TÜBİTAK

Turk J Elec Eng & Comp Sci, Vol.18, No.1, 2010, © TÜBİTAK
doi:10.3906/elk-0904-4

Differential power analysis resistant hardware implementation of the RSA cryptosystem

Keklik ALPTEKİN BAYAM[1], Berna ÖRS[2]
[1]*STMicroelectronics, Inc., Datastorage Division, 1060 E Brokaw Road, San Jose, CA 95131 USA*
e-mail: keklik.alptekin@st.com
[2]*İstanbul Technical University, Faculty of Electrical and Electronics Engineering, Maslak, İstanbul-TURKEY*
e-mail: siddika.ors@itu.edu.tr

Abstract

In this paper, RSA cryptosystem was implemented on hardware, then modified to be resistant against Differential Power Analysis attacks by using the Randomized Table Window method. This is the first FPGA realization of an algorithmic countermeasure which makes RSA resistant to power analysis attacks. Modular exponentiation is realized with Montgomery Modular Multiplication. The Montgomery modular multiplier has been realized with Carry-Save Adders. Carry-Save representation has been used throughout the RSA encryption algorithm. The primarily implemented RSA architecture prevents the extraction of the secret key using Simple Power Analysis attacks. When comparing the protected implementation with the unprotected, it can be seen that the total time has increased by 24.2%, while the throughput has decreased by 19.5%.

Key Words: *RSA, Montgomery Modular Multiplier, Carry Save Adder, Side-Channel Attacks, Differential Power Analysis Attack, Randomized Table Window Method.*

1. Introduction

RSA is a widely used public-key cryptosystem. RSA encryption is a one-way function, in that it is not possible to reverse without knowing the private key [1]. RSA is realized with large operands, such that the key length and the operands are greater than or equal to 512 bits. The encryption and decryption in an RSA cryptosystem is modular exponentiation: $M^E \bmod N$. To achieve efficiency and speed, RSA is best implemented in hardware [2].

In this paper, a hardware architecture of the RSA cryptosystem is proposed and implemented using Xilinx FPGA hardware. In this implementation a Montgomery Modular Multiplier (MMM) [3] with Carry Save Adder [4] based logic and representation has been used to speed up the calculations.

Side-channel attacks [5] derive information retrieved from the device, but is neither plaintext nor ciphertext. Power Analysis (PA) attacks [5] are a type of passive side-channel attacks, in which the power consumption

of the circuit is measured while the device is performing encryption or decryption. The private key or information about the private key is retrieved after an analysis of the measurement data. PA attacks have two types: Simple Power Analysis (SPA) attacks and Differential Power Analysis (DPA) attacks [6]. SPA attacks require a single measurement, while DPA attacks require many measurements followed by a statistical analysis to retrieve information about the private key.

There are hardware and algorithmic countermeasures against PA attacks. Itoh et al. have proposed an algorithmic countermeasure, knowns as the Randomized Table Window Method (RT-WM), against DPA attacks in [7].

The first implementation in this work prevents the extraction of the private key itself, though it cannot prevent the leakage of the Hamming weight information of the private key during an SPA attack. Protection against SPA attacks comes from the architectural design of the circuit but is unprotected against DPA attacks. In the second implementation in this work, RT-WM algorithm [7] has been implemented on top of the former unprotected implementation.

This paper presents a hardware implementation of the RSA cryptosystem resistant to differential power analysis. Section 2 and Section 3 explain the mathematical background, and the fundamentals of RSA and MMM architectures, respectively. This section is the basis to the architectural choices in the implementation. Section 4 presents the basics of side-channel attacks and gives detail about power analysis attacks and the countermeasures against them. Section 5 explains the unprotected implementation of the RSA cryptosystem. Section 6 investigates a DPA resistant implementation. Review of the present work and conclusions are given in Section 7.

2. The RSA cryptosystem

The RSA cryptosystem was developed by Rivest, Shamir, and Adleman in 1977 [1]. RSA is a public-key cryptosystem that serves both for encryption-decryption and digital signature.

RSA encryption and decryption are performed by modular exponentiation as $C = M^E \bmod N$ and $M = C^D \bmod N$, respectively, where M is the plaintext, C is the ciphertext, N and E are the public keys, D is the private key and $C, M, E, D \in \{0, 1, \ldots, N-1\}$ [1].

