NREPO: Normal Basis Recomputing with Permuted Operands

Xiaofei Guo[1], Debdeep Mukhopadhyay[2], Chenglu Jin[1], and Ramesh Karri[1]

[1] Dept. of Electrical and Computer Engineering
New York University School of Engineering
{xg243, cj875, rkarri}@nyu.edu
[2] Dept. of Computer Science and Engineering
Indian Institute of Technology Kharagpur
debdeep@cse.iitkgp.ernet.in

Abstract. Hardware implementations of cryptographic algorithms are vulnerable to natural and malicious faults. Concurrent Error Detection (CED) can be used to detect these faults. We present NREPO, a CED which does not require redundant computational resources in the design. Therefore, one can integrate it when computational resources are scarce or when the redundant resources are difficult to harness for CED. We integrate NREPO in a low-cost Advanced Encryption Standard (AES) implementation with 8-bit datapath. We show that NREPO has 25 and 50 times lower fault miss rate than robust code and parity, respectively. The area, throughput, and power are compared with other CEDs on 45nm ASIC. The hardware overhead of NREPO is 34.9%. The throughput and power are 271.6Mbps and 1579.3μW, respectively. One can also implement NREPO in other cryptographic algorithms.

1 Introduction

The Advanced Encryption Standard (AES) is used in a variety of applications, including smart cards, mobile phones, sensors, and other embedded systems. The decreasing cost of VLSI chips and increasing user throughput requirements make hardware implementation of AES necessary.

In embedded systems such as sensors, medical devices, and radio frequency identification tags (RFID), cost can be a significant constraint. To reduce the area and power, 32- or 8-bit instead of 128-bit AES datapath can be implemented [11, 14]. AES processes each byte with a substitution-box (S-box) which consumes the largest area [11]. One can reduce the S-box size by transforming its operations to a composite field with polynomial basis, normal basis, or mixed bases. The smallest AES S-boxes use normal basis [8, 23][3].

Although AES is difficult to break mathematically, its hardware implementations may contain security vulnerabilities. An attacker can inject malicious faults into a cryptographic device and build correlations between the faulty and the corresponding fault-free outputs to extract the key in a short time. This is known as differential fault analysis (DFA) [5]. DFA has been shown by injecting transient faults using clock glitches [2] and by lowering the supply voltage [17]. An attackers may also inject faults with lasers [7] and electromagnetic pulse [9].

Concurrent error detection (CED) can be used against DFA. Hardware redundancy duplicates the hardware and detects faults by comparing the outputs. Time redundancy computes the input twice on the hardware and compares the results [7]. Information redundancy uses error detecting codes [18, 19, 21]. Hybrid redundancy uses algorithm properties [12, 16, 24].

Hardware redundancy is not affordable when area is the limitation. Time redundancy is vulnerable to same faults injected in both the computation and recomputation [7]. Parity is low-cost, but provides limited defence against DFA [9]. One example hybrid redundancy [16] has large overhead if a cipher uses counter mode, output feedback mode, or cipher feedback mode because only encryption or decryption is needed [26]. Another example hybrid redundancy [24] shares hardware between encryption and decryption and interleaves the encryption and decryption operations in time. But such sharing is difficult and requires many registers to store results if the latency of round operations are different.

Recomputing with Permuted Operands (REPO) is another hybrid redundancy and is effective if the implementation has several computational resources[4]. The limitation of REPO is that if the computation and recomputation are processed by the same resource, its fault detection capability is the same as time redundancy which is vulnerable to the

[3] [8] has the lowest gate equivalence. [23] has the lowest area for a specific ASIC library.
[4] Each byte is recomputed by a different resource.

same faults in both the computation and recomputation [26]. Normal basis REPO (NREPO) overcomes this limitation. While REPO uses byte-wise permutation, NREPO uses bit-wise permutation. NREPO is effective even if the hardware only has a single computational resource. It is useful when computational resources are scarce or redundant structures are difficult to use for CED. Our results show that NREPO has much lower fault miss rate compared to other low-cost schemes.

The paper is organized as follows: Section 2 introduces the AES algorithm and normal basis. Section 3 presents NREPO and implementation results. Section 4 discusses applications of NREPO. Section 5 concludes the paper.

2 Advanced Encryption Standard

We consider 128-bit AES specified by the National Institute of Standards and Technology [22]. AES encryption contains 10 nearly identical rounds plus an initial round (round 0). The data and the key are maintained as 4-by-4 matrices, called *state*. The rounds consist of SubBytes (SB), ShiftRows (SR), MixColumns (MC), and AddRoundKey (ARK)[5].

