On the Implementation of Unified Arithmetic on Binary Huff Curves

Santosh Ghosh[1,⋆], Amit Kumar[2,⋆⋆], Amitabh Das[3], and Ingrid Verbauwhede[3]

[1] Security Center of Excellence (SeCoE)
Intel Corporation, 2111 NE 25th Avenue, Hillsboro, OR 97124, United States
firstname.lastname@esat.kuleuven.be
[2] Department of Electrical Engineering
Indian Institute of Technology Kharagpur, WB 721302, India
amitrohilla.iitkgp@hotmail.com
[3] COSIC-SCD/ESAT
KU Leuven & iMinds, Kasteelpark Arenberg 10, Bus 2446, Heverlee 3001, Belgium
santosh.ghosh@intel.com

Abstract. Unified formula for computing elliptic curve point addition and doubling are considered to be resistant against simple power-analysis attack. A new elliptic curve formula known as unified binary Huff curve in this regard has appeared into the literature in 2011. This paper is devoted to analyzing the applicability of this elliptic curve in practice. Our paper has two contributions. We provide an efficient implementation of the unified Huff formula in projective coordinates on FPGA. Secondly, we point out its side-channel vulnerability and show the results of an actual attack. It is claimed that the formula is unified and there will be no power consumption difference when computing point addition and point doubling operations, observable with simple power analysis (SPA). In this paper, we contradict their claim showing actual SPA results on a FPGA platform and propose a modified arithmetic and its suitable implementation technique to overcome the vulnerability.

Keywords: Elliptic curves, Binary fields, Side-channel, FPGA, Karatsuba multiplier, Power analysis, SPA.

1 Introduction

SIDE-CHANNEL ATTACKS [15] are a major threat in present day embedded security era, irrespective of whether the underlying cryptographic algorithm is based on public key or private key. In order to protect an elliptic curve algorithm against simple power-analysis attacks, there are three basic classes of countermeasures: *(i) Always double-and-add*, *(ii) Atomic execution*, and *(iii) Unified point addition*. The first one is too costly as the simple double-and-add algorithm executes a point addition only if the secret scalar bit (d_i) is one. In this

⋆ Part of the work has been performed when Santosh Ghosh was a Postdoctoral Fellow at COSIC, KU Leuven.

⋆⋆ Amit Kumar was a visiting scholar at COSIC, KU Leuven during this work.

G. Bertoni and J.-S. Coron (Eds.): CHES 2013, LNCS 8086, pp. 349–364, 2013.

countermeasure, a dummy point addition is executed if $d_i = 0$. However, there is a regular version of it due to Montgomery [13]. The second one executes point addition and point doubling by executing several atomic units of finite field operations, where a unit is formed with four operations in sequence: addition, multiplication, negation, and addition. The atomic execution helps to protect against side-channel attacks with several dummy finite field operations. The dummy operations in the first two countermeasures can however be targeted by a C safe-error attack [21]. The unified formula is a good option to protect elliptic curves against side-channel attacks. The development of such a formula is very difficult in practice, and only a few are available till today.

Unified formula for computing point addition and point doubling on an elliptic curve was introduced in 2001 [4,12]. Walter [20] observed a vulnerability of Brier and Joye's unified formula [4] with respect to the irregularity of the implementation of finite field operations. Subsequently, unified point addition formula in affine form [5] and respective projective form [18] were proposed. The latter one also reinforces Walter's [20] observations with timing analysis during software execution of unified formula. Thereafter, unified formula on Edwards curve was proposed in 2007 [2]. In CT-RSA 2011, unified binary Huff formula was proposed by Devigne and Joye [8], which outperforms other unified formula.

In this paper, we demonstrate that the unified binary Huff curve is not actually secure against side-channel attacks. Even though both point operations are executed by the same sequence of finite field operations, due to processing of different coordinates, they demand different amounts of power. This paper pinpoints to the fact that the point doubling with unified Huff formula produces zero output in some intermediate finite field operations, which are non-zero in point addition. These zero (non-zero) values for point doubling (point addition) are further used as multiplicands in the unified formula. Results of the multiplications are also zero (non-zero). The power consumption of the multiplier circuit having zero and non-zero data are significantly different and they are visually observable through their power consumption graphs[1]. We show the actual power consumption graphs of those operations on a SASEBO-G board [19] which proves our claim and successfully demonstrates the vulnerability of the unified huff formula against simple power analysis. Apart from the side-channel resistance analysis, this paper also provides an efficient architecture and an optimal countermeasure of binary Huff curve.

We start with a brief overview of binary Huff curve in § 2. We show the side-channel vulnerability of the unified binary Huff curve formula in § 3. The same section also demonstrates the actual power analysis on a SASEBO-G FPGA board. A suitable countermeasure is proposed in § 4 and it is validated by the actual SPA on the SASEBO-G board. The detailed architecture for implementing elliptic curve scalar multiplication based on our proposed SPA-resistant binary Huff curve addition formula is described in § 5. Finally, we conclude in § 6.

[1] We use the terms *plot, graph,* and *trace* with same meaning which represent a 2D plot of the variable (power consumption) with respect to time.

2 Binary Huff Curve

In the mid-twentieth century, while studying a Diophantine problem, Huff introduced a new model of elliptic curves [10]. After a long gap, the Huff model was revisited in 2010 [14], which fully described and formulated the case of odd characteristic fields and provided an outline for binary field. Thereafter, in 2011, the formal construction of a Huff model for binary field was developed by Devigne and Joye [8]. This construction instead of providing general point addition formula takes care of side-channel attacks and provides the unified point addition and point doubling formula in binary field. Here we provide a brief description that may help in understanding the contributions of the current paper.

Definition 1 ([14]). *A generalized binary Huff curve is the set of projective points* $(X : Y : Z) \in \mathbb{P}^2(\mathbb{F}_{2^m})$ *satisfying the equation*

$$E_{/F_{2^m}} : aX(Y^2 + fYZ + Z^2) = bY(X^2 + fXZ + Z^2), \tag{1}$$

where $a, b, f \in \mathbb{F}_{2^m}^*$ *and* $a \neq b$.

