HYBRID MASKED KARATSUBA MULTIPLIER FOR $GF(2^{233})$

Chester Rebeiro[1] and Debdeep Mukhopadhyay[2]

Abstract

The paper presents a detailed study on the implementation of Karatsuba Multiplier for GF (2^{233}), which is a state of the art field for secured Elliptic Curve Cryptography, according to NIST. The work suggests the trade-offs involved in designing this important class of multiplier and proposes a Hybrid Karatsuba Multiplier which requires least amount of space on a FPGA. The work also shows a novel masking technique to prevent the multiplier from power based side-channel attacks. Comparison shows that the proposed scheme requires lesser number of gates compared with the conventional approach.

Keywords: Karatsuba Multiplier, Elliptic Curve Cryptography, Masked Multiplication

1. Introduction

Elliptic Curve Cryptography (ECC) was invented independently by Koblitz and Miller in 1985. Since then, the security and efficiency of ECC has been proven, and ECC has been incorporated in several security standards. ECC offers more security per key bit than any other public key cryptosystem. This has prompted the U.S. National Security Agency to move to ECC based public key cryptography. For a given level of security, the size of the key and the operations involved in ECC computation is much shorter than other crypto algorithms. This makes ECC an attractive alternative for today's hand held devices where processing bandwidth, memory resources and power are limited.

NIST's standard for Digital Signatures [1] recommends using a prime field, $GF(p)$, or a binary extension field $GF(2^m)$ for Elliptic Curves. Binary Extension Fields have the advantage that field additions can be performed by *XOR* operations, therefore no carry is involved. This leads to implementations that require lesser area and have better performance. The NIST standard recommends binary extension curves of degree 163, 233, 283, 409 and 571. For our work on the implementation of Elliptic Curve Cryptosystem on a FPGA, we have selected the field $GF(2^{233})$ as there have been very few published works [3] for implementations specific to this field.

Implementation of Elliptic Curve Cryptosystems follows a layered hierarchical scheme as shown in Figure (1). The performance of the top layers of the hierarchy is greatly influenced by the performance of the underlying layers. It is therefore important to have efficient implementations of finite field operations such as additions, multiplications and inversion.

There are several methods to implement the finite field multiplication such

1. M.S. Student, Computer Sc. and Engg., IIT Madras. (chetrebeiro@gmail.com)
2. Asst. Professor, Computer Sc. and Engg., IIT Madras (debdeep@cse.iitm.ernet.in)

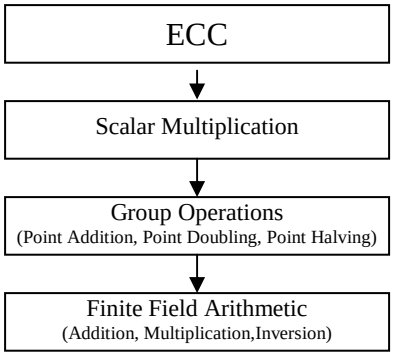

Figure 1: *Layered Hierarchical Model for ECC*

as the Classical Method, the Karatsuba Multiplier, Massey-Omura Multiplier and the Sunar-Koς Multiplier. Of these the Karatsuba Multiplication algorithm is the most efficient. However, the recursive Karatsuba algorithm cannot be applied when the number of bits in the multiplicands (n) is not a power of two, i.e. $n \neq 2^k$ .For multiplications in such fields, a combination of the Karatsuba multiplier and another multiplication algorithm is generally used. In [3] for example, a multiplier was designed for $GF(2^{233})$ using a combination of a Karatsuba multiplier and a classical multiplier. These techniques are not the best because of the inefficiency of the classical multipliers. There have been few techniques proposed for a generic recursive Karatsuba multiplier which can handle multiplications for any value of n [4][5]. In this paper, we propose a new hybrid technique to implement the Karatsuba multiplier for the field GF(2^{233}) . This technique results in most efficient designs on an FPGA. We also propose a masking technique for the Karatsuba multiplier to prevent Differential Side Channel Attacks [6]. We show that the proposed masked multiplier requires lesser number of multiplications compared with the conventional masked multiplier.