AES uses a polynomial basis in $GF(2^8)$ where bits are coefficients of a polynomial and the irreducible polynomial is $q(x) = x^8 + x^4 + x^3 + x + 1$. SB uses 16 identical S-boxes to map each byte of state into another byte independently. S-box outputs are computed by inverses in $GF(2^8)$ and an affine transformation. SR cyclic left shift the second, third, and fourth row of state by one, two, three bytes, respectively. MC performs a matrix multiplication. ARK performs XOR with the state and the round key[6]. Two steps in the S-box are:

- *Inversion*: Let $\boldsymbol{c} = \boldsymbol{a}^{-1}$ be the multiplicative inverse in $GF(2^8)$ (except if $\boldsymbol{a} = 0$ then $\boldsymbol{c} = 0$).
- *Affine Transformation*: Then the output is $\boldsymbol{s} = M\boldsymbol{c} \oplus \boldsymbol{b}$, with the constant bit matrix M and byte $\boldsymbol{b}$ shown below:

$$\begin{pmatrix} s_7 \\ s_6 \\ s_5 \\ s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \end{pmatrix} = \begin{pmatrix} 1\,1\,1\,1\,1\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1\,0\,0 \\ 0\,0\,1\,1\,1\,1\,1\,0 \\ 0\,0\,0\,1\,1\,1\,1\,1 \\ 1\,0\,0\,0\,1\,1\,1\,1 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 1\,1\,1\,0\,0\,0\,1\,1 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{pmatrix} \begin{pmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} \oplus \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

where bit 7 is the most significant. Let A be one root of $q(x)$; then the standard polynomial basis is $[A^7, A^6, A^5, A^4, A^3, A^2, A, 1]$. Since inversion in $GF(2^8)$ is difficult to calculate directly, an element G of $GF(2^8)$ can be represented as a linear polynomial (in y) over $GF(2^4)$, as $G = \gamma_h y + \gamma_l$, with multiplication modulo an irreducible polynomial $r(y) = y^2 + \tau y + \nu$. All coefficients are in $GF(2^4)$. So $[\gamma_h, \gamma_l]$ represents G with a polynomial basis $[Y, 1]$ where Y is one root of $r(y)$. We can also use the normal basis $[Y^{16}, Y]$ using both roots of $r(y)$. We get:

$$r(y) = y^2 + \tau y + \nu = (y + Y)(y + Y^{16})$$

$\tau = Y + Y^{16}$ is the *trace* and $\nu = (Y)(Y^{16})$ is the *norm*.

Similarly, we represent $GF(2^4)$ as linear polynomials (in z) over $GF(2^2)$, as $\gamma = \Gamma_h z + \Gamma_l$, with multiplication modulo an irreducible polynomial $s(z) = z^2 + Rz + M$, with all the coefficients in $GF(2^2)$. Again, we could use the normal basis $[Z^4, Z]$. As above, R is the trace and M is the norm of Z.

Lastly, we represent $GF(2^2)$ as linear polynomial in ω over $GF(2)$, as $\Gamma = g_h\omega + g_l$, with multiplication modulo $t(\omega) = \omega^2 + \omega + 1$, where g_h and g_l are single bits. This uses a normal basis $[W^2, W]$ with W one root of $t(\omega)$.

We can convert $GF(2^8)$ operations to $GF(2^4)$ operations, which are expressed in $GF(2^2)$ operations[7].

[5] In round 0, only ARK is performed and in round 10, MC is not performed.

[6] Round key generation includes S-box computations, word rotations, and XOR operations performed on the encryption key.

[7] Addition is XOR in these fields.

3 REPO in Composite Field

3.1 Permutation Property in Normal Basis Arithmetic

It is known that normal basis multiplicative inversion and multiplication in $GF(2^{2m})$ exhibit a cyclic property [23]. The authors use it to perform area optimizations for S-box. We utilize this property to develop a permutation for CED to protect AES against random faults and fault attacks. We give a formal proof of the permutation property.

Theorem 1. *If the elements a and b of $GF(2^{2m})$ are represented in a normal basis, it can be represented as (a_h, a_l) and (b_h, b_l), respectively. a_h and b_h have the first m bits, while a_l and b_l have the second m bits. Permutation P exists such that the following multiplication hold true:*

$$(a_h, a_l)(b_h, b_l) = P(P(a_h, a_l)P(b_h, b_l)) \tag{1}$$

The permutation is $P(a_h, a_l) = (a_l, a_h)$.

Proof. Let (a_h, a_l) and (b_h, b_l) be the coordinates of two elements of $GF(2^{2m})$. Let $T = v^e + v$ and $N = v^e v$ be the trace and norm of v, where v is an element of $GF(2^{2m})$ and $e = 2^m$. Then $\{v^e, v\}$ is a normal basis of $GF(2^{2m})$.

