into microcontroller assembly instructions. The total delay of 656 ms (7.8 Mcycles) and 78 ms (1 Mcycles) was achieved for the 83-bit HECC scalar multiplication at 12 MHz on 8051 and AVR microcontrollers, respectively.

Acknowledgement

This work is the result of cooperation between UCLA–EMSEC and the KUL–COSIC research groups.

The support of the Space and Naval Warfare Systems Center—San Diego under Contract no. N66001-02-1-8938, NSF, UC Micro, the Hertz Foundation (for David Hwang) and FWO projects (G.0141.03) and (G.0450.04) is gratefully acknowledged.

References

- [1] A. Menezes, P. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, 1997.
- [2] B. Byramjee, S. Duquesne. Classification of genus 2 curves over and optimization of their arithmetic, Cryptology ePrint Archive: Report 2004/107.
- [3] J. Pelzl, T. Wollinger, C. Paar, High performance arithmetic for hyperelliptic curve cryptosystems of genus two, in: Proceedings of ITCC, April 57, 2004, Las Vegas, Nevada, USA, 2004.
- [4] T. Lange, Formulae for arithmetic on genus 2 hyperelliptic curves, Applicable Algebra in Engineering, Communication and Computing 15 (5) (2005) 295–328.
- [5] J. Pelzl, T. Wollinger, C. Paar, Special Hyperelliptic Curve Cryptosystems of Genus Two: Efficient Arithmetic and Fast Implementation, Chapter in Embedded Cryptographic Hardware: Design and Security, Nova Science Publishers, 2004.
- [6] N. Gura, A. Patel, A. Wander, H. Eberle, S.C. Shantz, Comparing elliptic curve cryptography and RSA on 8 bit CPUs, in: M. Joye, J.J. Quisquater (Eds.), Proceedings of Sixth International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Lecture Notes in Computer Science, vol. 3156, 2004, pp. 119–132.
- [7] N. Boston, T. Clancy, Y. Liow, J. Webster. Genus two hyperelliptic curve coprocessor, in: B.S. Kaliski Jr., K. Ko, C. Paar, (Eds.), Proceedings of Fourth International Workshop on Cryptographic Hardware and Embedded Systems (CHES), Lecture Notes in Computer Science, vol. 2523, Springer, Berlin, 2002, pp. 400–414.
- [8] D. Cantor, Computing the Jacobian of a hyperelliptic curve, Math. Comput. 48 (1987) 95–101.
- [9] H. Kim, T. Wollinger, Y. Choi, K. Chung, C. Paar, Hyperelliptic curve coprocessors on a FPGA, in: Workshop on Information Security Applications WISA, Jeju Island, Korea, 2004.
- [10] S. Baktir, J. Pelzl, T. Wollinger, B. Sunar, C. Paar, Optimal tower fields for hyperelliptic curve cryptosystems, in: Proceedings of 38th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, USA, November 2004.
- [11] S. Kumar, C. Paar, Reconfigurable instruction set extension for enabling ECC on an 8 bit processor, in: Proceedings of International Conference on Field Programmable Logic and Applications (FPL), Lecture Notes in Computer Science, vol. 3203, Antwerp, Belgium, 2004.
- [12] N. Koblitz, Elliptic curve cryptosystem. Math. Comp., 48:203–209, 1987. 16. N. Koblitz. A family of Jacobians suitable for Discrete Log Cryptosystems, in: S. Goldwasser, (Ed.), Advances in Cryptology: Proceedings of CRYPTO'88, Lecture Notes in Computer Science, vol. 403, Springer, Berlin, 1988, pp. 94–99.
- [13] T. Itoh, S. Tsujii, Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$, Electron. Lett. 24 (6) (1988) 334–335.
- [14] Dalton 8051, <http://www.cs.ucr.edu/~dalton/8051/>
- [15] AVRORA, <http://compilers.cs.ucla.edu/avrora/>
- [16] P. Schaumont, I. Verbauwhede, Interactive cosimulation with partial evaluation, 2004 DATE 2004, pp. 642–647, February 2004.



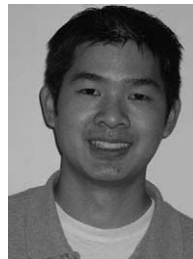
He is a student member of the IEEE.

Alireza Hodjat received his B.S degree in Electrical Engineering from the University of Tehran, Iran in 1999 and his M.S. degree in Electrical Engineering from the University of California, Los Angeles in 2002. He received his Ph.D. degree in Electrical Engineering from UCLA in 2005. His research interests include hardware/software co-design and Application Specific Instruction Set coprocessor architectures and VLSI implementations for secure embedded systems.



Currently, she is a postdoctoral researcher working with the COSIC research group at the Katholieke Universiteit Leuven, Belgium. Her research interests include Elliptic/Hyperelliptic Curve Cryptography (hardware implementations) and side-channel security.

Lejla Batina received the M.Sc. degree in Mathematics from the University of Zagreb, Croatia in 1995 and the MTD degree from the Technical University of Eindhoven, The Netherlands in 2001 (Mathematics for Industry—Postmasters study). She has held research positions in the Department of Mathematics at the Eindhoven University and the Institute for Experimental Mathematics in Essen, Germany. She received her Ph.D. degree from K.U. Leuven, in Leuven, Belgium in 2005.



He is a member of Phi Beta Kappa, Tau Beta Pi, and Eta Kappa Nu.

David D. Hwang received his M.S. and Ph.D. degrees in Electrical Engineering from UCLA in 2001 and 2005, respectively, as a graduate fellow of the Fannie and John Hertz Foundation. His academic research focused on VLSI architectures for secure embedded systems, cryptographic ASICs, and digital signal processing. He is currently with KeyEye Communications, where his research interests include VLSI DSP architectures, broadband digital communications, and high-speed ASIC design.



Her interests include circuits, processor architectures and design methodologies for real-time, embedded systems in application domains such as security, cryptography, digital signal processing and wireless applications. Prof. Verbauwhede was the general chair of the IEEE International Symposium on Low Power Electronic Devices (ISLPED) in 2003. She is or was a member of several program committees, including DAC, ISSCC, DATE, CHES, ICASSP, SIPS, ASAP. She is the design community chair on the 42nd and 43rd DAC executive community. She is a senior member of IEEE.

Ingrid Verbauwhede received the Electrical Engineering Degree in 1984 and the Ph.D. Degree in applied sciences from the K.U.Leuven, in Leuven, Belgium in 1991. She was a lecturer and visiting research engineer at UC Berkeley from 1992 to 1994. From 1994 to 1998 she was a principal engineer first with TCSI and then with Atmel in Berkeley, CA. She joined UCLA in 1998 as an associate professor and the K.U.Leuven in 2003.