There are three points at infinity satisfying the curve equation, namely $(a : b : 0)$, $(1 : 0 : 0)$, and $(0 : 1 : 0)$. For $P = (X_1 : Y_1 : Z_1)$ and $Q = (X_2 : Y_2 : Z_2)$, we get $P + Q = (X_3 : Y_3 : Z_3)$ with unified point addition/doubling formula [8]:

$$\begin{cases} X_3 = (Z_1Z_2 + Y_1Y_2)\big((X_1Z_2 + X_2Z_1)(Z_1^2Z_2^2 + X_1X_2Y_1Y_2) + \\ \qquad\qquad \alpha X_1X_2Z_1Z_2(Z_1Z_2 + Y_1Y_2)\big) \\ Y_3 = (Z_1Z_2 + X_1X_2)\big((Y_1Z_2 + Y_2Z_1)(Z_1^2Z_2^2 + X_1X_2Y_1Y_2) + \\ \qquad\qquad \beta Y_1Y_2Z_1Z_2(Z_1Z_2 + X_1X_2)\big) \\ Z_3 = (Z_1Z_2 + X_1X_2)(Z_1Z_2 + Y_1Y_2)(Z_1^2Z_2^2 + X_1X_2Y_1Y_2), \end{cases} \tag{2}$$

where $\alpha = \frac{a+b}{b}$ and $\beta = \frac{a+b}{a}$.

The unified formula provided in Eq. (2) can be evaluated as in [8]

$$\begin{array}{l} m_1 = X_1X_2, \quad m_2 = Y_1Y_2, \quad m_3 = Z_1Z_2, \\ m_4 = (X_1 + Z_1)(X_2 + Z_2) + m_1 + m_3, \quad m_5 = (Y_1 + Z_1)(Y_2 + Z_2) + m_2 + m_3, \\ m_6 = m_1m_3, \quad m_7 = m_2m_3, \quad m_8 = m_1m_2 + m_3^2, \quad m_9 = m_6(m_2 + m_3)^2, \\ m_{10} = m_7(m_1 + m_3)^2, \quad m_{11} = m_8(m_2 + m_3), \quad m_{12} = m_8(m_1 + m_3), \\ X_3 = m_4m_{11} + \alpha m_9, \quad Y_3 = m_5m_{12} + \beta m_{10}, \quad Z_3 = m_{11}(m_1 + m_3). \end{array}$$

This unified point addition for binary Huff curve consists of 17 field multiplications. If we assume that there is only one multiplier in the datapath of the point addition block then we can execute the above operations in 17 steps each of which consists of one binary field multiplication. The detailed RTL description is provided in Table 4 in the Appendix. It uses only six temporary registers. Based on this RTL definition the double-and-add algorithm (Algorithm 1 in Appendix) for elliptic curve point multiplication has been implemented on a SASEBO-G FPGA and a power analysis is performed.

3 SPA on Binary Huff Curve

Simple-power analysis or SPA on elliptic curve is based on the observations of power consumption of the cryptoprocessor during the executions of point addition and point doubling. When the operations of point addition and point doubling make use of different formula, they may produce different power traces revealing the secret value of scalar d in the computation of $Q = [d]P$. A potential approach that counteracts such vulnerability tries to unify the addition formula. Huff curves are equipped with such a unified point addition formula which results in the SPA resistant property. However, sometimes formula based on theoretical assumptions are vulnerable at their actual implementations. The practical results demonstrated in this section proves that the unified binary huff curve formula [8] is not actually secure against SPA attack.

3.1 Pinpointing the SPA Vulnerability

The authors in [8] claim that the unified formula for computing point addition and doubling on Binary Huff curve is secure against side-channel attacks, especially against SPA. However, with a close observation of Eq. (2), we find out that there are behavioral differences during the computations of $P + Q,\ P \neq Q$ and $P + P$. Let us consider the computation of $X_3 = (Z_1Z_2 + Y_1Y_2)((X_1Z_2 + X_2Z_1)(Z_1^2Z_2^2 + X_1X_2Y_1Y_2) + \alpha X_1X_2Z_1Z_2(Z_1Z_2 + Y_1Y_2))$. In this formula it is pointed out that the value of $(X_1Z_2 + X_2Z_1)$ for the $P + P$ computation is zero in $\mathbb{F}_{2^m}$ whereas it is in general non-zero for a $P + Q,\ P \neq Q$ computation. This zero (non-zero) value is further multiplied with $(Z_1^2Z_2^2 + X_1X_2Y_1Y_2)$, which produces a zero (non-zero) product for point doubling (point addition).

Similarly, in $Y_3 = (Z_1Z_2 + X_1X_2)((Y_1Z_2 + Y_2Z_1)(Z_1^2Z_2^2 + X_1X_2Y_1Y_2) + \beta Y_1Y_2Z_1Z_2(Z_1Z_2 + X_1X_2))$, the value of $(Y_1Z_2 + Y_2Z_1)$ is zero for point doubling and non-zero for point addition. To perform SPA on unified binary Huff curve, the respective formula provided in Eq. (2) has been implemented on an FPGA device. Fig. 1 shows ModelSIM simulation results for computing Eq. (2) as described in Table 4. It only displays respective values of the multiplicands and multiplication results. The sign '0' and '>' indicate zero and non-zero values respectively. It could be observed from the left half of the figure that the multiplicand $a2$ goes to zero twice during a point doubling, once for $(X_1Z_2 + X_2Z_1)$ and once for $(Y_1Z_2 + Y_2Z_1)$. However, it never goes to zero during a point addition, which is displayed at the right half of the figure. Processing of zero and non-zero values in the datapath consume different amounts of power which can be observed from their respective power graphs. This can help to break the unified binary Huff curve using simple power analysis. In the following section, we show the actual SPA results on FPGA platform.

