Secure Dual-Core Cryptoprocessor for Pairings Over Barreto-Naehrig Curves on FPGA Platform

Santosh Ghosh, Debdeep Mukhopadhyay, and Dipanwita Roychowdhury

***Abstract*—This paper is devoted to the design and the physical security of a parallel dual-core flexible cryptoprocessor for computing pairings over Barreto-Naehrig (BN) curves. The proposed design is specifically optimized for field-programmable gate-array (FPGA) platforms. The design explores the in-built features of an FPGA device for achieving an efficient cryptoprocessor for computing 128-bit secure pairings. The work further pinpoints the vulnerability of those pairing computations against side-channel attacks and demonstrates experimentally that power consumptions of such devices can be used to attack these ciphers. Finally, we suggest a suitable countermeasure to overcome the respective weaknesses. The proposed secure cryptoprocessor needs 1 730 000, 1 206 000, and 821 000 cycles for the computation of Tate, ate, and optimal-ate pairings, respectively. The implementation results on a Virtex-6 FPGA device shows that it consumes 23 k Slices and computes the respective pairings in 11.93, 8.32, and 5.66 ms.**

***Index Terms*—$\mathbb{F}_p$-arithmetic, field-programmable gate-array (FPGA) platform, pairing-based cryptography, power attack, programmable architecture, side-channel attack.**

I. Introduction

BILINEAR pairing first used in cryptography independently by Mitsunari *et al.* [36], Sakai *et al.* [40], and Joux [39] in 2000. One year later, Boneh and Franklin solved a long lasting problem of identity-based cryptography [37] based on pairing. Since then an impressive number of proposals arrived in the literature for designing cryptographic protocols based on pairings [29]. On the other hand, steep growth of the adversary's computation power demands increasing bit security in cryptographic protocols running in these applications. Practice has shown that one of the most efficient options to compute pairings for high bit security is to resort to Tate pairing operating on Barreto-Naehrig (BN) curves [22] defined over a 256-bit prime field having embedding degree $k = 12$.

Efficient computation of Tate pairing with linear complexity with respect to the size of the input was introduced long back in 1986 by Miller [30]. Significant improvements and the generalization of Miller's algorithm were independently proposed in 2002 by Barreto *et al.* [34] and Galbraith *et al.* [35]. Thereafter, intensive research has been carried out for further improvement of Tate pairing computations on specific algebraic curves. Recent breakthroughs include the η_T pairing [20], the ate pairing [23], the R-ate pairing [12], and the optimal-ate pairing [4]. Among them the η_T pairing is a symmetric pairing computed on supersingular curves, whereas other three belong to the class of asymmetric pairings on general elliptic curves.

In this paper, we extend the work presented in [4] and propose a pairing cryptoprocessor for BN curves. The design is flexible and resistant to side-channel attack. FPGA is one of the suitable platforms for implementing cryptographic algorithms. This paper proposes new implementation techniques of addition and multiplication on FPGAs. The in-built features available inside an FPGA device have been utilized to develop a high-speed 256-bit adder circuit. We show that when utilizing such adder circuits and adopting a parallelism technique, the multiplication in $\mathbb{F}_p$ can be substantially improved. Based on such $\mathbb{F}_p$ arithmetic cores, we develop a parallel configurable hardware for computing addition, subtraction, and multiplication on $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$. Existing techniques to speed up arithmetic in extension fields (see [21] and [24]) for fast computation in $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ are used on top of it. The major contributions of this paper are highlighted here.

- This paper implements underlying primitives for $\mathbb{F}_p$-arithmetic on FPGA platforms, which provides 1.7 times speedup from existing platform-independent techniques.
- A dual-core pairing cryptoprocessor for BN curves has been proposed on FPGA platform.
- Parallelism techniques are explored in different levels including underlying finite field operations which computes a pairing in 1/6-th number of clock cycles and achieves a comparable speed with the existing CMOS design.
- The paper further pinpoints the vulnerability of respective pairing computations against side-channel attacks. A differential power analysis (DPA) technique has been proposed. A suitable countermeasure is also outlined.

Section II gives an idea on cryptographic pairings and BN curves. Efficient design of finite field primitives on FPGA platforms are described in Section III. Section IV describes the proposed pairing cryptoprocessor, on which pairing computation is provided in Section V. In Section VI, we analyze the DPA attacks on pairings. Section VII shows the experimental results. Finally, the paper is concluded in Section VIII.

II. Background of Pairings

The name bilinear pairing indicates that it takes a pair of vectors as input and returns a number, and it performs linear transformation on each of its variables. For example, the dot product of vectors is a bilinear pairing [13]. Similarly, for cryptographic

Manuscript received May 30, 2011; revised September 21, 2011; accepted February 16, 2012. Date of publication April 09, 2012; date of current version February 20, 2013.

The authors are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, WB, 721302 Kharagpur, India (e-mail: santosh@cse.iitkgp.ernet.in; debdeep@cse.iitkgp.ernet.in; drc@cse.iitkgp.ernet.in).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2188655

applications the bilinear pairing (or pairing) operations are defined on elliptic or hyperelliptic curves. Pairing is a mapping $\mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$, where $\mathbb{G}_1, \mathbb{G}_2$ are additive subgroups of $E(\mathbb{F}_q)$, $E(\mathbb{F}_{q^k})$ and $\mathbb{G}_3$ is a subgroup of the multiplicative group of $E(\mathbb{F}_{q^k})$. The most important parameters for cryptographic pairings are the underlying finite field, the order of the curve, the embedding degree, and the order of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$. These parameters should be chosen such that the best exponential time algorithms to solve the discrete logarithm problem (DLP) in $\mathbb{G}_1$ and $\mathbb{G}_2$ and the subexponential time algorithms to solve the DLP in $\mathbb{G}_3$ take longer than a chosen security level.

The BN curve used in this paper is represented as: $E(\mathbb{F}_p)$: $Y^2 = X^3 + 3$ having the field characteristic $p = 36z^4 + 36z^3 + 24z^2 + 6z + 1$, the curve order $r = 36z^4 + 36z^3 + 18z^2 + 6z + 1$, and the trace of Frobenius $t = 6z^2 + 1$. With BN parameter $z = 6000000000001F2D$ (in hexadecimal) [21], it forms the group $E(\mathbb{F}_p)$ with r and p 256-bit primes of Hamming weight 91 and 87, respectively. The prime $p \equiv 7 \pmod 8$ (so -2 is a quadratic non-residue, we represent it by β).

Pairing computation consists of two major steps : the computation of Miller's function and the final exponentiation. Algorithm 1 shows computation of Tate pairing. The first part is computed by one of the optimized version of Miller's algorithm known as BKLS algorithm [34].

Algorithm 1: Computing the Tate pairing.

Input: $P \in G_1$, $Q \in G_2$, and $r = \sum_{i=0}^{L-1} r_i 2^i$.
Output: $e_r(P, Q)$.

1. $T \leftarrow P$, $f \leftarrow 1$;
2. **for** i *from* $L-2$ *downto* 0 **do**
3. $\quad T \leftarrow 2T$;
4. $\quad f \leftarrow f^2 \cdot l_{T,T}(Q)$;
5. $\quad$ **if** $r_i = 1$ *and* $i \neq 0$ **then**
6. $\quad T \leftarrow T + P$;
7. $\quad f \leftarrow f \cdot l_{T,P}(Q)$.
8. **return** $f^{(q^k-1)/r}$;

The BN curves admit a sextic twist [22], which means that the point Q is mapped on a point Q' defined over $\mathbb{F}_{p^2}$. Thus, the line functions $l_{T,T}(Q)$ and $l_{T,P}(Q)$ is computed over $\mathbb{F}_{p^2}$ instead of $\mathbb{F}_{p^{12}}$. Value of the line functions are represented as: $l_0 + l_1 W^2 + l_2 W^3$, with $l_0 \in \mathbb{F}_p$, $l_1, l_2 \in \mathbb{F}_{p^2}$, and a quadratic non-residue W over $\mathbb{F}_{p^2}$. The Miller function f is computed over $\mathbb{F}_{p^{12}}$, which is represented as: $f_0 + f_1 W + f_2 W^2 + f_3 W^3 + f_4 W^4 + f_5 W^5$, with $f_i \in \mathbb{F}_{p^2}$. In step 4 and step 7 of algorithm 1 the operations f^2, $f \cdot l_{T,T}(Q)$, and $f \cdot l_{T,P}(Q)$ are performed on $\mathbb{F}_{p^{12}}$, whereas all others are performed on $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$.