In Section 2 of the paper, we present the preliminaries of the Karatsuba algorithm for Finite Field Multiplication and the Masked Multiplication. In section 3 we discuss, analyze and compare the various techniques for Karatsuba multiplication in the field $GF(2^{233})$. In section 4 we present our Hybrid Karatsuba Algorithm and compare its performance with the other Karatsuba multiplier for 233 bit multiplications. Section 5 presents our masked multiplier design, and compares it with the conventional masked multiplier. Section 6 has the conclusion of the paper.

2. Preliminaries

In the present section we present a background on the topic of finite field multiplication and the Karatsuba method to efficiently multiply. We also briefly state about Side Channel Attacks on cryptographic algorithms exploiting statistics based on power consumption of the underlying multipliers.

2.1. Finite Field Multiplication and the Karatsuba Method

The finite field multiplication of two elements in the field $GF(2^n)$ is defined as

$$C(x) = A(x)B(x) \bmod P(x) \quad (1)$$

where, $A(x)$, $B(x)$ and $C(x) \in GF(2^n)$, and $P(x)$ is the irreducible polynomial of degree n which generates the field $GF(2^n)$. Implementing the multiplication requires two steps. First, the polynomial product $C'(x) = A(x)B(x)$ is determined then, the modulo operation is done on $C'(x)$. The Karatsuba algorithm is used for the polynomial multiplication. The Karatsuba multiplier achieves its efficiency by splitting the n bit multiplicands into two 2-term polynomials as shown in equation (2). The multiplication is then done using three $n/2$ bit multiplications.

$$\begin{aligned} C'(x) &= (A_h x^{n/2} + A_l)(B_h x^{n/2} + B_l) \\ &= A_h B_h x^n + (A_h B_l + A_l B_h) x^{n/2} + A_l B_l \\ &= A_h B_h x^n + ((A_h + A_l)(B_h + B_l) + A_h B_h + A_l B_l) x^{n/2} + A_l B_l \end{aligned} \quad (2)$$

When the multiplier operates in the field $GF(2^n)$ the basic Karatsuba multiplier may be applied recursively to amplify the gains. Most efficient multiplications are obtained when a fully recursive Karatsuba multiplier is used. This is possible only when the finite field has the form $GF(2^n)$. Such a field would require k iterations of the Karatsuba algorithm. The number of gates required for a fully recursive Karatsuba multiplier is given below.

$$\#AND \text{ gates}: \quad n^{\log_2 3}$$

$$\#XOR \text{ gates}: \quad \sum_{r=0}^{\log_2 n} 3^r \left(\frac{4n}{2^r} - 4\right)$$

2.2. Side Channel Resistant Multiplication

An important aspect of the finite field multiplier used for ECC cryptosystems is that they have to be resistant to Side Channel Attacks (SCAs). In these attacks knowledge is gathered about the key by exploiting information that leaks from the device. Information can leak from various sources in the device. However side channel analysis of power consumption of the device is the most researched. There are two techniques for power analysis: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). DPA exploits the fact that power consumption of a chip depends on intermediate results of the algorithm. The most common technique to counter DPA in multipliers is by using masking [8]. The main idea behind a masked multiplier is to make all intermediate values of the multiplier independent of the multiplicands. Such multipliers are secure against SCAs, if the underlying CMOS gates switch once per clock cycle.

3. Design Exploration for the 233 bit Karatsuba Multiplier

In the present section we explore various techniques to implement the 233 bit Karatsuba multiplier. We explore techniques like Padding, Binary, Simple, Generalized and the proposed Hybrid Karatsuba Multiplier. The design trade-offs involved in the various architectures are reported.

3.1. The Padded Karatsuba Multiplier

The Padded Karatsuba multiplier is the most simple method of implementing a fully recursive Karatsuba multiplier for a field $GF(2^n)$, where $n = 2^k + d$ and k is the largest integer such that $2^k < n$. The Padded Karatsuba multiplier extends the n bit multiplicands to 2^{k+1} bits by padding its most significant bits with $2^{k+1} - n$ zeroes. This then allows the use of the basic recursive Karatsuba

algorithm. The obvious drawback of this method is the extra arithmetic introduced due to the padding.