The product $c_h v^e + c_l v$ is:

$$\begin{aligned}
&= (a_h v^e + a_l v)(b_h v^e + b_l v)\\
&= a_h b_h v^{2e} + (a_h b_l + a_l b_h) v^{e+1} + a_l b_l v^2\\
&= a_h b_h (T v^e + N) + (a_h b_l + a_l b_h) v^{e+1} + a_l b_l (T v + N)\\
&= a_h b_h T v^e + (a_h + a_l)(b_h + b_l) N T^{-1}(v^e + v) + (a_l b_l T) v\\
&= (a_h b_h T + (a_h + a_l)(b_h + b_l) N T^{-1}) v^e\\
&+ (a_l b_l T + (a_h + a_l)(b_h + b_l) N T^{-1}) v
\end{aligned}$$

Therefore, we get:

$$c_h = a_h b_h T + (a_h + a_l)(b_h + b_l) N T^{-1} \tag{2}$$

$$c_l = a_l b_l T + (a_h + a_l)(b_h + b_l) N T^{-1} \tag{3}$$

If we permute (a_h, a_l) and (b_h, b_l), we get

$$\begin{aligned}
P(a_h, a_l)P(b_h, b_l) &= (a_l v^e + a_h v)(b_l v^e + b_h v)\\
\Leftrightarrow c_h^{'} &= a_l b_l T + (a_h + a_l)(b_h + b_l) N T^{-1} = c_l\\
\Leftrightarrow c_l^{'} &= a_h b_h T + (a_h + a_l)(b_h + b_l) N T^{-1} = c_h\\
\Leftrightarrow (a_h, a_l)(b_h, b_l) &= P(P(a_h, a_l)P(b_h, b_l))
\end{aligned} \tag{4}$$

Theorem 2. *The same permutation in Theorem 1 makes the following hold true:*

$$(a_h, a_l)^{-1} = P((P(a_h, a_l))^{-1}) \tag{5}$$

Proof. Let (a_h, a_l) be the coordinates of an element of $GF(2^{2m})$. The inverse element (d_h, d_l) has the following property:

$$\begin{aligned}
1 &= (a_h v^e + a_l v)(d_h v^e + d_l v)\\
&= a_h d_h v^{2e} + (a_l d_h + a_h d_l) v^{e+1} + a_l d_l v^2\\
&= a_h d_h (T v^e + N) + (a_l d_h + a_h d_l) N + a_l d_l (T v + N)\\
&= a_h d_h T v^e + a_l d_l T v + [(a_h + a_l)(d_h + d_l) N] T^{-1}(v^e + v)\\
&= ((a_h d_h T + (a_h + a_l)(d_h + d_l) N T^{-1}) v^e +\\
&((a_l d_l T + (a_h + a_l)(d_h + d_l) N T^{-1}) v = T^{-1}(v^e + v)
\end{aligned}$$