This paper follows the descriptions that are given in [18] for computing the Tate, ate, and optimal-ate pairings using Jacobian coordinate systems. The final exponentiation is performed by the technique provided in [11]. Let (m, s, i) denote the cost of multiplication, squaring, inversion in $\mathbb{F}_p$. As described in [18], the costs for Tate, ate, and optimal-ate pairings on BN curve are $35180m + i$, $23047m + i$, and $15093m + 2i$.

III. Implementing $\mathbb{F}_p$-Primitives on FPGA

In 1983, Blakley introduced an interesting algorithm to perform modular multiplication of two integers A and B modulo an integer M [43]. It is an iterative binary double-and-add algorithm. The main idea of the algorithm is that it keeps the intermediate result after each iteration below the modulus value, which it avoids final division. In this paper, the modulus M corresponds to p and we say it $\mathbb{F}_p$-multiplication. All arithmetic in $\mathbb{F}_p$ are performed in two's complement number system, which avoids input and output conversions like existing implementations [9], [10].

A. Fast Carry Chains for $\mathbb{F}_p$-Primitives

The main difficulty of the Blakley algorithm is the computation of addition on large operands. The modified Blakley algorithm for large operands are shown in [25] and [31]. The use of carry save adder (CSA) helps to speed up the repeated additions on large operands. However these modified versions require at least one final addition on large carry chain. Some pre-computed values too are used by this technique which require additional time and storage area.

This work exploits the features available in an FPGA device for efficient computation of Blakley algorithm on large operands. The specific features that are available in an FPGA device are efficiently utilized for developing arithmetic primitives in $\mathbb{F}_{2^m}$ fields in [17]. However, this paper looks after the same for $\mathbb{F}_p$. The modern FPGA consists of 16 slices (or 32 LUTs) within a single row which are connected through an in-built fast carry chain (FCC). The FCC can perform addition on two 32-bit operands most efficiently compared to any other adder structures [7], . It is experimentally shown that on a Virtex-4 FPGA device the latency of a 32-bit addition using a fast carry chain takes only 5.8 ns, whereas the same using a carry lookahead structure takes 8.7 ns. Hence, fast carry chain is 1.5 times faster than carry lookahead structure for computing addition of two 32-bit operands on an FPGA platform. In order to compute an addition of two operands longer than 32 bits, the FPGA will utilize more than one row which requires additional routing delay. For example, the addition $(A+B)$ of two 64-bit operands (A, B) using a single 64-bit carry chain is slower than the same using three 32-bit FCC and a 2:1 multiplexer [7].

We develop an efficient 256-bit adder useful to F_{p256}-arithmetic using 32-bit fast carry chains. The repeated Karatsuba decomposition is applied on 256-bit operands. An operand is decomposed upto a depth of three for converting it into eight pieces of 32-bit operands. A 64-bit addition is performed by using three 32-bit fast carry chains with a carry select structure. Let, $A = A_1 2^{32} + A_0$, $B = B_1 2^{32} + B_0$, and $C = A + B$, where A_i, B_i are 32-bit integers. We compute $A_0 + B_0$, $A_1 + B_1 + 0$, and $A_1 + B_1 + 1$ in parallel on three FCC. Then the carry out of the least significant addition $(A_0 + B_0)$ is used to multiplex the results of the most significant additions. Thus the latency of a 64-bit adder is 1 FCC + 1 MUX, where MUX corresponds to a 2:1 multiplexer. Similarly, an 128-bit adder is developed by three 64-bit adders, and a 256-bit adder is developed by three 128-bit adders. Therefore, a 256-bit adder is developed hierarchically from 32-bit adders. At every level of hierarchy it adds

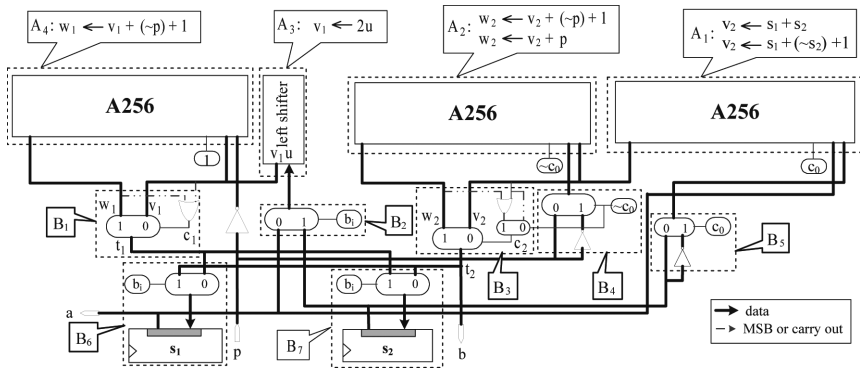

Fig. 1. Architecture of $\mathbb{F}_p$ adder/subtractor/multiplier unit.

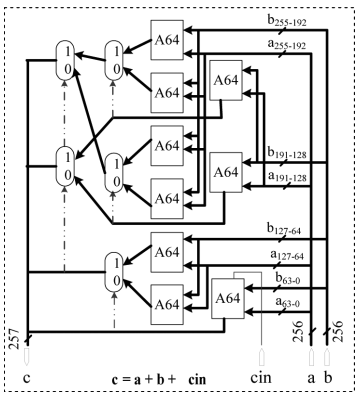

Fig. 2. Internal structure of a 256-bit adder—the A256 block.

one additional MUX in the critical path. Thus the latency of a 256-bit adder is 1 FCC + 3 MUX delay, which is 9.9 ns on a Virtex-4 FPGA, whereas the latency of a 256-bit carry lookahead adder on the same platform is 16.7 ns, which is 1.7 times slower than the above technique.

B. Programmable $\mathbb{F}_p$-Primitive

In this section we develop a programmable $\mathbb{F}_p$-primitive based on above 256-bit high-speed adder circuits. Essential operations for pairing computation are addition, subtraction, and multiplication in finite fields. Fig. 1 depicts the overall resulting architecture of the proposed $\mathbb{F}_p$-adder/subtractor/multiplier unit, where the internal dataflow of A256 blocks are shown in Fig. 2.

1) Architecture Description: Our first objective for designing such an integrated architecture is to reduce the overall hardware costs for computing three essential prime field operations in pairing computation. The architecture consists of several independent blocks which operate in parallel for accelerating the execution of respective operations. The whole architecture is subdivided into four macro-blocks (A_1, A_2, A_3, A_4) and seven micro-blocks $(B_1, B_2, B_3, B_4, B_5, B_6, B_7)$. The macro-blocks are used to compute the arithmetic operations, whereas, micro-blocks are primarily responsible for dataflow among the macro-blocks, the registers, and the i/o ports. The functionality of the individual blocks are described here.

- Macro-blocks A_1, A_2, and A_4 are 256-bit adders based on our proposed technique as described in Section III-A.
- Block A_3 performs $2u$ for an integer $u \in \mathbb{F}_p$. This is done by simply one bit left shift having only rewiring and no additional logic cells.
- Micro-block B_1 consists of one 2:1 multiplexer that selects either v_1 or w_1 based on the most significant bits (or carry-outs) of $2u$ and $2u - p$ operations. Therefore, this block completes the $2u \bmod p$ operation.
- Block B_2 selects either s_1 or s_2 as the input to the A_3.
- Blocks B_3, B_4, and B_5 help to compute $\mathbb{F}_p$—addition and subtraction in A_1 and A_2. The control signal c_0 holds zero for addition and one for subtraction. Thus, if $c_0 = 1$ then block B_5 selects $\neg s_2$ else it selects s_2. Similarly, if $c_0 = 1$ then block B_4 selects p else it selects $\neg p$. Block B_3 completes the operation by selecting the correct result. In case of $\mathbb{F}_p$-subtraction (i.e., $c_0 = 1$), it selects either v_2 or w_2 based on the most significant bit (MSB) of v_2 only, whereas, for $\mathbb{F}_p$-addition it does the same based on the MSB of both v_2 and w_2.
- Blocks B_6 and B_7 multiplex t_1 (the output of $2u \bmod p$) and t_2 (the output of $s_1 \pm s_2 \bmod p$) as the new value of s_1 and s_2 registers, respectively.