3.2 Actual Power Analysis Using SASEBO-G Board

The Side-channel Attack Standard Evaluation Board (SASEBO-G) [19] is an FPGA board especially designed to develop standard evaluation schemes to secure cryptographic modules against physical attacks. The SASEBO-G version

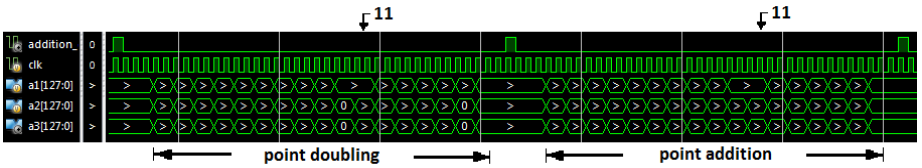

Fig. 1. Simulation of unified Huff curve point addition

board incorporates a Xilinx FPGA consisting of two Virtex-II pro devices. We implement the elliptic curve point multiplication (ECSM) based on the unified binary Huff curve addition formula (Eq. (2)) on the xc2vp30-fg676-5 device of the SASEBO-G board and perform power analysis. Fig. 2 shows a power trace during the execution of $[d]P$[2] on a binary Huff curve using unified formula. The power consumption for executing current unified Huff formula are mostly due to the execution of 17 finite field multiplications as shown in Table 4. Therefore, following observations can be made from this power trace.

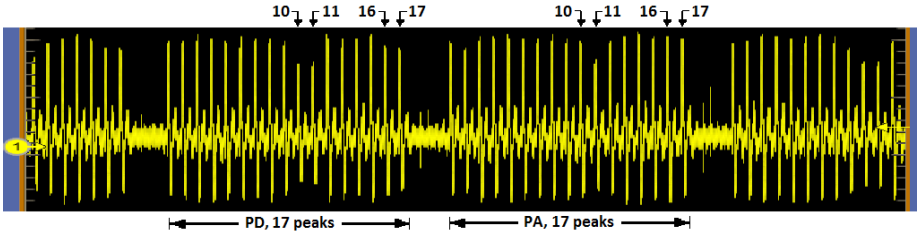

Fig. 2. Power consumption for computing [d]P with unified Huff formula

- The power trace during a point addition (PA) or a point doubling (PD) operation consists of 17 peaks for executing multiplications.
- The power consumption peak at the 11-th multiplication cycle is lower than other peaks, as during this cycle the multiplicand $a1$ remains unchanged from its previous value (see Fig. 1).
- The peak at the 10-th multiplication cycle is also lower for some point addition/doubling executions. They are due to values of the second multiplicand $a2$ which is zero during the processing of a point doubling operation (see Table 4 and Fig. 1).
- The peaks at 16-th and 17-th multiplication cycles during the execution of point doublings are also lower than that during point additions. This is because of the zero at the second multiplicand during 16-th multiplication (see Fig.1). The power consumption peaks are low in these two consecutive multiplication cycles due to transitions of $a2$ from non-zero to zero and again back to non-zero.

[2] We use $[d]P$ to represent elliptic curve point multiplication or ECSM where d is an integer and P is a point on the curve.

Based on the above observations, point addition and point doubling are easily distinguished from their power traces. Figures 3 and 4 show two compact views of power consumptions during a $[d]P$ execution. Through simple power analysis the secret scalar bits are easily guessed as shown in these figures. These results prove that the unified binary Huff curve [8] is vulnerable to SPA. We expect that similar conclusions can be made for software implementations of unified Huff curve on micro-controllers as our FPGA implementation also consists of only one datapath.

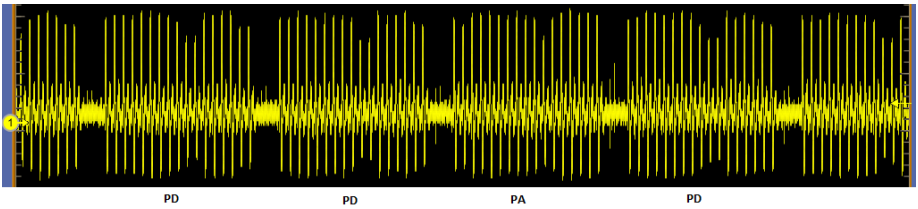

Fig. 3. SPA vulnerability of unified binary Huff curve

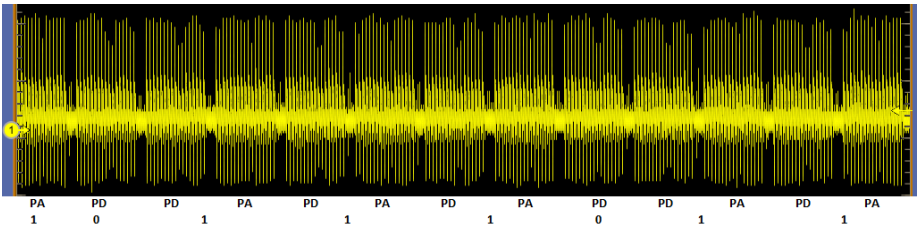

Fig. 4. SPA vulnerability : more compact view and leak to the key

4 Proposed SPA Countermeasure

As pointed out in Section 3.1, the main drawback of the original unified binary Huff formula is $X_1Z_2 + X_2Z_1$ and $Y_1Z_2 + Y_2Z_1$ computations. During point doubling $(P+P)$, resultant values of these two are zero, whereas they are nonzero for point addition $(P + Q,\ P \neq Q)$. Hence it could be implicitly inferred that elimination of these two sub-operations is sufficient to overcome the vulnerabilities to SPA.

4.1 Unified Huff Curve Arithmetic

Following is the proposed SPA resistant unified point addition technique on binary Huff curve. More specifically, we propose following arithmetic for executing Eq. (2) to overcome its vulnerability against SPA attack.

$$
\begin{aligned}
&m_1 = X_1X_2, \quad m_2 = Y_1Y_2, \quad m_3 = Z_1Z_2,\\
&m_4 = (X_1 + Z_1)(X_2 + Z_2), \quad m_5 = (Y_1 + Z_1)(Y_2 + Z_2),\\
&m_6 = m_1m_3, \quad m_7 = m_2m_3, \quad m_8 = m_1m_2 + m_3^2,\\
[t]\ &m_9 = m_6(m_2 + m_3)^2, \quad m_{10} = m_7(m_1 + m_3)^2, \quad m_{11} = m_8(m_2 + m_3),\\
&Z_3 = m_{11}(m_1 + m_3),\\
&X_3 = \alpha m_9 + m_4m_{11} + Z_3,\\
&Y_3 = \beta m_{10} + m_5m_8(m_1 + m_3) + Z_3.
\end{aligned}
$$

The cost of the above operations is $15M + 2D \approx 17M$, which is exactly the same as with the original one $(15M + 2D)$. In order to ensure the security of the proposed unified arithmetic, it is also implemented on the same FPGA which computes one unified point addition in 17 clock cycles. The architecture is sketched and described in the next section.