2.1. The m-ary method

The m-ary method reduces the number of multiplications processed in an exponentiation [8]. The exponent E is scanned r-bits at a time, where $m = 2^r$ and $sr = k$, where k is the bit length of E. Preprocessing is necessary for the exponentiation process, in which the powers of M mod N from 2 to $m-1$ are calculated [8, 2]. The m-ary method is given in Algorithm 1.

2.2. The sliding window technique

In the m-ary method, a zero word makes us skip the multiplication. In order to increase the number of skipped operations and reduce the number of total operations executed, the sliding window technique has been suggested by Bos and Coster and Knuth in [9, 8]. A sliding window exponentiation algorithm decomposes E into zero and nonzero words, which are called windows. In this technique, nonzero words cannot end with 0. Therefore

Algorithm 1 The m-ary Method: left to right.

Require: $N = (n_{k-1}, \ldots, n_1, n_0)_2$, $E = (e_{k-1}, \ldots, e_1, e_0)_2$, $M = (m_{k-1}, \ldots, m_1, m_0)_2$
Ensure: $C = M^E \bmod N$
1: Compute and store $M^w \bmod N$ for $w = 2.3, \ldots, m-1$.
2: Decompose E into r-bit words F_i for $i = 0, 1, \ldots, s-1$, $sr = k$
3: $C = M^{F_{s-1}} \bmod N$
4: **for** i from $s-2$ downto 0 **do**
5: $C = CC^{2^r} \bmod N$
6: **if** $F_i \neq 0$ **then**
7: $C = CM^{F_i} \bmod N$
8: **end if**
9: **end for**

the multiplications in the preprocessing step are only done to evaluate the odd numbers: $3, 5, 7, \ldots, m-1$. The preprocessing multiplications are almost halved.

The analysis performed in [10] shows that the sliding window technique proposed in [9] requires 5–8% fewer multiplications than the m-ary method, namely 6.37% for 512-bit key length.

3. Montgomery modular multiplication

In 1985 Montgomery introduced a new method for modular multiplication [3]. Montgomery's approach avoids the time consuming trial division that is a bottleneck for most other algorithms.

The Montgomery algorithm computes the result by replacing the division operation with k times division by a power of 2, where a, b and N are k-bit binary numbers. Montgomery multiplication is defined as $R = a'b'r^{-1} \bmod N$, where $r = 2^k$ and the real multiplicands a and b need to be transformed into their N-residues such as $a' = ar \bmod N$.

We need a post-processing, where R' and 1 are the multiplicands of the Montgomery Multiplication as $R = (abr)\,1r^{-1} \bmod N = ab \bmod N$.

In our implementation, we use the Montgomery Multiplication algorithm with no final subtraction, as given in Algorithm 2 [11, 12].

Algorithm 2 Montgomery Modular Multiplication with No Final Subtraction (*MonPro_NFS*).

Require: $N = (n_{k-1}, \ldots, n_1, n_0)_2$, $X = (x_k, \ldots, x_1, x_0)_2$, $Y = (y_k, \ldots, y_1, y_0)_2$, $r = 2^{k+2} \bmod N$, $n_0 = 1$.
Ensure: $T = \text{MonPro\_NFS}(X, Y, N) = XYr^{-1} \bmod N$
1: $T = 0$
2: **for** i from 0 to $k+1$ **do**
3: **if** $(T + x_iY)$ is even **then**
4: $T = (T + x_iY)/2$
5: **else**
6: $T = (T + x_iY + N)/2$
7: **end if**
8: **end for**

When the exponentiation operation uses Montgomery Multiplication Algorithm, it needs preprocessing, where the N residue of the base number is calculated; and a post-processing, where the result transferred from

the N residue to normal representation. A constant number has to be calculated for the preprocessing to evaluate the N residue of the plaintext. This constant number is $2^{2k+4} \bmod N$ when using *MonPro_NFS*.

Adders are necessary to realize the Montgomery multiplication, namely for steps 4 and 6 of Algorithm 2. Carry save addition is suitable, especially for large operands [4]. It is an appropriate way of reducing 3 k-bit operands to 2 k-bit operands. The result is in carry save representation (C,S).

One final addition has to be performed to reduce the result from 2 k-bit operands to 1 k-bit operand—to convert back to normal number representation. In this work, Carry Ripple Pipelined Adder (CRPA) has been used as for this operation.

A Carry Ripple Pipelined Adder (CRPA) processes k-bit operands word by word by in k/w clock cycles using w-bit carry ripple adders (CRAs).