2) Computation of $\mathbb{F}_p$-Multiplication: Proposed $\mathbb{F}_p$-primitive follows the parallelism technique of Montgomery ladder [32] for computing Blakley multiplication algorithm in $\mathbb{F}_p$[43]. Basic Blakley technique for computing $c = ab \bmod p$ with input parameters a and b is as follows:

$c \leftarrow 0$
for i from 0 to $k-1$ **do**
  $c \leftarrow 2c + a.b_{k-1-i}$
  $c \leftarrow c \bmod p$
return c.

The choice of this algorithm is due to its lower hardware cost and intrinsic adaptability to Montgomery ladder for parallelism. We rewrite it, in Algorithm 2 with parenthesized indices in superscript in order to emphasize the intrinsic dependency as well as parallelism of the multiplication procedure.

Algorithm 2: The interleaved multiplication based on Montgomery ladder †.

Input: p, $a = \sum_{i=0}^{n-1} 2^i a_i$ and $b = \sum_{i=0}^{n-1} 2^i b_i$.

Output: $a \cdot b \bmod p$.

1. $s_1^{(n)} \leftarrow 0$; $s_2^{(n)} \leftarrow a$;
2. **for** $i = n-1$ down to 0 **do**
3. **if** $b_i = 1$ **then** $u^{(i)} \leftarrow s_2^{(i+1)}$; **else** $u^{(i)} \leftarrow s_1^{(i+1)}$;
4. $v_1^{(i)} \leftarrow 2u^{(i)}$;
5. $v_2^{(i)} \leftarrow s_1^{(i+1)} + s_2^{(i+1)}$;
6. $w_1^{(i)} \leftarrow v_1^{(i)} + (\neg p) + 1$;
7. $w_2^{(i)} \leftarrow v_2^{(i)} + (\neg p) + 1$;
8. $c_1^{(i)} \leftarrow (v_1^{(i)})_n \mid (w_1^{(i)})_n$;
9. $c_2^{(i)} \leftarrow (v_2^{(i)})_n \mid (w_2^{(i)})_n$;
10. **if** $c_1^{(i)} = 1$ **then** $t_1^{(i)} \leftarrow w_1^{(i)}$; **else** $t_1^{(i)} \leftarrow v_1^{(i)}$;
11. **if** $c_2^{(i)} = 1$ **then** $t_2^{(i)} \leftarrow w_2^{(i)}$; **else** $t_2^{(i)} \leftarrow v_2^{(i)}$;
12. **if** $b_i = 1$ **then** $s_1^{(i)} \leftarrow t_1^{(i)}$; **else** $s_1^{(i)} \leftarrow t_2^{(i)}$;
13. **if** $b_i = 1$ **then** $s_2^{(i)} \leftarrow t_2^{(i)}$; **else** $s_2^{(i)} \leftarrow t_1^{(i)}$;
14. **end for**
15. **return** $s_1^{(0)}$;

† In the algorithm, $x^{(i)}$ represents the value of x at ith iteration, $(x)_n$ indicates the nth bit of x, and $\mid$ indicates logical OR.

The algorithm computes two intermediate results $s_1^{(i)}$ and $s_2^{(i)}$) in each iteration. The data transfer inside the architecture (see Fig. 1) for computing $(a \cdot b) \bmod p$ is as follows.

- The register s_1 and s_2 hold the iterative results $s_1^{(i)}$ and $s_2^{(i)}$ of Algorithm 2, which are initialized by zero and a, respectively, as specified in step 1.
- Iterative execution starts from $i = n-1$ and goes down to zero as shown in step 2. This step is executed by a 8-bit counter, which belongs to the control part of the proposed design and it is not shown in Fig. 1.
- Block B_2 of Fig. 1 executes step 3. The modular doubling (as computed by executing the steps 4, 6, 8, and 10) and the modular addition (as computed by executing the steps 5, 7, 9, and 11) are performed in parallel. In Fig. 1, steps 4 and 6 are performed in blocks A_3 and A_4, respectively, whereas, both the steps 8 and 10 are performed in block B_1. Similarly, steps 5 and 7 are performed in blocks A_1 and A_2, respectively, whereas, both the steps 9 and 11 are performed in block B_3. During the execution of $\mathbb{F}_p$-multiplication control signal c_0 remains zero.
- Finally, results of the current iteration are restored as specified in step 12 and step 13 in parallel by B_6 and B_7 blocks.

All steps from step 3 to step 13 of Algorithm 2 are performed within one clock by the proposed architecture. Therefore, to compute a multiplication in $\mathbb{F}_{p256}$ the proposed design takes only 256 clock cycles.

3) Computation of $\mathbb{F}_p$-Addition: The proposed design executes Algorithm 3 for computing $\mathbb{F}_p$-addition. As described in step 1, the architecture initializes registers s_1 and s_2 by operands a and b, respectively. It executes steps 2 and 3 in blocks A_1 and A_2. Based on the most significant bits of v_2 and w_2 it produces the correct result of $s_1 + s_2 \bmod p$ in block B_3 as described in step 3 and step 4. During the execution of $\mathbb{F}_p$-addition the control signal c_0 holds logic zero. The proposed architecture computes a $\mathbb{F}_p$-addition in one clock cycle.

Algorithm 3: The addition in prime field.

Input p, $a = \sum_{i=0}^{n-1} 2^i a_i$ and $b = \sum_{i=0}^{n-1} 2^i b_i$.
Output $a + b \bmod p$.

1. $s_1 \leftarrow a$; $s_2 \leftarrow b$;
2. $v_2 \leftarrow s_1 + s_2$;
3. $w_2 \leftarrow v_2 + (\neg p) + 1$;
4. $c_2 \leftarrow (v_2)_n \mid (w_2)_n$;
5. **if** $c_2 = 1$ **then** $t_2 \leftarrow w_2$; **else** $t_2 \leftarrow v_2$;
6. **return** t_2;

4) Computation of $\mathbb{F}_p$-Subtraction: Subtraction $a - b \bmod p$ on the proposed design is performed by executing Algorithm 4. It is executed by the architecture mostly like Algorithm 3 with additional help by block B5 for $\neg s_2$.

Algorithm 4: The subtraction in prime field.

Input: p, $a = \sum_{i=0}^{n-1} 2^i a_i$ and $b = \sum_{i=0}^{n-1} 2^i b_i$.
Output: $a - b \bmod p$.

1. $s_1 \leftarrow a$; $s_2 \leftarrow b$;
2. $v_2 \leftarrow s_1 + (\neg s_2) + 1$;
3. $w_2 \leftarrow v_2 + p$;
4. $c_2 \leftarrow (v_2)_n$;
5. **if** $c_2 = 1$ **then** $t_2 \leftarrow w_2$; **else** $t_2 \leftarrow v_2$;
6. **return** t_2;

IV. Dual-Core Pairing Cryptoprocessor

The main novelty of the architecture lies in its efficient utilization of FPGA features. Independent operations are exploited at each level of pairing computations to evolve an optimized parallel design. We explain here the top level of the design followed by its internal parts.