3.2. The Binary Karatsuba Multiplier

The Binary Karatsuba multiplier was proposed in [4]. The algorithm modifies the basic Karatsuba multiplier to handle any field of the form $GF(2^n)$, where $n = 2^k + d$, and k is the largest integer such that $2^k < n$. The algorithm splits each multiplicand into two terms, the higher term containing d bits and the lower term containing 2^k bits. The higher term partial product ($A_h B_h$) is determined by a Binary Karatsuba algorithm for d bits. The number of times the Binary Karatsuba algorithm is called recursively depends on the hamming weight of n. For example, the binary equivalent of 233 is $(11101001)_2$, therefore the Binary Karatsuba algorithm is used recursively for 5 iterations.

3.3. The Simple Karatsuba Multiplier

The Simple Karatsuba multiplier is the basic recursive Karatsuba multiplier with a small modification. If an n bit multiplication is needed to be done, n being any integer, it is split into two polynomials as in equation (2). The A_l and B_l polynomials have $\lceil n/2 \rceil$ terms while the A_h and B_h polynomials have $\lfloor n/2 \rfloor$ terms. The Karatsuba multiplication can then be done with two $\lceil n/2 \rceil$ bit multiplications and a single $\lfloor n/2 \rfloor$ bit multiplication.

The higher bound for the number of *AND* gates and *XOR* gates required for the Simple Karatsuba multiplier is the same as that of a $2^{\lceil \log_2 n \rceil}$ bit basic Karatsuba multiplier. The Simple Karatsuba multiplier requires at most one bit padding (for the $(A_h + A_l)(B_h + B_l)$ multiplication). It therefore requires lesser gates for implementation as compared with the Binary Karatsuba multiplier.

For a n bit multiplication, the number of times the Simple Karatsuba multiplier would be used recursively is $\lceil \log_2 n \rceil$. This is higher then the Binary Karatsuba multiplier which would be used recursively for $\lfloor \log_2 n \rfloor$ times. Therefore the delay in the Simple Karatsuba is expected to be higher than that of a Binary Karatsuba algorithm. The results presented in Table (1) agree with the above expected results.

3.4. The General Karatsuba Multiplier

The basic Karatsuba multiplier defines a method to multiply two n bit polynomials using three $n/2$ bit multipliers. This is achieved by splitting the n bit polynomial into a 2-term polynomial with each term having $n/2$ bits. In [5] it was shown that if A and B are two $n = 3k$ bit polynomials, the Karatsuba multiplier for 3-term polynomials can be used as shown in equation (4). This results in six multiplications and 13 additions.

$$
\begin{aligned}
C &= AB \\
&= (A_2 x^{2n/3} + A_1 x^{n/3} + A_0)(B_2 x^{2n/3} + B_1 x^{n/3} + B_0) \\
&= A_2 B_2 x^{4n/3} + (A_2 B_1 + A_1 B_2) x^n \\
&\quad + (A_2 B_0 + A_0 B_2 + A_1 B_1) x^{2n/3} + (A_1 B_0 + A_0 B_1) x + A_0 B_0
\end{aligned}
$$

$$
\begin{aligned}
&= A_2B_2x^{4n/3} + ((A_2 + A_1)(B_2 + B_1) + A_2B_2 + A_1B_1)x^n \\
&\quad + (((A_2 + A_0)(B_2 + B_0) + A_2B_2 + A_1B_1 + A_0B_0)x^{2n/3} \\
&\quad + ((A_1 + A_0)(B_1 + B_0) + A_1B_1 + A_0B_0)x^{n/3} \quad + A_0B_0
\end{aligned} \tag{4}
$$

The general formula for multiplying two m-term polynomials is given in equation (5). In the equations $D_i = A_iB_i$ and $D_{s,t} = (A_s + A_t)(B_s + B_t)$, and C_0 , C_{2n-2} , C_i are coefficients of C .

$$
\begin{aligned}
C_0 &= D_0 \\
C_{2n-2} &= D_{n-1} \\
C_i &= \sum_{\substack{s+t=i \\ t>s\geq 0}} D_{s,t} + \sum_{\substack{s+t=i \\ n>t>s\geq 0}} (D_s + D_t) \qquad \textit{for odd } i \quad 0 < i < 2n-2 \\
C_i &= \sum_{\substack{s+t=i \\ t>s\geq 0}} D_{s,t} + \sum_{\substack{s+t=i \\ n>t>s\geq 0}} (D_s + D_t) + D_{i/2} \qquad \textit{for even } i \quad 0 < i < 2n-2
\end{aligned} \tag{5}
$$

To apply the Binary Karatsuba multiplier recursively: let n be a composite number (if n is prime we pad it by one bit) with the prime factors in increasing order being $\{p_1, p_2, p_3 \cdots\}$ .To multiply two n bit numbers, we first do the p_1 term Karatsuba. Each term is of n/p_1 bits. The n/p_1 term multiplication is done using p_2 term Karatsuba. The p_1/p_2 term multiplication is done using the p_3 term Karatsuba and so on.