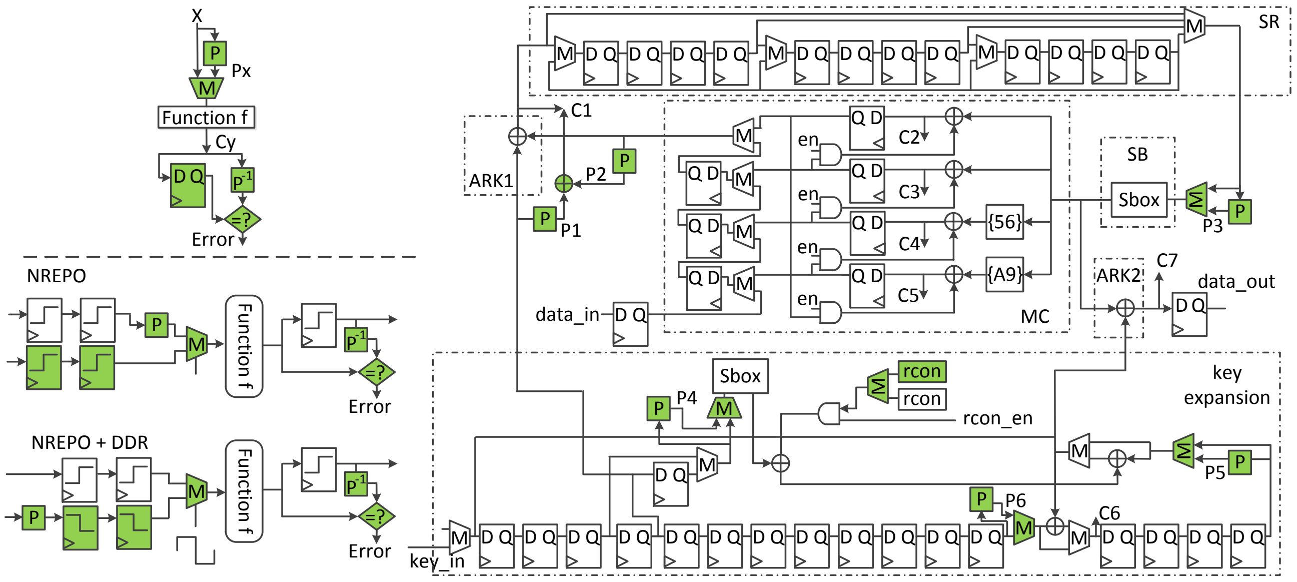

Fig. 1: Upper left: function f is protected by NREPO. P is the permutation. The modules in green are used in NREPO. Bottom left: NREPO and NREPO with DDR. Right: 8-bit AES is protected by NREPO. The SB, SR, MC, and ARK are shown in the dotted boxes. The mux in green shows the added NREPO protections. The duplicated registers are not shown.

Therefore we have:

$$T^{-1} = ((a_h d_h T + (a_h + a_l)(d_h + d_l)NT^{-1})$$
$$T^{-1} = ((a_l d_l T + (a_h + a_l)(d_h + d_l)NT^{-1})$$

Solving the above equation, we get:

$$d_h = (a_h a_l T^2 + (a_h^2 + a_l^2)N)^{-1} a_l \tag{6}$$
$$d_l = (a_h a_l T^2 + (a_h^2 + a_l^2)N)^{-1} a_h \tag{7}$$

If we permute (a_h, a_l), we get

$$(d_h^{'}, d_l^{'}) = (P(a_h, a_l))^{-1} = (a_l, a_h)^{-1}$$
$$d_h^{'} = ((a_h a_l T^2 + (a_h^2 + a_l^2)N)^{-1})a_h = d_l$$
$$d_l^{'} = ((a_h a_l T^2 + (a_h^2 + a_l^2)N)^{-1})a_l = d_h$$
$$\Leftrightarrow P((P(a_h, a_l))^{-1}) = (a_h, a_l)^{-1}$$

Theorems 1 and 2 show such permutation holds in SB and MC with normal basis multiplication and inversion. Such permutation also holds for SR and ARK since the former is wire permutation and the latter is bit operation.

3.2 NREPO Architecture

The NREPO architecture is shown in Fig. 1. An NREPO example is on the upper left. Let the input of function f be X. The permuted input is Px. In the normal clock cycle, the input is computed by f and stored in a register. In the check clock cycle, the input is permuted and computed by f. Then, the output is inverse permuted and checked with

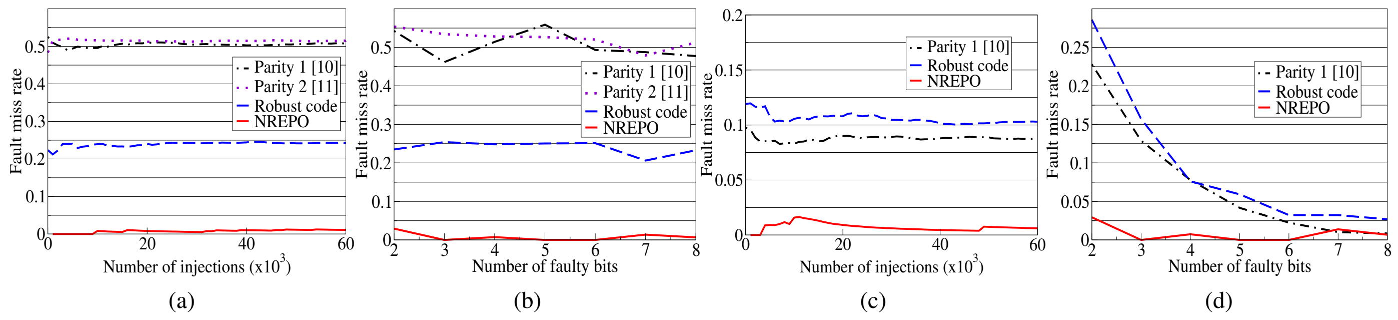

Fig. 2: Simulation results of burst faults show that the FMR of NREPO is superior to that of the parity [19, 21] and robust code [18]. (a) FMR in SB (b) FMR in SB classified by the number of faulty bits (c) FMR in MC and ARK (d) FMR in MC and ARK classified by the number of faulty bits.