A. Datapath Design

The major operations for pairing computations are point doubling (PD), point addition (PA), line computation ($l(Q)$), f^2, and $f \cdot l(Q)$. In case of Tate pairing on BN curve, the PA and PD are performed on $E(\mathbb{F}_p)$. Hence, the underlying operations are performed in $\mathbb{F}_p$. Similarly, the operation $l(Q)$ is performed in $\mathbb{F}_{p^2}$, while the other two operations are performed in $\mathbb{F}_{p^{12}}$. In case of ate and optimal-ate pairings, the PA, PD, $l(Q)$ are performed in $\mathbb{F}_{p^2}$, and f^2, $f \cdot l(Q)$ are performed in $\mathbb{F}_{p^{12}}$. However, each of the above computations are well defined and constitute a number of independent $\mathbb{F}_p$-operations. The proposed datapath executes those independent operations in parallel to speed up pairing computations.

Fig. 3 shows the overall resulting structure of the datapath. Two configurable $\mathbb{F}_{p^k}$ arithmetic units (CAU) are included which perform arithmetic in $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ depending on their mode of configurations. The instructions to configure the CAUs are stored into a small memory segment called *instruction memory*. There is a special instruction fetch and decode (IFD) unit which reads the respective instructions and converts them

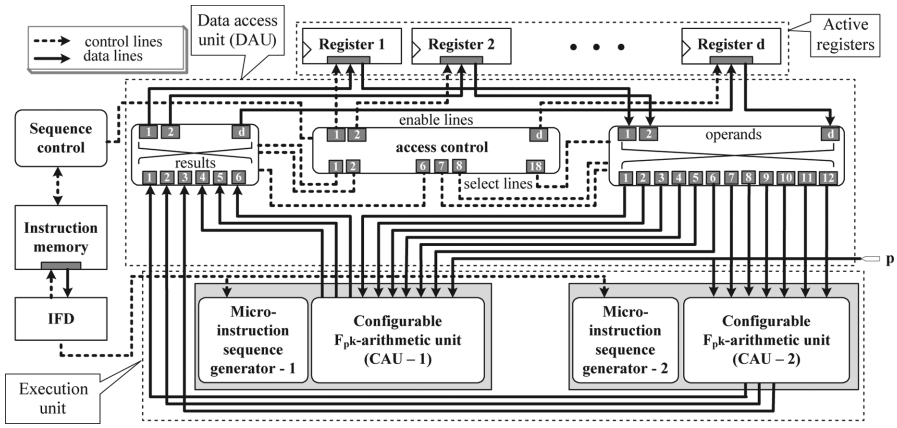

Fig. 3. Datapath of the pairing cryptoprocessor.

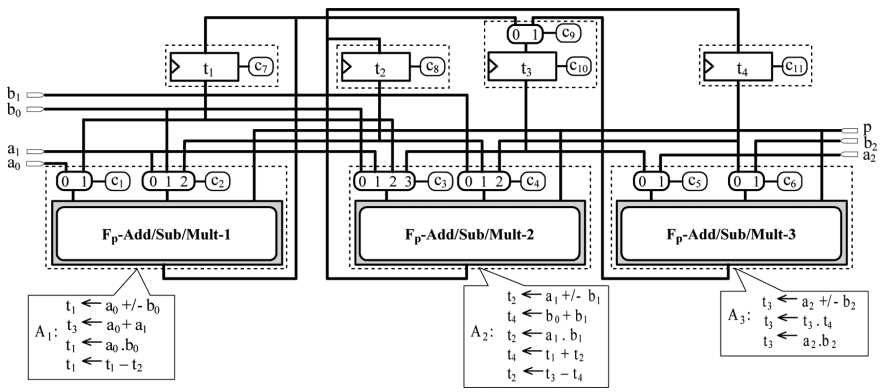

Fig. 4. Architecture of configurable $\mathbb{F}_{p^k}$ arithmetic unit (CAU).

to proper configuration signals for both the CAUs. The input data to the CAUs come in parallel from respective registers. The mechanism and regularity of data access for computing above operations are fairly simple. The distribution of access to the registers and resolution of access conflicts are handled efficiently at the runtime by a dedicated hardware block called data access unit (DAU) which communicates among the CAUs and the registers.

Each CAU performs atmost three $\mathbb{F}_p$-operations in parallel. Thus, overall 12 independent operands along with modulus p and 6 outputs are accessed in either directions between memory elements and the CAUs. This on-demand concurrent data requests result in multiple independent read or write connections between CAUs and DAU. The DAU takes care of granting accesses. Therefore, a simple multiplexing protocol is used between CAUs and registers, which is able to confirm a request within the same cycle in order not to cause any delay cycles when trying to access data in parallel. The data accesses and instruction sequences are hard coded into the *sequence control* of the architecture which avoids the additional software development costs.

The data access conflicts have been resolved prior to design of the DAU. The proposed one is a custom hardware for pairing computations which executes a fixed set of operations. The dependency of the instructions are predefined and thus the access conflicts are known. The priority of the data processing and the respective execution is rearranged accordingly which achieves maximum utilization of CAUs.

The `data access unit` or DAU acts as a mediator while transferring data between CAUs and memory elements. Due to the demand of parallel access, the proposed cryptoprocessor stores all intermediate results in its active registers. To fulfil our aimed parallelism of pairing computations on BN curves the proposed design consists of 50 256-bit registers (i.e., $d = 50$ in Fig. 3). Each of the register consists of data-in, data-out, and enable lines. It gets updated by data-in lines when the respective enable signal is invoked. The crossbar switch (*results*) redirects the outputs of each operation to registers. Similarly, the *operands* are redirected from registers to the input ports of the CAUs. The respective select signals are generated prior to the above two redirection procedures by the sequence control unit. The *access control* block synchronizes the select lines of the multiplexers for operands and results. It also synchronizes the enable signals of registers for restoring the intermediate results.

1) Configurable $\mathbb{F}_{p^k}$ Arithmetic Unit (CAU): Fig. 4 shows the architecture of the proposed CAU. It consists of three $\mathbb{F}_p$-adder/subtractor/multiplier units described before in Section III-B. Each of these units along with their input multiplexers are identified as separate blocks (A_1, A_2, A_3), which can operate in parallel. The CAU operates on two modes; namely, $\mathbb{F}_p$-mode and $\mathbb{F}_{p^2}$-mode. In $\mathbb{F}_p$-mode, it computes three independent $\mathbb{F}_p$-operations on A_1, A_2, and A_3 blocks. The respective operations that are computed in this mode are $t_1 \leftarrow a_0[+/-/\cdot]b_0$, $t_2 \leftarrow a_1[+/-/\cdot]b_1$, and $t_3 \leftarrow a_2[+/-/\cdot]b_2$ as shown in the figure.

TABLE I
MICRO-INSTRUCTIONS FOR PERFORMING ARITHMETIC IN $\mathbb{F}_p$ AND $\mathbb{F}_{p^2}$

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}
$\mathbb{F}_p$-mode, execution of three $\mathbb{F}_p$-operations in parallel										
0	1	1	0	1	1	1	1	1	1	0
$\mathbb{F}_{p^2}$-mode, execution of Algorithm 5†										
0	0	0	0	0	0	0	0	0	1	1
0	1	1	0	0	0	1	1	1	1	0
1	2	2	1	0	0	1	0	0	0	1
1	2	3	2	0	0	1	1	0	0	0

† : Each row of micro-instructions represents one respective step in Algorithm 5.

In $\mathbb{F}_{p^2}$-mode, the CAU computes $\mathbb{F}_{p^2}$-multiplication. Let an element $\alpha \in \mathbb{F}_{p^2}$ be represented as $\alpha_0 + \alpha_1 X$, where $\alpha_0, \alpha_1 \in \mathbb{F}_p$ and X is an indeterminate. The formula of Karatsuba multiplication $c = ab$ in $\mathbb{F}_{p^2}$ is

$$v_0 = a_0 b_0,\ v_1 = a_1 b_1$$
$$c_0 = v_0 + \beta v_1$$
$$c_1 = (a_0 + a_1)(b_0 + b_1) - v_0 - v_1$$

where $v_0, v_1, c_0, c_1, a_0, a_1, b_0, b_1 \in \mathbb{F}_p$. Here β is a quadratic non-residue in $\mathbb{F}_p$ which is -2 in case of BN curve. We compute $a \cdot b$ in the proposed CAU as described in Algorithm 5.