4. Hybrid Multiplier

This subsection presents our proposed hybrid multiplier. This design results in most efficient implementation of Karatsuba multiplication on a FPGA, in particular Xilinx FPGAs.

Table (1) shows that among the Padded, Binary, Simple and General Karatsuba multipliers, the General Karatsuba multiplier requires the lowest number of slices on the FPGA. This is ironic as the General Karatsuba algorithm requires the most number of *AND* and *XOR* gates. The reason for this is because of the granularity of the FPGA. Each slice in a Xilinx Virtex 4 FPGA [7] contains two function generators capable of implementing any arbitrary four input Boolean function. If two inputs are fed to the function generator instead of four, the function generator is not fully utilized. In the Padded, Binary and Simple Karatsuba implementations the smallest multiplication done is on two bits. This leaves several of the function generators under utilized. In a 233 bit General Karatsuba multiplier however, the smallest multiplication is a 13-term 13-bit multiplication. This has several operations that can be grouped in terms of four inputs (Equation 5). For example, to determine the value of C_{13} would require 20 additions, this would need only four slices on a FPGA. Therefore, the General Karatsuba multiplier obtains maximum utilization of the slices of the

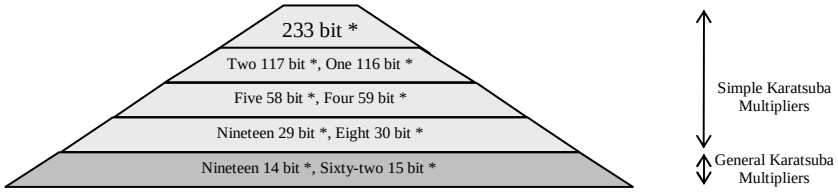

Figure 2: *Hybrid 233 bit Karatsuba Multiplier*

FPGA.

In our Hybrid Multiplier design we implement the initial recursions using the Simple Karatsuba multiplier. The final recursion is done using the General Karatsuba multiplier. The Simple Karatsuba multiplications reduce the gates required to the minimum, while the final recursion with the General Karatsuba multiplier maximizes the utilization of the function generator in each slice of the FPGA. For a 233 bit Hybrid Multiplier, (Figure 2) we do all the larger multiplications using the Simple Karatsuba algorithm. The smallest multiplications, i.e. 14-bit and 15-bit, are done using the General Karatsuba algorithm.

We now determine the upper bound for the number of gates required for a n bit Hybrid Karatsuba multiplier. Let $n' = 2^{\lceil \log_2 n \rceil}$ and let k be the number of recursions of Simple Karatsuba multiplication. The final recursion using the General Karatsuba algorithm is done with m ($m \le m' = n'/2^k$) bit multiplicands. The number of *AND* gates required for k recursions of Simple Karatsuba multiplication is 3^k. The number of *AND* gates required for a m' bit General Karatsuba multiplication is $m'(m'+1)/2$ [5].The upper bound for the total number of *AND* gates required for the n bit Hybrid Karatsuba Multiplication is given by

$$\#\,AND \text{ gates}: \quad 3^k m'(m'+1)/2$$

Similarly, k recursions of Simple Karatsuba multiplication require $\sum_{r=0}^{k} 3^r \left(\frac{4n}{2^r} - 4\right)$ *XOR* gates, and m' bit General Karatsuba multiplication require $(5/2)m'^2 - (7/2)m' + 1$ gates [5]. The upper bound for the total number of *XOR* gates required for the n bit Hybrid Karatsuba multiplication is given by

$$\#\,XOR \text{ gates}: \quad \left(\frac{5m'^2}{2} - \frac{7m'}{2} + 1\right)3^{k+1} + \sum_{r=0}^{k} 3^r \left(\frac{4n}{2^r} - 4\right)$$