4. Side-channel attacks

In cryptography, an attack based on side channel information is called a "side-channel attack." Side-channel information is the information that can be retrieved from the cryptographic device that is neither the plaintext nor the ciphertext [5]. Active attacks, also referred as tampering attacks, require access to the internal circuitry of the attacked device [5]. There are two types: Probing attack [13], and Fault induction attack [14, 15].

In passive attacks, the effects of the processing device are measured and used to retrieve the private key. These have mainly four types according to the type of the revealed output: Timing Analysis [16], Power Analysis [6], Electromagnetic Analysis [17, 18], and Acoustic Analysis [19]. All passive attacks can be either simple or differential. The difference is that, while, in simple analysis attacks, the attacker needs only one measurement, he needs numerous measurements and statistics of these measurements in differential analysis attacks.

4.1. Power analysis attacks

Power Analysis (PA) attacks are based on analyzing the power consumption of the cryptographic device while it performs encryption or decryption [6]. The physical supporting point of these attacks is that, today, Complementary Metal Oxide Semiconductor (CMOS) technology is the one most commonly used for digital integrated circuit implementations. The power consumption during transitions of a CMOS gate is not the same for $0 \rightarrow 1$ transitions and $1 \rightarrow 0$ transitions, with $0 \rightarrow 1$ transitions consuming more power. This gives the attacker a good starting point, where he uses Hamming weight information leaks. In this way the amount of consumed current can be calculated.

4.2. Randomized table window method

In this work, Randomized Table Window Method (RT-WM) algorithm has been implemented as a countermeasure against differential power analysis (DPA) attacks. RT-WM algorithm proposed by Itoh et al. is given in Algorithm 3 [7]. The main difference in the window method is that RT-WM uses randomized data inside the table instead of sequential powers of M.

The recalculation of E determines how the table and the rest of the algorithm works. Equation (1) shows

Algorithm 3 Randomized Table Window Method (RT-WM).

Require: $N = (n_{k-1}, \ldots, n_1, n_0)_2$, $E = (e_{k-1}, \ldots, e_1, e_0)_2$,
$M = (m_{k-1}, \ldots, m_1, m_0)_2$, $Const = 2^{2(k+2)} \bmod N$
Ensure: $M^E \bmod N$

1: $r = b$-bit random number; /* Generate random number */
2: $\omega_{\text{count}} = \lceil (k-b)/t \rceil$ /* Pre-computation Phase 1 starts */
3: $subt = r$
4: $dw = E$
5: **for** i from 0 to $\omega_{\text{count}} - 1$ **do**
6: **if** $dw \geq subt$ **then**
7: $dw = dw - subt$
8: **end if**
9: $subt = subt \cdot 2^t$
10: **end for**
11: $dm = (dw_{b-1} dw_{b-2} \ldots dw_1 dw_0)$
12: $\omega_0 = \left(dw_{k-1} dw_{k-2} \ldots dw_{(\omega_{\text{count}}-i-1)t+b}\right)$
13: $M' = \text{MonPro\_NFS}(M, Const)$ /* Enter MonPro Domain */
14: $Q = M'$ /* Pre-computation Phase 2 starts */
15: $V_0 = M'$
16: $R' = M'$
17: **if** $dm = 0$ **then**
18: $Q = 0$
19: **end if**
20: **for** i from 1 to $2^b - 1$ **do**
21: $R' = \text{MonPro\_NFS}(R', M')$
22: **if** $i = dm - 1$ **then**
23: $Q = R'$
24: **else if** $i = r - 1$ **then**
25: $V_0 = R'$
26: **end if**
27: **end for**
28: $U = R'$
29: **for** i from 1 to $2^t - 1$ **do**
30: $V_i = \text{MonPro\_NFS}(V_{i-1}, U)$ /* Pre-computation Phase 3 */
31: **end for**
32: Start$= 0$ /* Modular Exponentiation Process */
33: **for** i from 0 to $\omega_{\text{count}} - 1$ **do**
34: **if** Start$= 1$ **then**
35: $R' = V_{\omega_i}$
36: **for** j from 1 to $t - 1$ **do**
37: $R' = \text{MonPro\_NFS}(R', R')$
38: **end for**
39: **if** $\omega_i \neq 0$ **then**
40: $R' = \text{MonPro\_NFS}(R', V_{\omega_i})$
41: **end if**
42: **else if** $\omega_i \neq 0$ **then**
43: Start$= 1$
44: **end if**
45: **end for**
46: $R' = \text{MonPro\_NFS}(R', Q)$ /* Normalize Data */
47: $R' = \text{MonPro\_NFS}(R', 1)$ /* Exit MonPro Domain */

how $\omega[i]$, dm, r, b and t in Algorithm 3 make up the exponent E:

$$E = \left(\ldots\left(\left(\omega_0 2^b + r\right)2^t + \omega_1 2^b + r\right)2^t \cdots + \omega_s 2^b + r + dm\right). \quad (1)$$