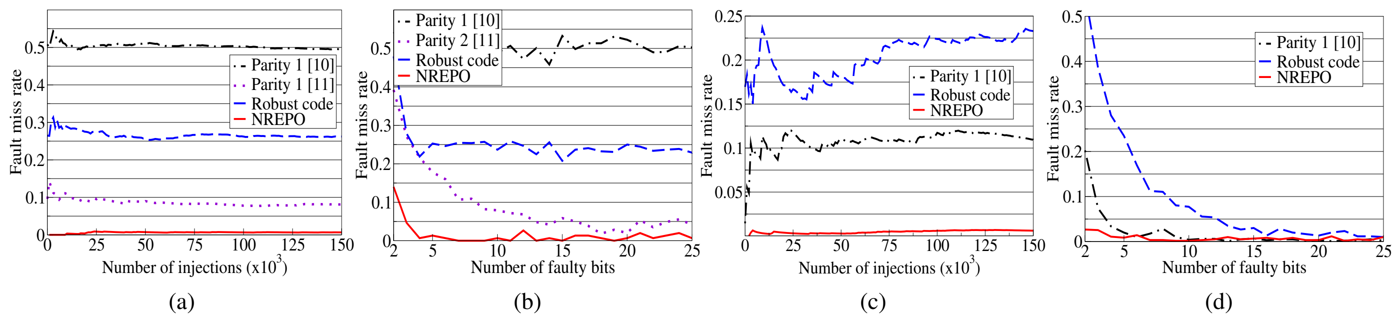

Fig. 3: Simulation results of random faults show that the FMR of NREPO is superior to that of the parity [19, 21] and robust code [18]. (a) FMR in SB (b) FMR in SB classified by the number of faulty bits (c) FMR in MC and ARK (d) FMR in MC and ARK classified by the number of faulty bits.

the value in the register. The wire that connects to the checking circuit is Cy. The green modules are the redundant circuits. f can be any byte-wise round operations in the AES. For S-box, there is an affine transformation after the inversion. Thus we connect the inversion to the affine transformation and the permuted affine transformation. Then we select the outputs from a mux.

3.3 Integrate NREPO in low-cost AES

Our baseline architecture is the compact ASIC implementation in [14]. In this architecture, the sequence of the SR and the SB operations are switched without affecting the AES functionality, and all the paths are 8-bit wide. On the right side in Fig. 1, we integrate NREPO into this low-cost AES implementation. The SB has an S-box. The SR are implemented as shift registers. We convert the MC into normal basis and it contains two constant multipliers[8] and several XOR gates. ARK contains XOR gates. The key expansion contains an S-box, shift registers, and XOR gates. To simplify the diagram, we did not show all the redundant hardware such as the details of the checking hardware.

Px $(1 \leq x \leq 6)$ shows the permuted inputs. Cy $(1 \leq y \leq 7)$ shows the outputs that connect to the checking circuits. All combinational circuits are protected by NREPO. Since the CED computation is independent from the original computation, we employ the double-data-rate (DDR) [20] to improve the throughput of our technique. As shown on the bottom left in Fig. 1, the registers connected to the combinational logic directly are duplicated and share the same computation units. The duplicated registers are triggered by the negative edge. All the permuted inputs are selected at the same clock cycle. The plaintext and key are fed at data_in and key_in one byte per clock cycle and they

[8] The original constants $\{02\}$ and $\{03\}$ in polynomial basis correspond to $\{A9\}$ and $\{56\}$ in the chosen normal basis.

propagate through shift registers in the first four clock cycles. When the fourth clock cycle rising edge comes, the first plaintext and key bytes are XORed by ARK1. The permuted input from the $P1$ and $P2$ are processed by duplicated XORed gates and the results are compared with the ARK1 result at $C1$. We did not compute the permuted data on the original XORs because duplication has less hardware overhead for simple XOR gates. Then the plaintext bytes propagate through SR. At the input of SB, the data can be permuted. The redundant computation results are compared at $C2$, $C3$, $C4$, and $C5$ to check faults in MC and SB. The permuted data is computed by the S-box through $P4$ in the key expansion. $P5$ and $P6$ are added to protect XOR gates in key expansion and ARK2. Rcon are wires with constant values and are also duplicated with a permuted one. The data permuted at $P3$, $P4$, and $P5$ are compared at $C7$ after processed by two S-boxes and XOR gates.

3.4 Fault Analysis

Fault miss rate (FMR) is calculated as:

$$FMR = 1 - fault\ coverage = \frac{T_{undetected}}{T_{total} - T_{correct}}$$

where $T_{undetected}$ is the number of tests in which faults are excited but not detected. T_{total} is the total number of tests we applied. $T_{correct}$ represents the tests in which the faults are not excited. We simulated multiple bit faults for NREPO and compared the FMR with parity 1 [19], parity 2 [21], and robust code [18]. We use burst and random fault models, and these models cover both natural faults and fault attacks [20].