4.2 Additional Implementation Guidelines and Security Analysis

Based on the data dependency and available resources the proposed unified arithmetic can be implemented in several ways. Let us take a sample implementation of this arithmetic on the same set of resources (one multiplier) as used in the previous SPA experiment. Figures 5 and 6 show the corresponding simulation and SPA results, which provides another strange twist! This implementation is also vulnerable against SPA.

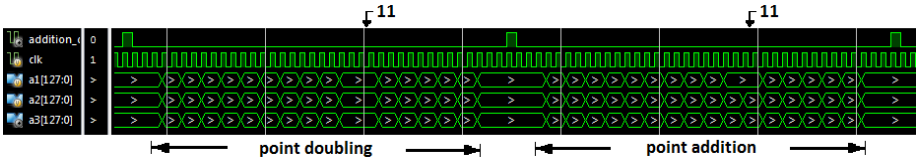

Fig. 5. Simulation result of an unsafe implementation of proposed arithmetic

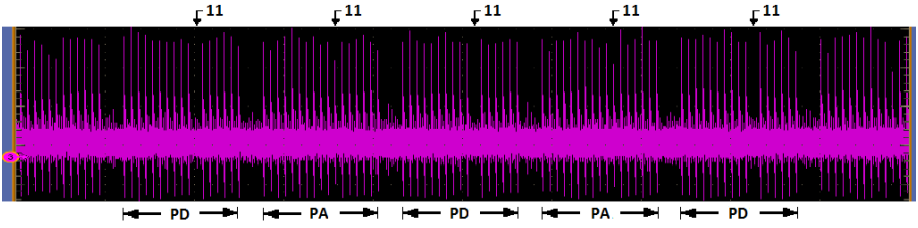

Fig. 6. SPA vulnerability of an unsafe implementation of proposed arithmetic

Let us see the cause of its vulnerability. It can be seen from the simulation result shown in Fig. 5 that the operands at 10-th and 11-th multiplication cycles are same for PD but one of them are different for PA. Therefore, the 11-th

multiplication for PD does not consume any power whereas it consumes power for PA (see Fig. 6). This happens due to the following features adopted in this implementation.

1. It schedules the multiplications $m_{11}(m_1+m_3)$ and m_4m_{11} at 10-th and 11-th cycles.
2. It chooses m_{11} as operand a for both multiplications.
3. It chooses m_1+m_3 and m_4 as operand b, respectively.

In case of PD the value of $m_4 := (X_1+Z_1)(X_2+Z_2)$ and the value of $m_1+m_3 := X_1X_2+Z_1Z_2$ are same but they are different in PA. This makes the difference in power consumptions and makes the implementation vulnerable to SPA. This result exposes the demand of security awareness on the implementation engineers.

In order to achieve an actual secure unified Huff curve hardware we suggest the implementation and scheduling of operations as shown in Table 5. We demonstrate the results of SPA that has been performed on SASEBO-G board on our proposed unified arithmetic and its scheduling technique. Figure 7 shows the simulation dataflow inside the multiplier with our proposed countermeasure. Contrary to the previous simulation result shown in Fig. 1, in this implementation, the intermediate result used as an operand in the multiplier never becomes zero. In other words, the multiplier never produce a zero for which the power consumption is distinguishable from other non-zero results.

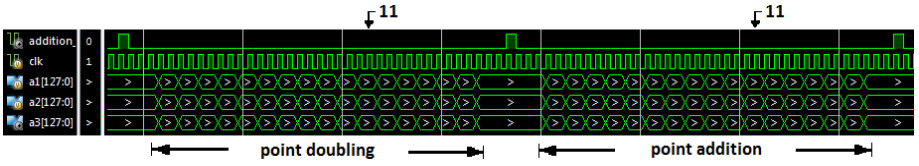

Fig. 7. Simulation of the modified unified Huff curve point addition arithmetic

The power analysis as described in Section 3.2 has been repeated for this new implementation and the results are shown in Figures 8, 9, and 10. There are 18 peaks for computing 18 multiplications. However, no observable power consumption difference has been found for computing point doubling and point addition. These experimental results ascertain the security of the proposed countermeasure against SPA. Therefore, the weakness of original Unified Binary Huff curve point addition formula is overcome by the proposed computation technique.

5 Architectural Description

Point multiplication, $[d]P = P + P + \cdots (d-1\ times)$, $d \in \mathbb{Z}^*$ is the main operation in elliptic curve cryptography. We develop the architecture for 256-bit binary field and it is also scaled to 128-bit and NIST recommended 233-bit binary fields. For 256-bit, the binary field $\mathbb{F}_{2^{256}}$ is defined with the irreducible polynomial $f(x) = x^{256} + x^{10} + x^5 + x^2 + 1$.