The calculation for the table values computed in pre-computation phases 2 and 3 as $V_i = M^{\omega_i 2^b + r}$. Using the values in the table, the rest of the algorithm becomes like square for 2^t times and multiply with a table value until the mentioned equation is evaluated. This algorithm brings a preprocessing time and additional memory for the table. An extra subtraction module is not necessary if an adder is already being used within the RSA.

5. The unprotected implementation of RSA cryptosystem

In order to implement the RSA cryptosystem, Montgomery Multiplication block has been realized with *MonPro_NFS_CSA* algorithm, which is given as Algorithm 4. When Montgomery multiplication is realized using Carry Save representation, the multiplicand, multiplier and the result are doubled as Carry and Save.

The RSA Encryption/Decryption algorithm, which uses Montgomery Multiplication, also changes accordingly and it is named *MonExp_NFS_CSA* [20], is given in Algorithm 5.

Algorithm 4 Montgomery Multiplication with No Final Subtraction using Carry Save Adder Representation (*MonPro_NFS_CSA*).

Require: $N = (n_{k-1}, \ldots, n_1, n_0)_2$, $XC = (xc_{k+1}, \ldots, xc_1, xc_0)_2$,
$XS = (xs_{k+1}, \ldots, xs_1, xs_0)_2$, $YC = (yc_{k+1}, \ldots, yc_1, yc_0)_2$,
$YS = (ys_{k+1}, \ldots, ys_1, ys_0)_2$, $r = 2^{k+2} \bmod N$, $n_0 = 1$.
Ensure: $(TC, TS) = (XC, XS)(YC, YS)\, r^{-1} \bmod N$
1: $TC = 0$, $TS = 0$
2: **for** i from 0 to $k+1$ **do**
3: $x_i = xc_i + xs_i$
4: $(C1, S1) = TC + TS + x_i YC_0$
5: $(C2, S2) = C1 + S1 + x_i YS_0$
6: **if** $s2_0 = 0$ **then**
7: $(TC, TS) = (C2 + S2)/2$
8: **else**
9: $(TC, TS) = (C2 + S2 + N)/2$
10: **end if**
11: **end for**

Figure 1 shows the I/O ports, blocks, and connections and important registers inside the RSA implementation. Figure 2 shows the main processing element of the hardware implementation using CSA representation. There are three levels of CSAs, which determine the multiplier's delay.

MonPro_NFS_CSA takes $k+2$ clock cycles. The maximum frequency of the implementation with Xilinx XC2V2000E for $k = 512$ is 140.96 MHz, which takes 3.65 μs resulting in a throughput rate of 140.41 Mb/s. When implemented on Xilinx XC2V4000 for $k = 1024$, the maximum frequency achieved becomes 129.05 MHz; the total time 7.95 μs, and the throughput rate 128.80 Mb/s. As shown in Table 1, the resulting throughput rates are faster than [21, 22, 23], and almost the same speed as [24], which are also architectures using CSAs to realize Montgomery multipliers.

Algorithm 5 RSA Encryption with Montgomery Multiplication with No Final Subtraction using Carry Save Adder Representation (*MonExp_NFS_CSA*).

Require: $N = (n_{k-1}, \ldots, n_1, n_0)_2$, $E = (e_{k-1}, \ldots, e_1, e_0)_2$,
$M = (m_{k-1}, \ldots, m_1, m_0)_2$, $Const = 2^{k+2} \bmod N$
Ensure: $R = M^E \bmod N$

```
Start=0
(MC', MS') = MonPro_NFS_CSA (M, 0, Const, 0, N)
(RC', RS') = (MC', MS')
for i from k − 1 downto 0 do
   if Start=1 then
      (RC', RS') = MonPro_NFS_CSA (RC', RS', RC', RS', N)
      if e_i = 1 then
         (RC', RS') = MonPro_NFS_CSA (RC', RS', MC', MS', N)
      end if
   else if e_i = 1 then
      Start=1
   end if
end for
(RC, RS) = MonPro_NFS_CSA (RC', RS', 1, 0, N)
R = RC + RS
```