Algorithm 5: The multiplication in $\mathbb{F}_{p^2}$.

Input: p, $a = a_0 + a_1 X$ and $b = b_0 + b_1 X$.
Output: $a \cdot b$.

1. $t_3 \leftarrow a_0 + a_1$; $t_4 \leftarrow b_0 + b_1$;
2. $t_1 \leftarrow a_0 \cdot b_0$; $t_2 \leftarrow a_1 \cdot b_1$; $t_3 \leftarrow t_3 \cdot t_4$;
3. $t_1 \leftarrow t_1 - t_2$; $t_4 \leftarrow t_1 + t_2$;
4. $t_1 \leftarrow t_1 - t_2$; $t_2 \leftarrow t_3 - t_4$.
5. **return** $t_1 + t_2 X$;

All operations within a step of the Algorithm 5 are computed in parallel, whereas, individual steps are executed one-by-one. Step 1 of the algorithm is computed by block A_1 and block A_2. Then the CAU executes three independent $\mathbb{F}_p$-multiplications as defined in step 2 by A_1, A_2, and A_3, respectively. After executing steps 3 and 4 by A_1 and A_2 blocks the final result is stored into the registers t_1 and t_2 as defined in step 5. The cost of multiplication in $\mathbb{F}_{p^2}$ is 3 m, where m represents the cost of one $\mathbb{F}_p$-multiplication. However, due to three parallel independent $\mathbb{F}_p$-multiplication units this cost on the proposed CAU is only m. The $\mathbb{F}_{p^2}$-squaring is performed as $a \cdot a$ for reducing the multiplexer complexity of the CAU for which too we pay the same cost.

The *micro instruction sequence generator* finds the current operation type and generates the respective micro instructions which are nothing but the control signals c_i, $1 \leq i \leq 11$. The respective values of control signals, which on the other hand, represents the scheduling of different operations on CAU are depicted in Table I.

This sequence generator is constructed as a typical state machine which generates micro instructions at each state. Its deterministic state transition takes place at every clock cycle based on the current state and overall status of the CAU. In case of a multiplication in $\mathbb{F}_{p256}$, it remains in a same state for 256 cycles, whereas it remains for one cycle only in a state for computing $\mathbb{F}_{p256}$—addition and subtraction. Thus, the cost m means 256 clock cycles in the proposed pairing cryptoprocessor. Similarly, the computation of $c = ab$ in $\mathbb{F}_{(p256)^2}$ takes only 259 clock cycles which is approximately equal to the cost of m.

V. PAIRING COMPUTATIONS

This section describes the computation of Algorithm 1 on our dual-core cryptoprocessor. The Tate pairing (t_r) over BN curve takes input points P and Q over $\mathbb{F}_p$ and $\mathbb{F}_{p2}$, respectively. The Miller loop runs for 255 iterations having 255 doubling steps and 90 addition steps. Our dual-core design consists of a fixed number of functional units. Therefore, an optimization is done based on the available functional units and the operations which is described here.

A. Computation of Doubling Step

The doubling step consists of the following computations.

- The point doubling $(2T)$ operation.
- The computation of tangent line at point $T(l_{T,T}(Q))$.
- The squaring of Miller function (f^2).
- The multiplication of Miller function with line function $(f^2 \cdot l_{T,T}(Q))$.

The computation of $2T$, $l_{T,T}(Q)$, and f^2 are performed in parallel on our design. In Jacobian coordinates the formulae for doubling a point $T = (X, Y, Z)$ are $2T = (X_3, Y_3, Z_3)$, where $X_3 = 9X^4 - 8XY^2$, $Y_3 = (3X^2)(4XY^2 - X_3) - 8Y^4$ and $Z_3 = 2YZ$. The tangent line at T, after clearing 26 denominators, is $l(x, y) = 3X^3 - 2Y^2 - 3X^2Z^2x + Z_3Z^2y$ [25].

In case of Tate pairing computation on BN curves, the parameters $\{x, y\} \in \mathbb{F}_{p^2}$ and $\{X, Y, Z, X_3, Y_3, Z_3\} \in \mathbb{F}_p$. Let us assume that x and y are represented as $x_0 + x_1\mathcal{U}$ and $y_0 + y_1\mathcal{U}$, where $\{x_0, x_1, y_0, y_1\} \in \mathbb{F}_p$ and $\mathcal{U}$ is an indeterminant. The computation of X_3, Y_3, Z_3, and $l(x, y)$ are performed by one of the CAUs by 14 instructions, among them, 6 are nonlinear $\mathbb{F}_p$-operations. If we assume that $\mathbb{F}_p$ squaring $(s) \approx \mathbb{F}_p$ multiplication (m) then the cost of above operations is $6m$ on a CAU.

At the same time, other core starts the computation of f^2. We represent the Miller function $f \in \mathbb{F}_{((p^2)^3)^2}$ as : $(f_{0,0} + f_{0,1}\mathcal{V} + f_{0,2}\mathcal{V}^2) + (f_{1,0} + f_{1,1}\mathcal{V} + f_{1,2}\mathcal{V}^2)\mathcal{W}$, where $f_{i,j} \in \mathbb{F}_{p^2}$.

The operation $c = f^2$ is performed using complex method as: $v = f_0 \cdot f_1$; $c_0 = (f_0 + f_1)(f_0 + \beta f_1) - v - \beta v$; $c_1 = 2v$; where v, c_0, c_1 are in $\mathbb{F}_{p^6}$ and is a quadratic non-residue, and thus $c = c_0 + c_1 W$. It requires two $\mathbb{F}_{p^6}$ multiplications. Now, one $\mathbb{F}_{p^6}$- multiplication is performed in the tower field $\mathbb{F}_{(p^2)^3}$ using Karatsuba technique by six multiplications in $\mathbb{F}_{p^2}$ [24]. Let us consider that an element $a_i \in \mathbb{F}_{p^2}$ is represented as: $a_{i0} + a_{i1}U$, $a_{ij} \in \mathbb{F}_p$. The result $v = f0 \cdot f1 \in \mathbb{F}_{p^6}$ is represented as: $(v_{00} + v_{01}U) + (v_{10} + v_{11}U)V + (v_{20} + v_{21}U)v^2$, where $v_{ij} \in \mathbb{F}_p$. It is perform on a CAU in the cost of $6m$. This computation is performed in parallel with $2T$, $l_{T,T}(Q)$, which are executed in other CAU. The second $\mathbb{F}_{p^6}$ multiplication, i.e., the computation of $(f_0 + f_1)(f_0 + \beta f_1)$ is performed by both the programmable cores, which costs only $3m$ in the proposed design. Therefore, the total cost of computing $2T$, $l_{T,T}(Q)$, and f^2 is $9m$.