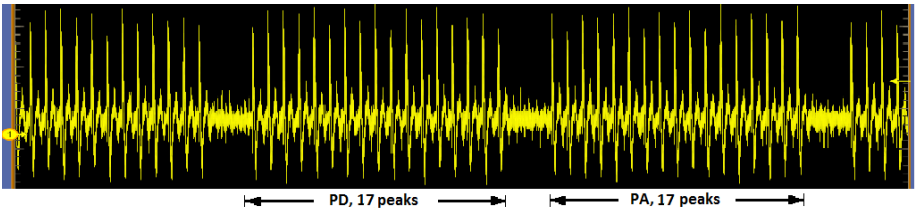

Fig. 8. Power consumption for computing [d]P with modified unified Huff arithmetic

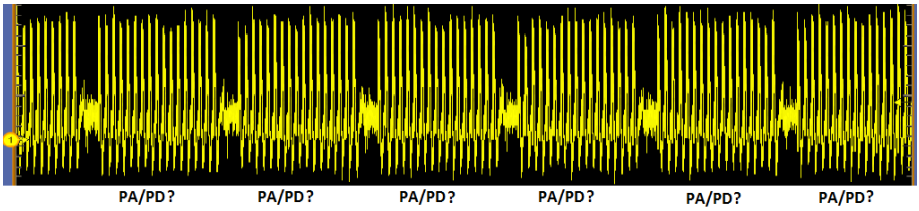

Fig. 9. SPA result of proposed unified binary Huff curve arithmetic

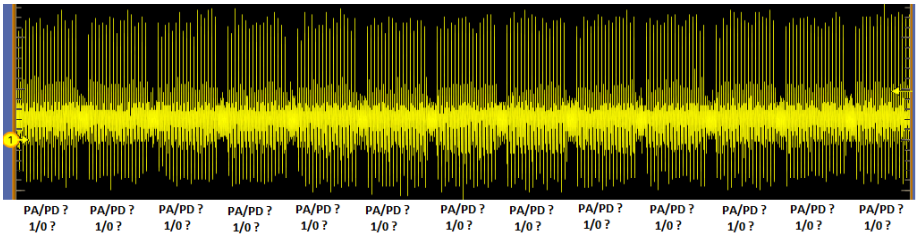

Fig. 10. SPA result in compact view on proposed unified binary Huff curve arithmetic

5.1 Input/Output

In this finite field, point coordinates $(X : Y : Z)$ and scalar d are all 256-bit long. Hence, total input pins required will be at least $256 \times 4 = 1024$ and total number of output pins required will be $256 \times 3 = 768$. So, a total of $1024 + 768 = 1792$ I/O pins are needed. Because of the limitations of the I/O pins on the available FPGA, we send the input parameters through a 32-bit port on FPGA. Hence, to input a 256-bit number, 8 clock cycles are required. Similarly, the output is displayed on a 32-bit port and 24 clock cycles are required to display three coordinates of the resultant point of a $Q = [d]P$ computation. Parameters for the next $[d]P$ operation could be taken in parallel as the design consists of different input and output ports.

Figure 11 depicts the top level architecture of the proposed cryptoprocessor for unified binary huff curve. Input parameters are fed through a 32-bit port and stored in 256-bit shift registers. There is a control bus $cntrl[3 : 0]$ which constitute the selector pins of a decoder and basically determines which of the four input parameters will be fed to the FPGA currently. There is an *act* signal which will enable the circuit to perform a scalar multiplication on the given

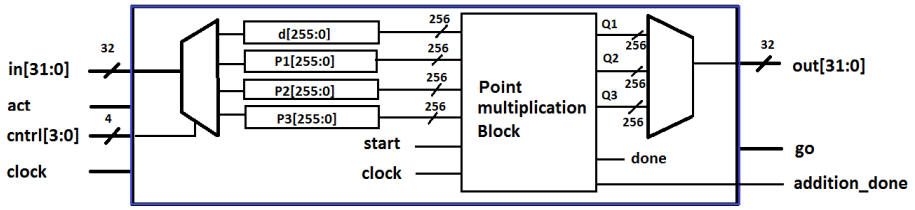

Fig. 11. Architecture of the top module

input scalar and the point. The integer d, and $x,\ y,\ z$ coordinates of the input point P are sent to the *point multiplication block* for computation. The result of the block goes to a multiplexer and the output is displayed on a 32-bit output port. The control pins of the multiplexer will select x, y and z coordinates one by one. Each coordinate will take 8 clock cycles to be displayed, making a total of 24 cycles. Signal *go* will be displayed during the display of the output and a signal *addition_done* signifies the completion of one point addition/doubling operation.

5.2 Datapath for Binary Huff Curve

Figure 12 shows the architecture of the Huff curve point multiplication block which executes left-to-right binary algorithm (Shown in Algorithm 1 in the Appendix). As the point addition and doubling are performed by unified formula, this algorithm can defend against simple power-analysis attacks. The Q registers are initialized with input base point P[3]. There is a 9-bit counter i which counts from $m-2$ down to zero. At every iteration this counter helps to select the corresponding bit of the scalar d. There are two intermediate signals $flag_1$ and $flag_2$ which tell us about the on-going point operation of either addition $(P+Q)$ or doubling $(Q+Q)$. If point doubling operation is going on, $flag_1$ will be enabled and $flag_2$ will be disabled. If point addition operation is going on, $flag_2$ will be enabled and $flag_1$ will be disabled. So, at any point of time during the whole process, one out of these two flags will be high.

There are two sets of input points which are fed to the Point Addition Block. One of the two inputs is Q, which is fed back from the output. Second input is coming from a $2:1$ multiplexer which will send P during point addition operation and Q during point doubling operation. The selector pins of the multiplexer results from a control circuitry made of $flag_1$, $flag_2$ and $d[i]$. It should be noted that the point addition and doubling operations are performed using the same block implemented using Unified Addition Formula ($Unif\_Add$ as described in Table 5). The start pin of the Point Addition Block results from a control circuitry made of $flag_1$, $flag_2$ and *addition_done* signals. After the completion of one point operation, *addition_done* will be enabled for one clock cycle during

[3] In this figure, registers and data buses $P_i,\ Q_i,\ 1 \leq i \leq 3$ represent the value of $x,\ y,\ z$ coordinates of the points P and Q, respectively.