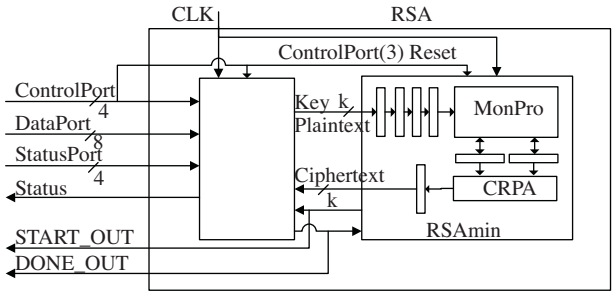

Figure 1. RSA module and its blocks.

Table 1. Implementation results of the Montgomery Multiplier in comparison with the previous works.

Design	Device	Bit length (k)	Clock Fre. (MHz)	Area (# of Slices)	Throughput (Mb/s)
This work	XC2V1500	512	140.96	4339	140.41
	XC2V4000	1024	129.05	5509	128.80
[21]	XC2V1500	512	72.1	3125	71.82
[22]	XC2V1500	512	105.57	4962	105.36
[23]	XC2V1500	512	126.71	5170	126.46
[24]	FPGA	1024	129.1	3611	129

Addition with CRPA takes k/w clock cycles, where k is the key length and w is the word length of CRPA. The decision to choose the word length w was done according to the optimum frequency of the synthesis results (see Table 2). In order not to make the exponentiation slower than the Montgomery Multiplication block, $w = 16$ was chosen.

One RSA encryption takes $\left(k^2 + 3k + k/w + 2\right)$ clock cycles for the best case, where the exponent is