The $l(Q)$ is represented as: $(l_0 + l_1\mathcal{V}) + (l_2\mathcal{V})\mathcal{W}$, where $l_0 \in \mathbb{F}_p, l_1, l_2 \in \mathbb{F}_{p^2}$, which is equivalent to $l_0 + l_1\mathcal{W}^2 + l_2\mathcal{W}^3$. The computation of $f \cdot l(Q)$ is performed in the tower field $\mathbb{F}_{((p^2)^3)^2}$ as: $f' = f \cdot l(Q) = ((f_{0,0} + f_{0,1}\mathcal{V} + f_{0,2}\mathcal{V}^2), +(f_{1,0} + f_{1,1}\mathcal{V} + f_{1,2}\mathcal{V}^2)\mathcal{W}), \cdot((l_0 + l_1\mathcal{V}) + (l_2\mathcal{V})\mathcal{W})$. The top most extension is quadratic. Thus the computation of $f \cdot l(Q)$ is done by three $\mathbb{F}_{p^6}$ multiplications, which are identified as: $t_1^1 = (l_0 + l_1\mathcal{V}) \cdot (f_{0,0} + f_{0,1}\mathcal{V} + f_{0,2}\mathcal{V}^2)$, $t_2^1 = (l_2\mathcal{V}) \cdot (f_{1,0} + f_{1,1}\mathcal{V} + f_{1,2}\mathcal{V}^2)$, $t_3^1 = (l_0 + (l_1 + l_2)\mathcal{V}) \cdot (((f_{0,0} + f_{1,0}), +(f_{0,1} + f_{1,1})\mathcal{V} + (f_{0,2} + f_{1,2})\mathcal{V}^2)$. One multiplication in $\mathbb{F}_{p^6}$ using Karatsuba method requires 18 $\mathbb{F}_p$ multiplications. However, due to the sparse representation of $l(Q)$ the cost of computing $t_i^1,\ 1 \leq i \leq 3$ is lesser than the actual costs of three $\mathbb{F}_{p^6}$ multiplications. Each of the equations for t_1^1 and t_3^1 requires 14 $\mathbb{F}_p$ multiplications. In our parallel cryptoprocessor, the above two equations are computed in parallel on two CAUs, which costs $5m$. The computation of t_2^1 requires only nine $\mathbb{F}_p$ multiplications, which is performed on both the cores and it costs only $2m$. Therefore, the computation of $f \cdot l(Q)$ requires 37 $\mathbb{F}_p$ multiplications, which costs only $7m$ in our design. In total, the cost of doubling step (the computation of $2T, l_{T,T}(Q), f^2$, and $f \cdot l(Q)$) is $9m + 7m = 16m$.

B. Computation of Addition Step

The addition step consists of the computations of $T + P$, $l_{T,P}(Q)$, and $f \cdot l_{T,P}(Q)$. The formulae for mixed Jacobian-affine addition are the following: if $T = (X_1, Y_1, Z_1)$ is in Jacobian coordinates and $P = (X_2, Y_2)$ is in affine coordinates, then $T + P = (X_3, Y_3, Z_3)$, where $X_3 = (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)^2(X_1 + X_2Z_1^2)$,$Y_3 = (Y_2Z_1^3 - Y_1)(X_1(X_2Z_1^2 - X_1)^2 - X_3) - Y_1(X_2Z_1^2 - X_1)^3$, $Z_3 = Z_1(X_2Z_1^2 - X_1)$. The line through T and P is $l(x, y) = (X_2(Y_2Z_1^3 - Y_1) - Y_2Z_3) - (Y_2Z_1^3 - Y_1)x + Z_3 \cdot y$. During the addition step of Miller algorithm we compute the above operations in parallel on both cores. There are limited independent operations in this step. Therefore, there are scopes for optimizing the scheduling of operations on $\mathbb{F}_p$ arithmetic units for reducing the additional registers and related wiring. The cost of addition step is $12m$.

C. Computation of Final Exponentiation

The final exponentiation follows the optimization to factor $(p^{12} - 1)/r$ into three parts [21] and compute $f^{(p^{12}-1)/r}$ as: $f^{(p^{12}-1)/r} = f^{(p^6-1)\times(p^6+1)/(p^4-p^2+1)\times(p^4-p^2+1)/r} = ((f^{(p^6-1)})^{(p^2+1)})^{(p^4-p^2+1)/r}$.

The power of $(p^6 - 1)$ in $\mathbb{F}_{(p^6)^2}$ is an easy exponentiation, which is performed by a conjugation (Frobenius) and a division [12], [15]. The operation $f^{p^6} = f_0 - f_1\mathcal{W}$. Thus, f^{p^6-1} is performed by one inversion and one multiplication in $\mathbb{F}_{p^{12}}$, which costs $29m$ on our dual-core design. The exponentiations f^{6z+5}, T^z and $(T^z)^{6z}$ are performed by repeated square-and-multiply.

D. Pairing Computations

The total cost for evaluating iterative Miller function of the Tate pairing computation is $5176m$ on our proposed dual-core cryptoprocessor. The cost for computing the final exponentiation is $1610m$. Hence, the total cost for computing a Tate pairing over BN curves by our cryptoprocessor is $6786m$, which takes 1 730 000 cycles. Similarly, cost of ate pairing is $4642m \approx 1\,206\,902$ clock cycles, and costs for optimal-ate pairing is $3158m \approx 821\,080$ clock cycles.

VI. Side-Channel Vulnerability

Page and Vercauteren [28] presented SPA and DPA attacks on the pairing computations performed by the Duursma-Lee algorithm [32] and the BLKS algorithm [34] over $\mathbb{F}_{3^m}$. The power consumption attack on η_T pairing computation over $\mathbb{F}_{2^m}$ is described by Kim *et al.* [19]. However, the same in case of $\mathbb{F}_p$ has not been studied so far. In the decryption step of identity-based encryption schemes [38], a dominant operation is $e(U, S_{ID})$, where S_{ID} is the fixed secret key, and U is a part of a ciphertext. During the addition step of Tate pairing computation the formula of the line function is $l(x, y) = (y - Y_2)Z_3 - (x - X_2)(Y_2Z_1^3 - Y_1)$[27]. In pairing-based cryptographic schemes, the point $T = (X_1, Y_1, Z_1)$ is an intermediate resultant point of current point doubling operation, the point $U = (X_2, Y_2)$ is used as a public parameter (it could be the plain texts or messages), and $S_{ID} = (x, y)$ is used as the private key. The resultant point $(T + U)$ is represented by (X_3, Y_3, Z_3). Therefore, in such a scheme the operations $(x - X_2)$ and $(y - Y_2)$ could be exploited through side-channel attacks.

A. Proposed DPA Attack

In this section, we investigate differential power analysis (or DPA) attack against the subtraction $(x - X_2)$ used in the Tate pairing on elliptic curves in $\mathbb{F}_p$, where x is secret and X_2 is public and known to, or even chosen by, the attacker. The subtraction $(x - X_2)$ in $\mathbb{F}_p$ is computed by first computing $S = x - X_2$ and then the result is reduced (if required) by adding p with S. Let us assume that all operations are performed on 2's complement numbers. Therefore, the subtraction $S = x - X_2$ could be performed as: $S = \sum_{i=0}^{k} 2^i s_i = \sum_{i=0}^{k-1} 2^i x_i + \sum_{i=0}^{k-1} 2^i \bar{X}_{2_i} + 1$, where k represents the bit length of operands (x, X_2) and $\bar{X}_{2_i}$ corresponds to the 1's complement of X_{2_i}. The subtraction is started from the least significant bit (or LSB) by computing *sum* and *carry* bits iteratively. The formula for i-th carry bit is: $c_i = x_i\bar{X}_{2_i} \oplus x_ic_{i-1} \oplus \bar{X}_{2_i}c_{i-1}$. Similarly, the ith sum bit is computed as: $s_i = x_i \oplus \bar{X}_{2_i} \oplus c_{i-1}$ for $k - 1 \leq i \leq 0$ with $c_{-1} = 1$.

The attacker first collects the power consumption traces of n number of randomly chosen public point U. We consider the simplified Hamming weight model for power leakage [38]. The power consumption W is computed as: $W = \varepsilon H + \eta$, where H, ε, and η represent the Hamming weight of the intermediate data, the incremental amount of power for each extra 1 in the Hamming weight, and the noise, respectively. We assume that the average of noise η is zero.