Burst faults occur at the 8-bit operation input or output in the AES encryption. This includes both bit set and reset faults. The size, location, and type of the burst are randomly generated. The simulation results of the AES encryption are shown in Fig. 2. The dot-dash line respresents the FMR of parity 1 [19]. The dotted line represents the FMR of parity 2 [21] in the AES S-box[9]. The dash line represents the FMR of robust code in [18]. The solid line represents the FMR of NREPO. We analyze the FMR of SB and that of MC and ARK separately. Since none of the four techniques are designed for the shift registers in SR, one needs to apply other techniques such as time redundancy. Thus we consider the FMR for SR are the same for all four techniques. For SB, we injected 60,000 burst faults at the operation inputs and outputs. As shown in Fig. 2(a), the FMR for [19], [21], and [18] is around 0.5, 0.5, and 0.25, respectively. The FMR of NREPO is around 0.01; a reduction of 25 to 50 times. Fig. 2(b) shows a detailed analysis when we inject a specific number of faulty bits. The FMR of [19], [21], and [18] are 25 to 50 times of that of NREPO. For MC and ARK, we did similar experiments as illustrated in Fig. 2(c) and Fig. 2(d). The FMR of NREPO is superior than that of [19] and [18]. When the number of faulty bits increases in Fig. 2(d), the FMR of [19] and [18] are close to each other and close to NREPO. Both [19] and [18] protect the MC and ARK by XORing the result of the four state bytes in a column before MC, the corresponding four bytes of the key, and the corresponding four bytes of the output.

Random faults are injected at random locations, i.e., any wire inside the netlist. We injected 150,000 random faults. Fig. 3(a) shows the FMR for the four schemes. The FMR of [19] and [18] are similar to the burst fault results. The FMR of [21] is 0.1 and is lower than that of the burst fault, because this parity technique divides the S-box into five logic levels and implements a parity bit for each level. Thus, a 2-bit fault can also be detected if each bit appears at a different logic level. Fig. 3(b) shows the FMR of a specific number of faulty bits. The FMR of NREPO is between 0.008 and 0.18, while that of [19] and [18] are around 0.5 and 0.25, respectively. The FMR of MC and ARK are shown in Fig. 3(c) and 3(d).

3.5 Security Analysis against DFA

Low-cost fault attack either manipulates the clock or the power supply [15]. By lowering the supply voltage or increasing the clock frequency, the critical path should fail first. This is either caused by the slower rising time of gates or the setup time violation of the flip flop. NREPO protects combinational logics and registers that connect to them with the permutation property. Since this low-cost AES implementation has shift registers, an attacker may target the registers using other more costly injection methods such as laser or electromagnetic wave. In this case, the designer can add protection to registers using the DDR technique [20].

[9] This scheme is for the S-box and we compare the one for normal basis.

Table 1: CED implementations on 45nm ASIC [1]. a. hardware redundancy b. MC and ARK are protected by the parity scheme in [21] c. robust code

Scheme	Gates (overhead)	Freq. (MHz)	Thro. (Mbps)	Power(μW) (overhead)
Original	1,125	628	314.2	1,058.7
H.W.[a]	2,184 (94.1%)	584	292.2	2,476.5 (134.0%)
DDR [20]	1,850 (64.4%)	609	304.5	2,282.4 (115.5%)
Parity 1 [19]	1,255 (11.6%)	628	318.4	1,328.3 (25.5%)
Parity 2 [21][b]	1257 (11.7%)	652	326.4	1427.9 (34.9%)
Rob. [18][c]	1,367 (21.5%)	577	288.6	1,594.1 (50.6%)
REPO [26]	1,534 (36.4%)	607	303.7	1917.2 (81.1%)
NREPO	**1,517 (34.9%)**	**543**	**271.6**	**1,579.3 (49.1%)**

The FMR for NREPO is around 0.01 for byte faults which most DFA uses [3, 13]. An attacker can try 100 fault injections to obtain a single byte fault and thus extract the secret key. However, the designer can set a threshold counter to record the number of fault detections, and the circuit fires a command to erase the key if the threshold is reached. Compared to to parity and robust code, NREPO gives a much improved protection. Compared to time redundancy and REPO which cannot detect faults that last for both the computation and recomputation, the FMR of NREPO is 0.025 when such faults are injected.

3.6 Implementation

The results are shown in Table 1. We implement 8-bit architecture and our S-box is the same as [8]. Hardware redundancy, DDR [20], parity [19, 21], robust code [18], REPO [26], and NREPO are compared. The metrics include (1) the number of gates, (2) gate overhead (ratio of the number of gates for CEDs over that for AES), (3) clock frequency, (4) throughput, (5) power, and (6) power overhead (ratio of power consumption for CED schemes over that for AES).