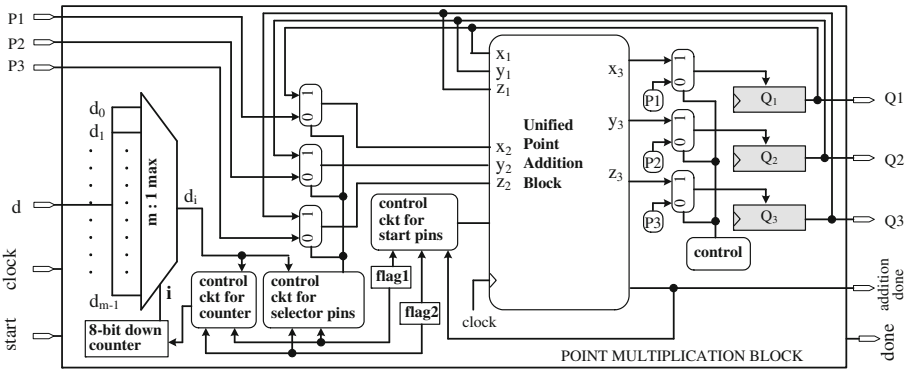

Fig. 12. Architecture of Huff curve point multiplication

which the Q registers are updated by the new intermediate result coming out from the Unified Point Addition Block. Finally, once the whole process is complete (at the end of the iteration when $i = 0$), the *done* signal will be enabled which enables the top level input-output circuitry.

Execution of SPA-Resistant Unified Point Operations. The proposed SPA-resistant Huff curve point addition formula is executed with the help of seven temporary registers. The detailed RTL description is provided in Table 5 in the Appendix. The number of registers used in the whole design is optimized through careful data flow analysis of the algorithm. The life time graph of the registers is depicted in Fig. 13. In the graph, the changes of a line style in a lifeline indicates the register is reassigned. The line beyond clock cycle 19 indicates that the value of that respective register is used in future. It is mainly used for Q registers, for which the resultant value of the current point addition is used as the input for the next addition.

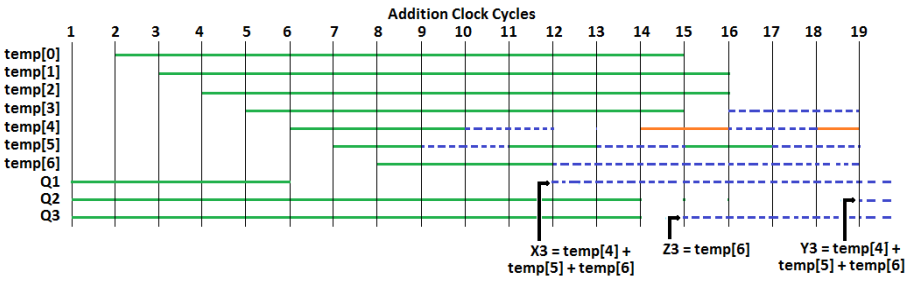

Fig. 13. Life time diagram of registers

The proposed design performs one multiplication in one clock cycle. Thus, next multiplicands and current multiplication result are stored in the same clock

transition. The multiplicands are sent to the combinatorial circuit of a Karatsuba Multiplier synchronously at the same clock when the result of the previous multiplication is stored in temporary registers. Hence, 17 multiplications (Ref Table 5) are completed in 17 clock cycles. Apart from that, one clock cycle is needed to start the operation, one is needed to store the result of current point addition/doubling, and one is needed to reset all the signals and flip flops. Hence, a total of 20 clock cycles are required to perform one point addition/doubling operation in our proposed SPA-resistant architecture.

Point Addition Block consists of a single multiplier sub-block where the input multiplicands are fed through a multiplexer. The inputs to the multiplexer are a bank of registers and the output signals of some combinatorial circuits which are used to perform addition and squaring operations. The primary inputs that represent the coordinates of two points are coming from the registers placed in the point multiplication block (Fig. 12). Both the squaring and multiplication operations are followed by the reduction sub-block. The selector pins of the multiplexer result from the control circuitry of the 20-bit register flag and the start pin. The outputs of the multiplier block after reduction are stored in one of the seven temporary registers as defined in our RTL table (Table 5). The coordinates of the resultant point are stored in the m-bit X_3, Y_3, and Z_3 registers and sent to the Point Multiplication Block with a signal *addition_done* indicating that a point addition/doubling operation is complete. The field multiplication unit is based on the hybrid Karatsuba multiplier as described in [17]. For a 256-bit field, we use simple Karatsuba decomposition and accumulation upto the multiplicand size of 32-bit and general Karatsuba for 16-bit multiplication.

5.3 Area and Time Results

The architecture is described in Verilog (HDL) and synthesized by Xilinx ISE tool to generate the FPGA configuration file. The area and timing results for three different field sizes are depicted in Table 1 and Table 2.

Table 1. Area and timing results of a **Unified Huff Addition** on FPGA

Device	**128 − bit**			**233 − bit**			**256 − bit**		
	Slice	**Clock [MHz]**	**Time [μs]**	**Slice**	**Clock [MHz]**	**Time [μs]**	**Slice**	**Clock [MHz]**	**Time [μs]**
Virtex-2Pro	7,270	110	0.19	16,214	109	0.19	19,256	88	0.24
Virtex-4	7,627	145	0.15	16,661	167	0.13	19,242	122	0.17
Virtex-6	3,027	180	0.12	8,091	172	0.12	8,239	152	0.14
Virtex-7	2,363	190	0.11	6,503	180	0.12	7,312	164	0.13

The performance comparison with existing recent results are shown in Table 3. Compared to the only implementation of unified binary Huff curve in [7], our design provides 75% performance improvement on a Virtex-4 FPGA compared to the only existing binary Huff curve architecture available in the literature.

Table 2. Area and timing results of **scalar multiplication** on FPGA

Device	128 − bit			233 − bit			256 − bit		
	Slice	Clock [MHz]	Time [μs]	Slice	Clock [MHz]	Time [μs]	Slice	Clock [MHz]	Time [μs]
Virtex-2Pro	8,345	110	37	19,043	110	67	21,423	98	82
Virtex-4	8,713	138	29	19,352	134	55	21,325	103	78
Virtex-6	3,924	182	22	7,150	172	43	11,083	146	55
Virtex-7	3,432	195	21	6,032	183	40	9,115	162	49

It is claimed by the authors in [8] that the unified binary Huff curve formula is faster than the unified formula on Edwards curve [2], which costs 18M+7D (or 21M+4D). This is proved by the design in [7] as well as in our design, which is ~ 3 times faster than the implementation of unified Edwards curve [6]. Another implementation of binary Edwards curves over $\mathbb{F}_{2^{163}}$ using Gaussian normal basis has been presented in [1], which computes a [d]P operation in $23.3\mu s$.