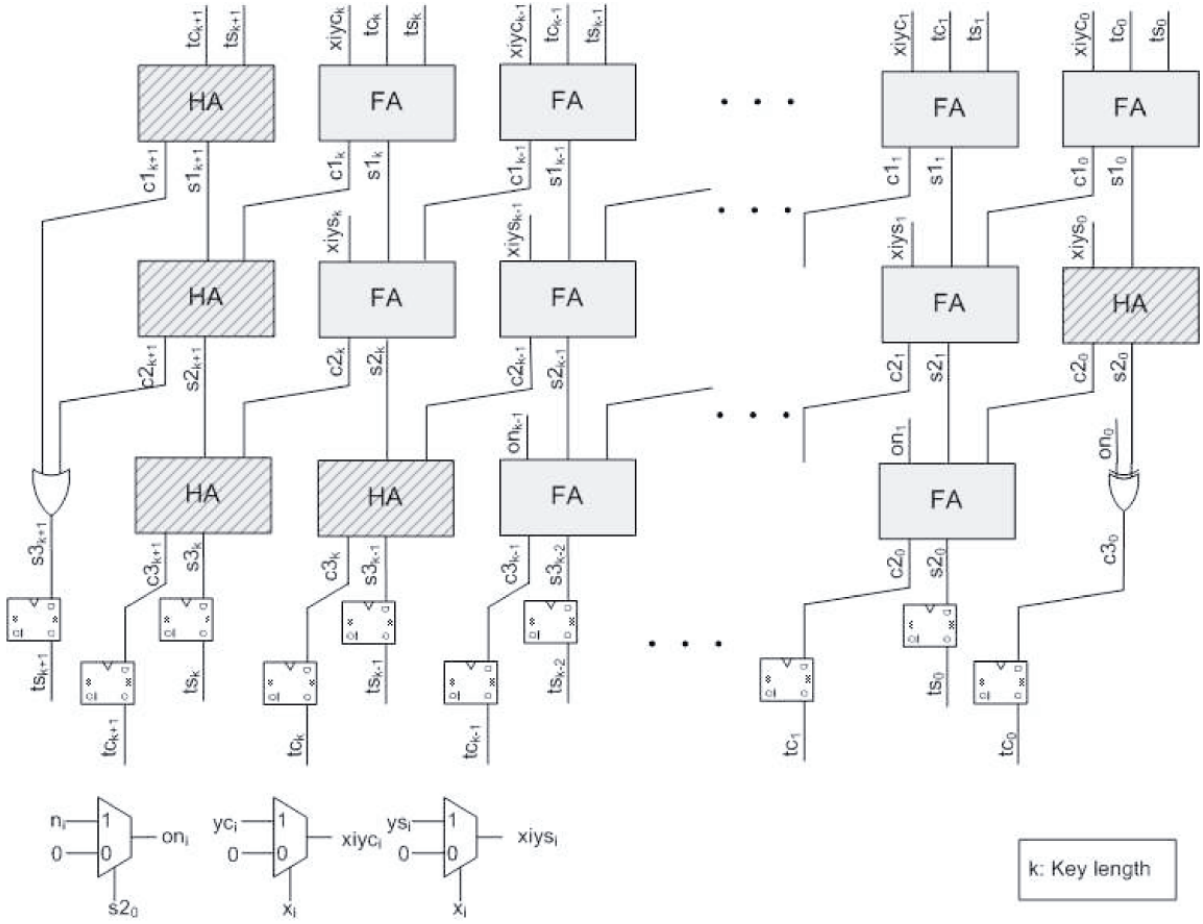

Figure 2. Hardware implementation of the Montgomery Multiplication unit using CSAs.

Table 2. Implementation results of the CRPA module.

Key length (# of bits)	Word size (# of bits)	Time (clock cycles)	Area (Slices)	Clock Fre. (MHz)
512	32	16	976	145.73
512	16	32	932	179.87

$E = 2^{k-1}$, and $\left(2k^2 + 4k + k/w\right)$ clock cycles for the worst case where the exponent is $E = 2^k - 1$. The average for the exponentiation is $\left(\frac{3}{2}k^2 + 5k + k/w + 4\right)$ clock cycles. Table 3 shows the implementation results of the Montgomery multiplier and the RSA modules.

6. The protected implementation of RSA cryptosystem

The RT-WM algorithm given in Section 4.2 is applied as a countermeasure against DPA attacks in this work. There are three phases in the preprocessing in Algorithm 3. ω_{count} comparisons and subtractions take place

Table 3. Implementation results of the Montgomery multiplier and RSA modules.

Module	Parameters	# of Clock cycles	Time s	Area (Slices)	Clock Fre. (MHz)	Throu. (b/s)
MonPro (XC2V1500)	$k = 512$	$k + 2$	3.65 μs	4339	140.96	140.41 M
MonPro (XC2V4000)	$k = 1024$	$k + 2$	7.93 μs	5509	129.05	128.80 M
RSA (XC2V2000)	$k = 512$ $w = 16$	$\frac{3}{2}k^2 + 5k + \frac{k}{w} + 4$	3.4 ms	10240	116.35	150.50 K
RSA (XC2V6000)	$k = 1024$ $w = 16$	$\frac{3}{2}k^2 + 5k + \frac{k}{w} + 4$	18.7 ms	25193	84.33	54.72 K

in preprocessing phase 1. One comparison takes one clock cycle and, since the existing CRPA is used in subtractions, one subtraction costs w (word count of CRPA) clock cycles.

The 2nd phase of the preprocessing calculates $M^r \bmod N$, $M^{dm} \bmod N$ and $M^{2^b} \bmod N$. It takes $\left(2^b - 1\right)$ *MonPro* calculations for this phase.

The 3rd phase of the preprocessing finalizes the table. The table has 2^t k-bit items and it takes $(2^t - 1)$ *MonPro* calculations to finish the table. Since one *MonPro* calculation takes $(k + 2)$ clock cycles in the proposed design, the total time spent in the preprocessing calculations becomes $\lceil (k - b) \rceil (w + 1) + (2 + b + 2^t - 2)(k + 2)$ clock cycles. The preprocessing brings an overhead of 2.1% in total time when compared to the binary method.

The RT-WM parameters selected for this study and the resulting additional time are shown in Table 4. The exponentiation method which replaces the square and multiply method now becomes like t times square and multiply once with a table value. A final multiplication is needed for the normalization. Therefore, accepting that $\omega_0 \neq 0$ for k-bit exponents, the exponentiation time achieved is $(\omega_{\text{count}})(t + 1) + 1 = \lceil (k - b)/t \rceil (t + 1) + 1$ Montgomery multiplications. The exponentiation takes $(\lceil (k - b)/t \rceil (t + 1) + 3)(k + 2) + k/w$ clock cycles. If $k = 512$, $b = 3$, $t = 2$ then one exponantiation takes 394784 clock cycles.

Table 4. Preprocessing time of the RT-WM.