The hardware overhead of NREPO is much lower than that of hardware redundancy. This overhead mainly comes from the normal basis implementation of MC and the extra muxes. The DDR technique has much higher hardware and power overhead than NREPO because all registers are duplicated and extra muxes are added. Although NREPO also uses DDR technique, it only duplicates the registers that are directly connected to the combinational logic. [19] uses one bit for each byte, but the FMR is 50 times of that of NREPO. Although NREPO has slightly higher hardware overhead than [18], the FMR is 25 times lower than it. Since [21] only protects the S-box, we implement the protection of MC and ARK in [19]. Although [21] uses five parity bits for the S-box, the hardware overhead is 11.7% and is close to [19]. While it is low cost, the FMR for burst fault is 50 times of that of NREPO. [18] has around 21.5% hardware overhead and 50.6% power overhead. Althought the FMR of robust code is lower than parity in most cases, it is much higher than NREPO. The hardware overhead of REPO is slightly higher than NREPO and the power overhead is larger because REPO adds registers to store the intermediate values so that it can shift the column order and recompute.

4 Discussion

4.1 What is the difference between NREPO and REPO?

The key difference is that REPO has the same fault detection capability when only a single computational resource is available. We show four of 16 S-boxes in a 128-bit datapath AES with REPO in Fig. 4(a). $S0$, $S4$, $S8$, and $S12$

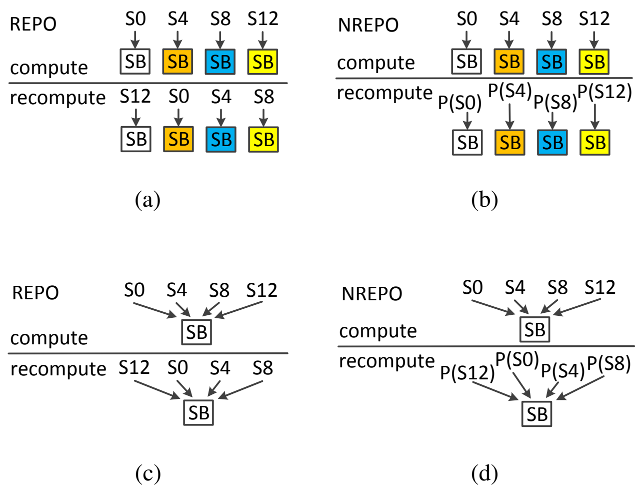

Fig. 4: A comparison between REPO and NREPO (a) REPO in 128-bit datapath (b) NREPO in 128-bit datapath (c) REPO in 8-bit datapath (d) NREPO in 8-bit datapath

are the bytes in the first row of the state.They are byte-wise permuted and recomputed on different S-boxes. In Fig. 4(b), NREPO bit-wise permutes each byte and recomputes the bytes on the same S-box. In Fig. 4(c), only one S-box is present in an 8-bit datapath AES. Although the byte order is permuted in recomputation, the bytes are computed by the same S-box. Therefore, REPO cannot detect faults that last during the computation and recomputation. Even if there is only one S-box, each byte is bit-wise permuted and recomputed on the same S-box in Fig. 4(d). NREPO has 0.025 FMR for those faults that last for both the computation and recomputation.

4.2 Can one use NREPO in 32-bit or 128-bit AES datapath?

NREPO has low FMR and area overhead in 8-bit AES implementation. The property NREPO uses still hold in 32-bit and 128-bit datapaths. We recommend REPO for 128-bit datapath since it detects all single byte faults [26]. We recommend the scheme in [7] for 32-bit datapath since its data in the computation and recomputation are done in different modules, which yields a lower FMR than NREPO.

4.3 Can one use NREPO in the AES decryption?

One can use NREPO in the AES decryption. Although AES decryption uses InvSubBytes and InvMixColumns, InvSubBytes contains the inversion and affine transformation and InvMixColumns can be converted to normal basis multiplication.

4.4 Can one combine NREPO with other CED techniques to achieve stronger fault resilience?

NREPO is a technique that exploits the algorithmic property. As we have shown, one can combine it with DDR to boost the throughput. One can also implement parity along with NREPO to achieve even lower FMR but without modifying the parity scheme since the permutation does not change the parity of the byte. Most notably, one can also combine NREPO with REPO to reduce the FMR of REPO in the 128-bit datapath.

4.5 Can one use NREPO in other cryptographic algorithms?

NREPO uses the permutation property in normal basis multiplication and inversion. Theorem 1 and 2 show that any $GF(2^{2m})$ has such property. Block ciphers such as Camellia [4] and Clefia [25], stream ciphers such as LEX [6], hash functions such as Grøstl [10] can all integrate NREPO.

5 Conclusion

NREPO uses the permutation property in normal basis arithmetic. We integrate NREPO into the AES circuit, and evaluated its performance using a 45nm ASIC library. The throughput of NREPO can be enhanced by DDR. The FMR is 25 to 50 times lower than parity and robust code. NREPO is a fine-grained CED primitives that one can use for normal basis multiplication and inversion. It is particularly suited for implementations in which the computation resources are scarce or it is difficult to use redundant resources.