4.1. Implementation results for 233 bit Karatsuba multiplier

Table (1) compares the gate requirements for the 233 bit Karatsuba multiplication algorithms. The third and forth column shows the number of slices taken and the maximum delay on a Xilinx Vertex 4 FPGA. The Hybrid Karatsuba implementation requires minimum number of slices on the FPGA, however the delay is the maximum. One method of decreasing the delay of the implementation is to implement the Hybrid algorithm using a combination of the Binary Karatsuba algorithm along with the General Karatsuba algorithm. If the same algorithms were to be implemented on an ASIC instead of an FPGA where the granularity is much lower, then the Simple Karatsuba implementation would take the least amount of resources. However it will have a longer delay as compared to the Binary Karatsuba multiplier. This is because the number of recursions of the algorithm is more in case of the Simple Karatsuba (8 recursions) compared to a Binary Karatsuba (7 recursions).

	# *AND*	# *XOR*	Slices	Delay
Padded Karatsuba	6561	37320	13067	14.625ns
Binary Karatsuba	6349	36028	12819	14.625ns
Simple Karatsuba	6292	34952	12569	15.743ns
General Karatsuba	9828	49832	11284	13.863ns
Hybrid Karatsuba	9435	47350	**10434**	16.100ns

Table 1: *Comparison of 233 bit Karatsuba Implementations*

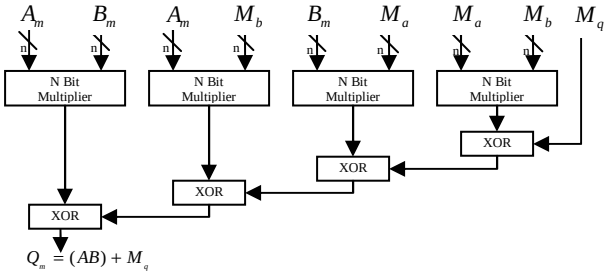

Figure 3: *Generic n bit Masked Multiplier*

5. Masked Karatsuba Multiplier

The most common approach for a masked multiplier design to prevent DPA attacks is as shown in Figure (3). The multiplicands A and B are masked with M_a and M_b respectively. The input to the masked multiplier is the masked value A_m and B_m, and the masks M_a, M_b and M_q. The output Q_m is the product of unmasked multiplicands and the mask M_q. If the length of A and B is n bits, then $4MUL_A(n)$ *AND* gates and $4MUL_X(n)+4(2n-1)$ *XOR* gates are required, where $MUL_A(n)$ and $MUL_x(n)$ is the number of *AND* gates and *XOR* gates required for a n bit multiplication. Our modified masked multiplier shown in Figure (4) splits the multiplicands into two as is done in the Karatsuba multiplier. We then use an $n/2$ bit mask M_a to mask both the A_h and A_l terms. Similarly, an $n/2$ bit mask M_b is used to mask the B_h and B_l terms (Equation 6). The output Q_m is the product of the two multiplicands masked with the mask M_q (Equation 7). It may be noted that each state in Equation (7) uses an additive mask, thus each computation masks the original values and hence the multiplier prevents attacks based on DPA.

$$\begin{aligned} A_h &= A_{mh} + M_a \qquad & A_l &= A_{ml} + M_a \\ B_h &= B_{mh} + M_b \qquad & B_l &= B_{ml} + M_b \end{aligned} \tag{6}$$

$$\begin{aligned} Q_m &= (A_h x^{n/2} + A_l)(B_h x^{n/2} + B_l) + M_q \\ &= ((A_{mh} + M_a)x^{n/2} + (A_{ml} + M_a))((B_{mh} + M_b)x^{n/2} + (B_{ml} + M_b)) + M_q \\ &= (A_{mh}B_{mh} + A_{mh}M_b + B_{mh}M_a + M_aM_b)x^n \\ &\quad + (A_{mh}B_{ml} + A_{mh}M_b + B_{ml}M_a + A_{ml}B_{mh} + A_{ml}M_b + B_{mh}M_a)x^{n/2} \\ &\quad + (A_{ml}B_{ml} + A_{ml}M_b + B_{ml}M_a + M_aM_b) + M_q \end{aligned} \tag{7}$$