Key length (bits)	b	t	CRPA word count	Time (clock cycles)
512	3	2	16	9492

In addition to the mentioned preprocessing, 2 multiplications are needed for entering and exiting the *MonPro* domain (Algorithm 3) and k/w clock cycles are needed for CRPA addition. The total time spent in RT-WM algorithm with the last CRPA addition is 404276 clock cycles.

The implementation results of the RT-WM algorithm, realized with 512-bit key length, 2-bit window length, and, a 3-bit random number, on Xilinx XCV2600E. An exponentiation time of 18.43 Kb/s throughput and an area of 22712 slices are achieved. The maximum clock frequency is 14.55 MHz. The total encryption process takes 27.79 ms, which was 3.4 ms for the unprotected implementation.

The unprotected implementation fits into XCV1000E, occupying 9037 slices, which is 73% of the available slices. When implementing the protected architecture, the most important addition to the previous implementation are 6 pairs of k-bit registers due to the RT-WM algorithm (Algorithm 3). As there are two registers in

each slice of Virtex-E family, this need causes an inefficient use of the slices which prevents fitting into the same device. The number of slices are 2.5 times of the unprotected implementation. Thus the routing also becomes inefficient, causing a great decrease in the clock frequency.

6.1. Optimization of the hardware implementation

The protected design needed 22712 slices, which could fit into the Xilinx XCV2600E FPGA. We have applied optimizations in order to reduce the number of slices used. Virtex-E family FPGAs incorporate large block SelectRAM memories, where the data widths of the ports can be configured, and the routing is optimized. Hence we used these built-in block RAM structures for the protected design in order to fit into the XCV1000E. The RT-WM algorithm needs 8×513 bits to be used as the "randomized table" values for the chosen parameters as shown in Section 4.2, which were realized with registers. One needs to separate the carry and save pairs in different RAM blocks in order to have read/write access to them at the same clock cycle. Therefore two RAM blocks of 513-bit data length and 4 entries have been defined.

The resulting implementation fit into the device occupying 10986 slices, as 89% of the available slices. All implementation results on XCV1000E are given in Table 5. Comparing the protected RSA implementations, we see that the clock speed increased from 14.55 MHz to 66.66 MHz, making the average case throughput increase from 18.48 Kb/s to 84.42 Kb/s. Total exponentiation time is reduced from 27.11 ms to 6.06 ms.

Table 5. All implementation results on Xilinx XCV1000E FPGA.

Design Module	Unprotected RSA	Protected RSA	Protected RSA
Parameters	$k = 512$, $w = 16$ $k = 512$, $w = 16$	$k = 512$, $w = 16$ $b = 3$, $t = 2$	$k = 512$, $w = 16$ $b = 3$, $t = 2$
Block RAM	No	No	$2 \times 4 \times 513$
Area (slices)	9037	22712	10986
Time (clock cycles)	395812	404276	404276
Clock Fre. (MHz)	81.06	14.55	66.66
Throughput (Kbit/s)	104.85	18.43	84.42
Exp. time (ms)	4.88	27.79	6.06

The time and area cost of the protected design is reduced with block SelectRAM usage.

7. Conclusions

We have implemented an RSA cryptosystem on hardware then modified it to be resistant against DPA attacks. This work is the first hardware implementation of a RSA cryptosystem which is resistant to power analysis attacks. Modular exponentiation is realized with Montgomery Modular Multiplication.

The Montgomery modular multiplier has been realized with Carry-Save Adders. The primarily implemented RSA circuits architecture prevents the extraction of the secret key using Simple Power Analysis attacks.

In the second implementation of this work, the changes within the Randomized Table-Window Method (RT-WM) have been applied over the first implementation in order to have a DPA resistant implementation. This is the first hardware realization of RT-WM.

Both architectures have been implemented on a Xilinx XCV1000E Virtex-E field progammable gate array. The unprotected implementation was clocked at 81.06 MHz and exhibited 104.85 Kb/s throughput, with 4.88 ms total exponentiation time and occupied an area of 9037 slices. The protected implementation was clocked at 66.66 MHz and exhibited 84.42 Kb/s of throughput, with 6.06 ms total exponentiation time and occupied an area of 10986 slices with the use of the built-in block SelectRAM structure inside the XCV1000E. When comparing the protected implementation with the unprotected, it can be seen that the total time increased by 24.2%, while the throughput has decreased by 19.5%. Thus, the final protected implementation became DPA resistant, still fitting into the same device, but slower.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[2] C. K. Koç. High-speed RSA implementation. Technical Report TR 201, RSA Laboratories, November 1994.