Let W be the power consumption associated with the subtraction operation $(x - X_2)$. We start from the LSB and iteratively find all bits of the x-coordinate of the secret point $S_{ID} = (x, y)$. To recover the ith bit of x, we guess that $x_i = 0$ and divide power consumptions into two sets by $\bar{X}_{2_i} \oplus c_{i-1}$

$$P_k = \{\ W \mid \bar{X}_{2_i} \oplus c_{i-1} = k\} \quad \text{with } k = \{0, 1\}.$$

Thus, the differential power consumption is $\Delta =< P_1 - P_0 >$. If the guess is correct, then the averages of P_1 and P_0

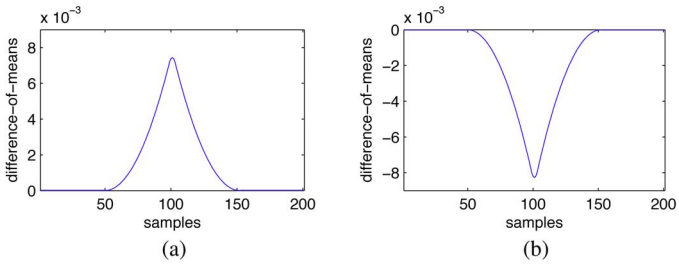

Fig. 5. Correlation between LSB and corresponding average power differences of an addition in $\mathbb{F}_p$ (a) for $x_0 = 0$ and (b) for $x_0 = 1$.

are, $\varepsilon(M+1)/2$ and $\varepsilon(M-1)/2$, where M corresponds to the bit length of S. Thus, if $\Delta > 0$, we know that $x_i = 0$; otherwise, the averages of P_1 and P_0 is $\varepsilon(M-1)/2$ and $\varepsilon(M+1)/2$. Thus, if $\Delta < 0$ then $x_i = 1$. There should be a positive peak when $x_i = 0$ and a negative peak when $x_i = 1$.

B. Mounting the DPA on FPGA Platform

We perform the actual DPA attack on aforementioned pairing cryptoprocessor. The design is implemented on a customized FPGA board for power analysis.

We choose an x with $x_0 = 0$ and perform $(x - X_2)$ for 2000 times with 2000 random X_2. The respective power consumptions are stored in 2000 one dimensional vectors and we compute differential power consumption Δ. Then we accumulate the samples of Δ and plot it, which is depicted in Fig. 5(a). The positive peak ensures that $x_0 = 0$.

The same experiment has been repeated for another x having $x_0 = 1$ and plotted the respective difference-of-means in Fig. 5(b). In this case the expectation of $\langle P_1 - P_0 \rangle$ is negative and we achieve the result as expected. Above experimental result ensures that an attacker can easily mount the DPA on pairing computations over $\mathbb{F}_p$. After finding out the LSB, DPA can be performed for second LSB, and so on. The same power traces could be utilized for finding out all secret bits.

C. Proposed Counteracting Technique

The line function $l_{T,P}(x, y)$ is computed during the addition step of the Miller algorithm. In IBE scheme [38] P is replaced by U. The formula of $l_{T,P}(x, y)$ is $l_{T,U}(x, y) = (y - Y_2)Z_3 - (x - X_2)(Y_2 Z_1^3 - Y_1)$, where $T(X_1, Y_1, Z_1)$ is the intermediate result of doubling step and (X_3, Y_3, Z_3) represents the addition result of $T + U$. In this formula both public point $U = (X_2, Y_2)$ and private point $S_{ID} = (x, y)$ are used. The computation of $l_{T,U}(x, y)$ is the main weakness of pairing computation over $\mathbb{F}_p$ against side-channel attacks. The main drawback of the above formula is that the public and private parameters are directly involved to perform an $\mathbb{F}_p$ operation. The side-channel attack thus exploit the respective $\mathbb{F}_p$ operation for finding out the secret bits by manipulating public parameter U. To counteract against side-channel attacks it could be computed as $l_{T,U}(x, y) = (X_2(Y_2 Z_1^3 - Y_1) - Y_2 Z_3) - (Y_2 Z_1^3 - Y_1)x + Z_3 \cdot y$.

This computation technique have no $\mathbb{F}_p$-primitive which consists of one public parameter and one private parameter which defend DPA attacks. This countermeasure requires $182m$, $108m$, and $32m$ overhead operations for each Tate, ate, and optimal-ate pairings to defend DPA attacks [2].

TABLE II
IMPLEMENTATION RESULT OF PAIRING CRYPTOPROCESSOR

Pairings	Slice	LUT	FF	Freq. $[MHz]$	Cycles	Times $[ms]$
Tate	23 k	87 k	27 k	145	1,730,000	11.93
ate					1,207,000	8.32
optimal-ate					821,000	5.66

TABLE III
HARDWARE AND SOFTWARE IMPLEMENTATIONS OF PAIRING OVER BN-CURVES

Ref.	Platform	Type	Freq. [MHz]	Area	Cycles	Times [ms]
Our	Virtex-6	Tate	145	23k slice	1,730,000	11.93
		ate			1,206,000	8.32
		opt-ate			821,000	5.66
[1]§	Virtex-6	ate	210	4k slice 42 DSP	336,366	1.60
		opt-ate			245,430	1.17
[9]	130 nm CMOS	Tate	338	97k gate	11,627,200	34.40
		ate			7,706,400	22.80
		opt-ate			5,340,400	15.80
[10]§	130 nm CMOS	ate	204	183k gate	861,724	4.20
		opt-ate			592,976	2.90
[3]	core 2	opt-ate	2394	-	4,470,408	1.86
[8]	core i7	opt-ate	2800	-	2,330,000	0.83
[18]	P-4 core 2	ate	2400	-	15,000,000	6.25
		opt-ate			10,000,000	4.17
[16]	core 2	ate	2400	-	14,429,439	6.01

§ Implementation specifically for BN-curves with fixed parameters.

VII. IMPLEMENTATION RESULTS

The whole design has been done in Verilog (HDL) on Xilinx ISE Design Suit using a Virtex-6 xc6vhx250t-3ff1154 FPGA. The design can run at a maximum frequency of 145 MHz. The pairing hardware uses $23k$ logic slices including controllers and data access unit. It uses $27k$ flip flops for registers. It finishes one Tate, ate, and optimal-ate pairing computations in 11.93, 8.32, and 5.66 ms. Table II shows the implementation results.

Performances are compared with actual implementations of cryptographic pairings on software and dedicated hardware achieving a 128-bit security level. Table III gives a performance comparison of related implementations.

Due to the parallel structure our design computes six $\mathbb{F}_p$ multiplications in parallel which are completed in 256 cycles. The main features that strengthen the proposed cryptoprocessor for pairing computations are as follows.

- Adopted parallelism and efficient use of $\mathbb{F}_{p^k}$ arithmetic cores reduce the total cycles by 1/6 from [9].
- Due to the inherent properties the frequency of a design in FPGA is much lower than that in ASIC (CMOS standard cell). However, the speed of the pairing cryptoprocessor is comparable to the CMOS standard cell design.
- The proposed design is flexible w.r.t. curve parameters.

The underlying platform plays a crucial role in determining the performance of a design. Thus, existing designs on different platforms does not lead to a fair comparison. The cycles required to compute pairings on different designs may be considered such a platform independent parameter. With respect to clock cycles count, the present design is the best design providing flexibility to work with any curve parameters.

VIII. Conclusion

In this paper we explored the inherent FPGA features for designing efficient $\mathbb{F}_p$-primitives, based on which the paper further proposed a dual-core cryptoprocessor for computing pairings over BN curves. The proposed design can be programmed for any curve parameters. The paper further analyzed the effect of DPA attack on the pairing cryptoprocessor.

References

[1] J. Fan, F. Vercauteren, and I. Verbauwhede, "Efficient hardware implementation of $\mathbb{F}_p$—arithmetic for pairing-friendly curves," *IEEE Trans. Computers* [Online]. Available: http://dx.doi.org/10.1109/TC.2011.78

[2] S. Ghosh and D. Roychowdhury, "Security of prime field pairing cryptoprocessor against differential power attack," in *InfoSecHiComNet, LNCS 7011*, 2011, pp. 16–29.

[3] M. Naehrig, R. Niederhagen, and P. Schwabe, "New software speed records for cryptographic pairings," Cryptology ePrint Archive, Tech. Rep. 2010/186, 2010. [Online]. Available: http://eprint.iacr.org

[4] F. Vercauteren, "Optimal pairings," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 455–461, Jan. 2010.

[5] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "High speed flexible pairing cryptoprocessor on FPGA platform," *Pairing'10, LNCS*, vol. 6487, pp. 450–466, 2010.