The number of gates required reduces to $9MUL_A(n/2)$ *AND* gates and $9MUL_X(n/2)+9(n-1)+ADD_{pp}$. $MUL_A(n/2)$ and $MUL_X(n/2)$ is the number of *AND* gates and *XOR* gates required for an $n/2$ bit multiplication respectively. ADD_{PP} is the number of *XOR* gates required for the final $2n-1$ bit concatenation and *XOR*. This method of generating a masked multiplier can be incorporated with Karatsuba multipliers wherever the first recursion uses $n/2$ bit multiplications. For example, a 233 bit masked multiplier can use Padded, General, Simple or the proposed Hybrid Karatsuba multiplications. It cannot use the Binary Karatsuba multiplier because the first split in the multiplicands is not of same size (i.e 105 bits and 128 bits). Table (2) compares the gate

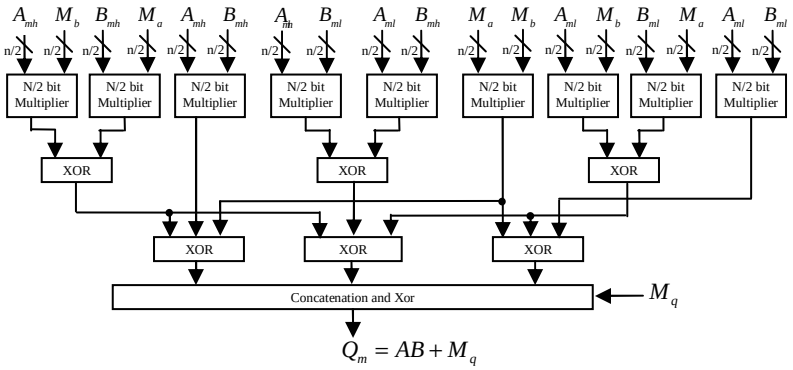

Figure 4: *Proposed n-bit Masked Multiplier*

requirements for the generic masked multiplier with the proposed masked multiplier for various multiplication algorithms. Results show that the proposed masked multiplier requires lesser gates than the generic masked multiplier. It may be noted that the masked multiplier with the Hybrid Karatsuba implementation requires minimum number of slices. On a Virtex 4 FPGA, the Hybrid Generic Masked Multiplier required 41107 slices, while the Hybrid Proposed Masked Multiplier required 30435 slices. The delay of the two designs was almost equal (around 17ns).

	Generic Masked Multiplier		**Proposed Masked Multiplier**	
	# AND	*# XOR*	*# AND*	*# XOR*
Padded Karatsuba	26244	151140	19683	112217
Simple Karatsuba	25168	141668	18873	103937
General Karatsuba	39312	201188	29484	149612
Hybrid Karatsuba	37740	191940	28485	143937

Table 2: *Comparison of 233 bit Masked Multiplier Implementations*

6. Conclusion

In this paper we proposed a novel Hybrid Karatsuba multiplier which uses the best of the Simple and the General Karatsuba algorithms. This resulted in lesser space requirements on a FPGA. We also proposed a new design for a masked multiplier based on the Karatsuba algorithm which requires lesser number of gates compared to the conventional masked multiplier. This implementation of finite field multiplication forms an ideal base for an elliptic curve cryptosystem over $GF(2^{233})$.

References

[1] U.S. Department of Commerce/National Institute of Standards and Technology, *Digital Signature Standards*, Federal Information Processing Standards Publication 186.

[2] C. Paar, *Efficient VLSI Architectures for Bit Parallel Computations in Galois Fields*. PhD thesis, Universtät GH Essen, VDI Verlag, 1994.

[3] C. Grabbe, et.al, *FPGA Designs of Parallel High Performance GF(2^{233}) Multipliers*, Proceedings of the 2003 International Symposium on Circuits and Systems, ISCAS'03, 2003.

[4] F. R. Henriquez, et. al., *On Fully Parallel Karatsuba Multipliers for GF(2^m)*, Proceedings of the International Conference on Computer Science and Technology, CST 2003, 2003.

[5] A. Weimerskirch, et. al, *Generalizations of the Karatsuba Algorithm for Efficient Implementations*, http://eprint.iacr.org/2006/224.pdf, 2006.

[6] P. Kocher, et. al, *Differential Power Analysis*, Advances in Cryptology-CRYPTO'99, 1999.

[7] Xilinx, *Virtex 4 User Guide (Chapter 5 : Configurable Logic Blocks)*, 2007.

[8] J. D. Golic, *Techniques for Random Masking in Hardware*, http:// eprint.iacr.org/2005/026.ps