Table 3. Performance of the proposed implementation compared to others

Work	Platform	Field [m]	Slices Count	Clock [MHz]	Latency [μs]	Area × Latency ×[10^5]
Ours	$XC4V140$	233	19,352	134	55	10.6
Unified Edwards [6]	$XC4V140$	233	21,816	50	170	37.1
Unified Huff [7]	$XC4V140$	233	20,437	81	73	14.9

In general, an n-bit point multiplication based on the proposed Unified Huff curve arithmetic costs 25.5n M. This is not a cheap solution of side-channel attacks compared to existing other solutions. It is little bit slower than double-and-add always using Lopez-Dahab which costs 19n M (5M for one PD and 14M for one PA). The same is much slower than the Montgomery ladder, based on Lopez-Dahab fast point multiplication [16] trick which costs only 6n M. However, it should be noted that providing side-channel security is not the main goal of a Huff curve; rather, it provides a complete addition formula for all subgroups on an elliptic curve − even in a subgroup that does not contain the points at infinity. Due to this property the Huff curve is secure against exceptional procedure attacks [11] and batch computing [3]. In this respect, the Huff curve is one step ahead compared to its competitors Edwards [2] and Generalized Hessian curves [9] − on both of which the point addition is complete only on some specific subgroups.

6 Conclusion

Through close observations, we have pin-pointed a severe weakness of the unified addition formula of binary Huff curve against simple power-analysis attacks. The actual power analysis has been performed using SASEBO-G board. It has

been successfully demonstrated that the unified binary Huff curve is vulnerable against SPA. A suitable countermeasure has been also proposed and its robustness against SPA is demonstrated. The final design with SPA protection has been projected as the best performing unified elliptic curve implementation.

Acknowledgments. This work was supported in part by the Research Council KU Leuven: GOATENSE (GOA/11/007), by the Flemish iMinds projects. In addition, this work is supported in part by the Flemish Government, FWO G.0550.12N, by the Hercules Foundation AKUL/11/19. Santosh Ghosh was a beneficiary of a mobility grant from the Belgian Federal Science Policy Office co-funded by the Marie Curie Actions from the European Commission. Amitabh Das was initially funded by the Erasmus Mundus External Cooperation Window Lot 15 (EMECW15) when part of the work was performed. The authors are thankful to Marc Joye and Junfeng Fan for their valuable suggestions to perform this work. The authors are also thankful to the Anonymous Reviewer for useful comments in improving the paper.

References

1. Azarderakhsh, R., Reyhani-Masoleh, A.: Efficient FPGA Implementations of Point Multiplication on Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis. IEEE Trans. on VLSI Systems 20(8), 1453–1466 (2012)
2. Bernstein, D.J., Lange, T., Rezaeian Farashahi, R.: Binary Edwards Curves. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 244–265. Springer, Heidelberg (2008)
3. Bernstein, D.J.: Batch binary Edwards. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 317–336. Springer, Heidelberg (2009)
4. Brier, É., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002)
5. Brier, É., Déchène, I., Joye, M.: Unified point addition formulæ for elliptic curve cryptosystems. In: Embedded Cryptographic Hardware: Methodologies and Architectures, pp. 247–256. Nova Science Publishers (2004)
6. Chatterjee, A., Sengupta, I.: FPGA implementation of Binary edwards curve using ternary representation. In: GLSVLSI 2011, pp. 73–78 (2011)
7. Chatterjee, A., Sengupta, I.: High-speed unified elliptic curve cryptosystem on FPGAs using binary Huff curves. In: Rahaman, H., Chattopadhyay, S., Chattopadhyay, S. (eds.) VDAT 2012. LNCS, vol. 7373, pp. 243–251. Springer, Heidelberg (2012)
8. Devigne, J., Joye, M.: Binary huff curves. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 340–355. Springer, Heidelberg (2011)
9. Farashahi, R.R., Joye, M.: Efficient Arithmetic on Hessian Curves. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 243–260. Springer, Heidelberg (2010)
10. Huff, G.B.: Diophantine problems in geometry and elliptic ternary forms. Duke Math. J. 15, 443–453 (1948)
11. Izu, T., Takagi, T.: Exceptional procedure attack on elliptic curve cryptosystems. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 224–239. Springer, Heidelberg (2002)

12. Joye, M., Quisquater, J.-J.: Hessian elliptic curves and side-channel attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 402–410. Springer, Heidelberg (2001)
13. Joye, M., Yen, S.M.: The Montgomery powering ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
14. Joye, M., Tibouchi, M., Vergnaud, D.: Huff's model for elliptic curves. In: Hanrot, G., Morain, F., Thomé, E. (eds.) ANTS-IX 2010. LNCS, vol. 6197, pp. 234–250. Springer, Heidelberg (2010)
15. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
16. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^m) without precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999)
17. Rebeiro, C., Mukhopadhyay, D.: High speed compact elliptic curve cryptoprocessor for FPGA platforms. In: Chowdhury, D.R., Rijmen, V., Das, A. (eds.) INDOCRYPT 2008. LNCS, vol. 5365, pp. 376–388. Springer, Heidelberg (2008)
18. Stebila, D., Thériault, N.: Unified point addition formulæ and side-channel attacks. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 354–368. Springer, Heidelberg (2006)
19. Side-channel attack standard evaluation board, `http://www.morita-tech.co.jp/SASEBO/en/board/sasebo-g.html`
20. Walter, C.D.: Simple power analysis of unified code for ECC double and add. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 191–204. Springer, Heidelberg (2004)
21. Yen, S.-M., Kim, S., Lim, S., Moon, S.-J.: A countermeasure against one physical cryptanalysis may benefit another attack. In: Kim, K.-C. (ed.) ICISC 2001. LNCS, vol. 2288, pp. 414–427. Springer, Heidelberg (2002)

Appendix

Algorithm 1. SPA-resistant elliptic curve point multiplication.