[6] R. Granger and M. Scott, "Faster squaring in the cyclotomic subgroup of sixth degree extensions," *PKC '10, LNCS 6056*, pp. 209–223, 2010.

[7] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, "High speed $\mathbb{F}_p$ multipliers and adders on FPGA platform," presented at the DASIP'10, Scotland, U.K., 2010.

[8] J. L. Beuchat, J. E. G. Díaz, S. Mitsunari, E. Okamoto, F. R. Henríquez, and T. Teruya, "High-speed software implementation of the optimal-ate pairing over Barreto-Naehrig curves," Cryptology ePrint Archive, Tech. Rep. 2010/354, 2010. [Online]. Available: http://eprint.iacr.org/.

[9] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar, "Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves," *CHES '09, LNCS 5747*, pp. 254–271, 2009.

[10] J. Fan, F. Vercauteren, and I. Verbauwhede, "Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto-Naehrig curves," *CHES '09, LNCS 5747*, pp. 240–253, 2009.

[11] M. Scott, N. Benger, M. Charlemagne, L. J. D. Perez, and E. J. Kachisa, "On the final exponentiation for calculating pairings on ordinary elliptic curves," *Pairing '09, LNCS 5671*, pp. 78–88, 2009.

[12] E. Lee, H. S. Lee, and C. M. Park, "Efficient and generalized pairing computation on Abelian varieties," Cryptology ePrint Archive, Tech. Rep. 2008/040, 2008. [Online]. Available: http://eprint.iacr.org/

[13] J. Hoffstein, J. Pipher, and J. H. Silverman, *An Introduction to Mathmatical Cryptography*. New York: Springer, 2008.

[14] M. Naehrig, P. S. L. M. Barreto, and P. Schwabe, "On compressible pairings and their computation," *AFRICACRYPT'08, LNCS 5023*, pp. 371–388, 2008.

[15] A. Barenghi, G. Bertoni, L. Breveglieri, and G. Pelosi, "A FPGA coprocessor for the cryptographic Tate pairing over $\mathbb{F}_p$," in *Proc. 5th Int. Conf. Inform. Technol.: New Generations (ITNG)*, 2008, pp. 112–119.

[16] P. Grabher, J. Großschädl, and D. Page, "On software parallel implementation of cryptographic pairings," *SAC '08. LNCS 5381*, pp. 35–50, 2008.

[17] C. Rebeiro and D. Mukhopadhyay, "High speed compact elliptic curve cryptoprocessor for FPGA platforms," *Indocrypt'08, LNCS 5365*, pp. 376–388, 2008.

[18] D. Hankerson, A. Menezes, and M. Scott, *Software Implementation of Pairings. In Identity-Based Cryptography*, M. Joye and G. Neven, Eds. Amsterdam, The Netherlands: IOS Press, 2008.

[19] T. H. Kim, T. Takagi, D. G. Han, H. Kim, and J. Lim, "Power analysis attacks and countermeasures on ηT pairing over binary fields," *ETRI J.*, vol. 30, no. 1, pp. 68–80, 2008.

[20] P. S. L. M. Barreto, S. D. Galbraith, C. ÓhÉigeartaigh, and M. Scott, "Efficient pairing computation on supersingular abelian varieties," *Designs, Codes, Cryptography*, vol. 42, pp. 239–271, 2007.

[21] A. J. Devegili, M. Scott, and R. Dahab, "Implementing cryptographic pairings over Barreto-Naehrig curves," *Pairing '07. LNCS 4575*, pp. 197–207, 2007.

[22] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *SAC'05 LNCS 3897*, 2006, pp. 319–331.

[23] F. Hess, N. P. Smart, and F. Vercauteren, "The eta pairing revisited," *IEEE Trans. Inform. Theory*, vol. 52, no. 10, pp. 4595–4602, Oct. 2006.

[24] A. Devegili, C. ÓhÉigeartaigh, M. Scott, and R. Dahab, "Multiplication and squaring on pairing-friendly fields," Cryptology ePrint Archive, Tech. Rep. 2006/471, 2006.

[25] D. N. Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler, "Efficient hardware architectures for modular multiplication on FPGAs," in *Proc. Int. Conf. Field Program. Logic Appl.*, 2005, pp. 539–542.

[26] S. Chatterjee, P. Sarkar, and R. Barua, "Efficient computation of Tate pairing in projective coordinate over general characteristic fields," *ICISC 2004, LNCS 3506*, pp. 168–181, 2005.

[27] S. Galbraith, "Pairings," in *Advances in elliptic curve cryptography, London Mathematical Society Lecture Note Series*, I. F. Blake, G. Seroussi, and N. P. Smart, Eds. Cambridge, MA: Cambridge University Press, 2005, ch. IX.

[28] D. Page and F. Vercauteren, "Fault and side-channel attacks on pairing based cryptography," Cryptology ePrint Archive, Tech. Rep. 2004/283, 2004. [Online]. Available: http://eprint.iacr.org/.

[29] R. Dutta, R. Barua, and P. Sarkar, "Pairing-based cryptographic protocols: A survey," Cryptology ePrint Archive, Tech. Rep. 2004/64, 2004. [Online]. Available: http://eprint.iacr.org

[30] V. S. Miller, "The weil pairing, and its efficient calculation," *J. Cryptology*, vol. 17, pp. 235–261, 2004.

[31] V. Bunimov and M. Schimmler, "Area and time efficient modular multiplication of large integers," in *Proc. ASAP*, 2003, pp. 400–409.

[32] I. Duursma and H. Lee, "Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$," in *ASIACRYPT 2003, LNCS 2894*, 2003, pp. 111–123.

[33] M. Joye and S. M. Yen, "The Montgomery powering ladder," in *CHES '02, LNCS 2523*, 2003, pp. 291–302.

[34] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing–based cryptosystems," in *CRYPTO '02, LNCS 2442*, 2002, pp. 354–368.

[35] S. D. Galbraith, K. Harrison, and D. Soldera, "Implementing the tate pairing," in *Proc. ANTS*, 2002, pp. 324–337.

[36] S. Mitsunari, R. Sakai, and M. Kasahara, "A new traitor tracing," *IEICE Trans. Fundam.*, vol. 2, pp. 481–484, 2002.

[37] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *CRYPTO 2001, LNCS 2139*, 2001, pp. 213–229.

[38] T. S. Messerges, "Using second-order power analysis to attack DPA resistant software," in *CHES 2000, LNCS 1965*, Berlin, Germany, 2000, pp. 238–251.

[39] A. Joux, "A one round protocol for tripartite diffie–hellman," in *Proc. ANTS*, 2000, pp. 385–394.

[40] R. Sakai, K. Ohgishi, and M. Kasahara, "Cryptosystems based on pairing," in *Proc. SCIS*, 2000, pp. 26–28.

[41] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Adv. Cryptology—CRYPTO'99, LNCS 1666*, 1999, pp. 388–397.

[42] S. Hauck, M. M. Hosler, and T. W. Fry, "High-performance carry chains for FPGAs," in *Proc. FPGA*, 1998, pp. 223–233.

[43] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, RSA, DSS and other systems," in *Adv. Cryptology—CRYPTO'96, LNCS 1109*, 1996, pp. 104–113.

[44] G. R. Blakley, "A computer algorithm for calculating the product A*B modulo M," *IEEE Trans. Comput.*, vol. C-32, no. 5, pp. 497–500, May 1983.

Santosh Ghosh received the Ph.D. degree from the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India, in 2011.

Currently he is a Post-doctorate Researcher with COSIC/ESAT, Katholieke Universiteit Leuven, Leuven, Belgium.

Debdeep Mukhopadhyay received the Ph.D. degree from Indian Institute of Technology (IIT), Kharagpur, in 2007.

He is currently an Assistant Professor with the Computer Science and Engineering Department, IIT.

Dipanwita Roy Chowdhury received the Ph.D. degree from Indian Institute of Technology (IIT), Kharagpur, in 1994.

Currently, she is a Professor with the Department of Computer Science and Engineering, IIT.