6 Acknowledgments

This material is based upon work supported by the NSF CNS program under grant 0831349. Debdeep would like to thank Indo-USSTF fellowship to NYU School of Engineering, in 2012.

References

1. Predicative Technology Model. http://ptm.asu.edu/.
2. Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When clocks fail: On critical paths and clock faults. *In Proc. CARDIS*, pages 182–193, 2010.
3. SkSubidh Ali, Debdeep Mukhopadhyay, and Michael Tunstall. Differential fault analysis of AES: Towards reaching its limits. *J. Crypto. Engineering*, pages 1–25, 2012.
4. Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-bit block cipher suitable for multiple platforms - design and analysis. *In Proc. Selected Areas in Cryptography*, pages 39–56, 2001.
5. A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 100(11):3056–3076, 2012.
6. A. Biryukov. The design of a stream cipher lex. *In Proc. Selected Areas in Cryptography*, pages 67–75, 2007.
7. G. Canivet, P. Maistri, R. Leveugle, J. Clédière, F. Valette, and M. Renaudin. Glitch and laser fault attacks onto a secure aes implementation on a sram-based fpga. *J. Cryptology*, 24, 2011.
8. David Canright. A Very Compact Rijndael S-box. *Technical Report NPS-MA-04-001*, 2004.
9. A. Dehbaoui, J. Dutertre, B. Robisson, and A. Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. *In Proc. IEEE FDTC*, pages 7–15, 2012.
10. P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. Grøstl-a sha-3 candidate. http://www.groestl.info/Groestl.pdf, 2011.
11. Tim Good and Mohammed Benaissa. AES on FPGA from the fastest to the smallest. *In Proc. CHES*, pages 427–440, 2005.
12. Xiaofei Guo and R. Karri. Invariance-based Concurrent Error Detection for Advanced Encryption Standard. *In Proc. DAC*, pages 573–578, 2012.
13. Xiaofei Guo, Debdeep Mukhopadhyay, and Ramesh Karri. Provably secure concurrent error detection against differential fault analysis. Cryptology ePrint Archive, Report 2012/552, 2012. `http://eprint.iacr.org/`.
14. P. Hämäläinen, T. Alho, M. Hännikäinen, and T.D. Hämäläinen. Design and implementation of low-area and low-power AES encryption hardware core. *In Proc. IEEE Euromicro Conf. on Digital System Design*, pages 577–583, 2006.
15. D. Karaklajić, J.-M. Schmidt, and I. Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Trans. VLSI*, 21(12):2295–2306, 2013.
16. Ramesh Karri, Kaijie Wu, P. Mishra, and Y. Kim. Concurrent error detection schemes of fault based side-channel cryptanalysis of symmetric block ciphers. *IEEE Trans. CAD*, 21(12):1509–1517, 2002.
17. Farouk Khelil, Mohamed Hamdi, Sylvain Guilley, Jean Luc Danger, and Nidhal Selmane. Fault analysis attack on an aes fpga implementation. *In Proc. New Technologies, Mobility and Security*, pages 1–5, 2008.
18. Konrad J. Kulikowski, Mark G. Karpovsky, and Er Taubin. Robust codes for fault attack resistant cryptographic hardware. *In Proc. IEEE FDTC*, pages 1–12, 2005.
19. M. Mozaffari-Kermani and A. Reyhani-Masoleh. Concurrent structure-independent fault detection schemes for the Advanced Encryption Standard. *IEEE Trans. Computers*, 59(5):608–622, 2010.
20. P. Maistri and R. Leveugle. Double-data-rate computation as a countermeasure against fault analysis. *IEEE Trans. Computers*, 57(11):1528–1539, Nov 2008.
21. Mehran Mozaffari-Kermani and Arash Reyhani-Masoleh. A lightweight high-performance fault detection scheme for the Advanced Encryption Standard using composite field. *IEEE Trans. VLSI*, 19(1):85–91, 2011.

22. National Institute of Stardards and Technology (NIST). Advanced Encryption Standard (AES). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, Nov 2001.
23. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Using normal bases for compact hardware implementations of the AES s-box. *In Proc. Security and Cryptography for Networks*, pages 236–245, 2008.
24. Akashi Satoh, Takeshi Sugawara, Naofumi Homma, and Takafumi Aoki. High-performance concurrent error detection scheme for AES hardware. *In Proc. CHES*, pages 100–112, 2008.
25. Sony Corporation. The 128-bit blockcipher clefia: Algorithm specification. http://www.sony.net/Products/clefia, 2007.
26. X. Guo and R. Karri. Recomputing with permuted operands: A concurrent error detection approach. *IEEE Trans. CAD*, 32(10):1595–1608, 2013.