Input: $P, d = 2^{m-1} + \sum_{i=0}^{m-2} 2^i d_i$.
Output: $Q = [d]P$.

1: $Q \leftarrow P$;
2: **for** i from $m-2$ **downto** 0 **do**;
3: $\quad Q \leftarrow Unif\_Add(Q, Q)$;
4: $\quad$ **if** $d_i = 1$ **then** $Q \leftarrow Unif\_Add(Q, P)$;
5: **Return** Q;

Table 4. RTL description of unified Huff formula

PA/PD Cycles	Operations	RTL description
1	$m_1 = x_1 \times x_2$	$temp[0] \leftarrow x_1 \times x_2$
2	$m_2 = y_1 \times y_2$	$temp[1] \leftarrow y_1 \times y_2$
3	$m_3 = z_1 \times z_2$	$temp[2] \leftarrow z_1 \times z_2$
4	$m_1 \times m_2$	$temp[3] \leftarrow temp[0] \times temp[1]$
5	$(x_1 + z_1)(x_2 + z_2)$	$temp[4] \leftarrow (x_1 \oplus z_1) \times (x_2 \oplus z_2)$
6	$m_6 = m_1 \times m_3$	$temp[5] \leftarrow temp[0] \times temp[2]$
7	$m_{11} = m_8 \times (m_2 + m_3)$	$temp[6] \leftarrow (temp[3] \oplus temp[2]^2) \times (temp[1] \oplus temp[2])$
8	$m_9 = m_6 \times (m_2 + m_3)^2$	$temp[5] \leftarrow temp[5] \times (temp[1] \oplus temp[2])^2$
9	$\alpha \times m_9$	$temp[5] \leftarrow \alpha \times temp[5]$
10	$m_{11} \times m_4$	$temp[4] \leftarrow temp[6] \times (temp[4] \oplus temp[0] \oplus temp[2])$
11	$Z_3 = m_{11} \times (m_1 + m_3)$	$temp[6] \leftarrow temp[6] \times (temp[0] \oplus temp[2])$
12	$m_7 = m_2 \times m_3$	$temp[5] \leftarrow temp[1] \times temp[2]$
13	$m_{10} = m_7 \times (m_1 + m_3)^2$	$temp[5] \leftarrow temp[5] \times (temp[0] \oplus temp[2])^2$
14	$m_{12} = m_8 \times (m_1 + m_3)$	$temp[4] \leftarrow (temp[3] \oplus temp[2]^2) \times (temp[0] \oplus temp[2])$
15	$(y_1 + z_1)(y_2 + z_2)$	$temp[3] \leftarrow (y_1 \oplus z_1) \times (y_2 \oplus z_2)$
16	$m_{12} \times m_5$	$temp[4] \leftarrow temp[4] \times (temp[3] \oplus temp[1] \oplus temp[2])$
17	$\beta \times m_{10}$	$temp[5] \leftarrow \beta \times temp[5]$
Final outputs are:		$X_3 \leftarrow temp[4] \oplus temp[5]$ at the end of step 10, $Z_3 \leftarrow temp[6]$ at the end of step 11, $Y_3 \leftarrow temp[4] \oplus temp[5]$ at the end of step 17.

Table 5. $Unif\_Add((x_1, y_1, z_1), (x_2, y_2, z_2))$: RTL description of proposed SPA-resistant unified Huff curve addition arithmetic

PA/PD Cycles	Operations	RTL description
1	$m_1 = x_1 \times x_2$	$temp[0] \leftarrow x_1 \times x_2$
2	$m_2 = y_1 \times y_2$	$temp[1] \leftarrow y_1 \times y_2$
3	$m_3 = z_1 \times z_2$	$temp[2] \leftarrow z_1 \times z_2$
4	$m_1 \times m_2$	$temp[3] \leftarrow temp[0] \times temp[1]$
5	$m_4 = (x_1 + z_1)(x_2 + z_2)$	$temp[4] \leftarrow (x_1 \oplus z_1) \times (x_2 \oplus z_2)$
6	$m_6 = m_1 \times m_3$	$temp[5] \leftarrow temp[0] \times temp[2]$
7	$m_{11} = m_8 \times (m_2 + m_3)$	$temp[6] \leftarrow (temp[3] \oplus temp[2]^2) \times (temp[1] \oplus temp[2])$
8	$m_9 = m_6 \times (m_2 + m_3)^2$	$temp[5] \leftarrow temp[5] \times (temp[1] \oplus temp[2])^2$
9	$m_{11} \times m_4$	$temp[4] \leftarrow temp[6] \times temp[4]$
10	$\alpha \times m_9$	$temp[5] \leftarrow \alpha \times temp[5]$
11	$Z_3 = m_{11} \times (m_1 + m_3)$	$temp[6] \leftarrow temp[6] \times (temp[0] \oplus temp[2])$
12	$m_7 = m_2 \times m_3$	$temp[5] \leftarrow temp[1] \times temp[2]$
13	$m_5 = (y_1 + z_1)(y_2 + z_2)$	$temp[4] \leftarrow (y_1 \oplus z_1) \times (y_2 \oplus z_2)$
14	$m_{10} = m_7 \times (m_1 + m_3)^2$	$temp[5] \leftarrow temp[5] \times (temp[0] \oplus temp[2])^2$
15	$m_5 \times m_8$	$temp[4] \leftarrow temp[4] \times (temp[3] \oplus temp[2]^2)$
16	$\beta \times m_{10}$	$temp[5] \leftarrow \beta \times temp[5]$
17	$(m_5 m_8) \times (m_1 + m_3)$	$temp[4] \leftarrow temp[4] \times (m_1 \oplus m_3)$
Final outputs are:		$X_3 \leftarrow temp[4] \oplus temp[5] \oplus temp[6]$ at clock cycle 12, $Z_3 \leftarrow temp[6]$ at clock cycle 15, $Y_3 \leftarrow temp[4] \oplus temp[5] \oplus temp[6]$ at clock cycle 19.