[3] P. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, Vol. 44:519–521, 1985.

[4] R.F. Tinder. *Engineering Digital Design*. Academic Press, San Diego, U.S.A., revised second edition edition, 2000.

[5] S. B. Ors. *Hardware Design Of Elliptic Curve Cryptosystems And Side-Channel Attacks*. PhD thesis, Katholieke Universiteit Leuven, Faculteit Toegepaste Wetenschappen, Departement Elektrotechniek, Kasteelpark Arenberg 10, 3001 Leuven (Heverlee), Belgium, February 2005.

[6] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology: Proceedings of CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15-19 1999. Springer-Verlag.

[7] K. Itoh, J. Yajima, M. Takenaka, and N. Torii. DPA countermeasures by improving the window method. In B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, editors, *Proceedings of the 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 2523 of *Lecture Notes in Computer Science*, pages 303–317, Redwood Shores, CA, USA, August 13-15 2002. Springer-Verlag.

[8] D. E. Knuth. *The Art of Computer Programming*, volume 2/Seminumerical Algorithms. Addison-Wesley, 1997.

[9] J. Bos and M. Coster. Addition chain heuristics. In G. Brassard, editor, *Advances in Cryptology - CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 400–407. Springer-Verlag, 1989.

[10] C. K. Koç. Analysis of sliding window techniques for exponentiation. *Computers and Mathematics with Applications*, 10(30):17–24, 1995.

[11] C. D. Walter. Montgomery's multiplication technique: How to make it smaller and faster. In Ç. K. Koç and C. Paar, editors, *Proceedings of the 1st International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*, pages 80–93, Worcester, MA, USA, August 12-13 1999. Springer-Verlag.

[12] C. D. Walter. Montgomery exponentiation needs no final subtraction. *Electronic letters*, 35(21):1831–1832, October 1999.

[13] O. Kömmerling and M. G. Kuhn. Design principles for tamper resistant smartcard processors. In *Proceedings of the USENIX Workshop on Smartcard Technology*, pages 9–20, Chicago, Illinois, USA, May 1999.

[14] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *Advances in Cryptology: Proceedings of EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Konstanz, Germany, May 11-15 1997. Springer-Verlag.

[15] M. Joye, A. K. Lenstra, and J.-J. Quisquater. Chinese remaindering based cryptosystem in the presence of faults. *Journal of Cryptology*, 4(12):241–245, 1999.

[16] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In N. Koblitz, editor, *Advances in Cryptology: Proceedings of CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18-22 1996. Springer-Verlag.

[17] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, volume 2162 of *Lecture Notes in Computer Science*, pages 255–265, Paris, France, May 13-16 2001. Springer-Verlag.

[18] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smard cards. In I. Attali and T. Jensen, editors, *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security (E-smart)*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210, Cannes, France, September 19-21 2001. Springer-Verlag.

[19] A. Shamir and E. Tromer. Acoustic cryptanalysis. Preliminary proof-of-concept presentation, 2004. `http://www.wisdom.weizmann.ac.il/~tromer/acoustic/`.

[20] K. Alptekin Bayam, S. B. Örs, and B. Örencik. A hardware implementation of RSA. In *Proceedings of The International Conference on Security of Information and Networks - SIN*, Gazimagusa, North Cyprus, May 8-10 2007.

[21] K. Manochehri and S. Pourmozafari. Modified radix-2 montgomery modular multiplication to make it faster and simpler. In *Proceedings of The International Conference on Information Technology: Coding and Computing*, pages 598 – 602, Las Vegas, Nevada, USA, 2005. IEEE.

[22] C. McIvor, M. McLoone, and J. V. McCanny. Fast montgomery modular multiplication and RSA cryptographic processor architectures. In *Proceedings of The 37th Asilomar Conference on Signals, Systems and Computers*, pages 379–384, Pacific Grove, California, USA, November 2003. IEEE.

[23] C. McIvor, M. McLoone, and J. V. McCanny. Modified montgomery modular multiplication and RSA exponentiation techniques. In *Proceedings of The Computers and Digital Techniques*, pages 402–408, 2004.

[24] A. P. Fournaris and O. Koufopavlou. A new RSA encryption architecture and hardware implementation based on optimized montgomery multiplication. In *Proceedings of The International Symposium on Circuits and Systems (ISCAS)*, pages 4645–4648, Kobe, Japan, May 2005. IEEE.