Reliable Security: Double Error Correction for AN Codes in Encryption Systems

Yuan-Fu Liu and Tsung-Chu Huang
Department of Electronics Engineering, National Changhua University of Education, Changhua, Taiwan

***Abstract*—Reliable encryption is crucial for secure computing. Product (AN) Codes enable fault-tolerant encryption, while conventional AN codes only correct single-bit errors. This paper presents a novel double error correction (DEC) method for AN codes. We propose a trade-off algorithm that combines software-based error searching with a single AWE LUT to achieve efficient error correction with minimal area overhead. Experimental results demonstrate that our method enhances encryption reliability while optimizing computational cost, making it suitable for long-integer computations including cryptographic applications such as RSA.**

I. Introduction

In modern consumer electronics, ensuring cryptographic data integrity is essential. Errors introduced will propagate through during encryption and decryption and threaten security [1]. RSA encryption [2] is a widely used asymmetric encryption scheme that involves computationally intensive operations [3], such as long integer multiplication and modular arithmetic [4][5]. Processing data up to 2048 bits, it is necessary for secure communication. However, it remains vulnerable to fault injection attacks [5], which can introduce critical errors, particularly arithmetic weight errors (AWE), potentially compromising the security and reliability of the encryption system [1].

AN codes[6], a type of arithmetic product code, offers error detection and correction capabilities while maintaining its functionality after arithmetic operations [7]. This distinguishes it from channel codes used in communication systems, such as BCH codes and RS codes [8] which effectively correct transmission errors but fail to address arithmetic errors in computational operations. Taking the channel with computations in Fig.1 for illustration, stages 1-3 show a typical channel 2 which can be corrected by conventional channel codes. However, AWEs occur in stages 1 and 3-10 cannot be detected and corrected by any conventional channel codes. In fact, the AN codes can also be applied as a channel codes which can enclose all the conventional channel codes encoders and decoders, as well as the other arithmetic computations.

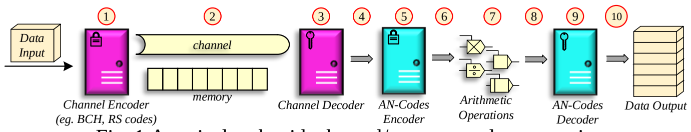


Fig. 1 A typical path with channel/memory and computations.

However, conventional implementations are restricted to detecting single-bit AWE [9], limiting their reliability in encryption applications. The author of [10] pointed some analogy between product codes and Hamming-based channel codes, however, so far no analytical decoding or systematic LUT-based approach for DEC has been explored.

In this paper, we propose a method to correct double AWEs in AN codes, enhancing reliability compared to existing implementations. In Section II the AWE model and AN codes are reviewed. Section III details the proposed technique. In Section IV the trade-off feasibility is evaluated by comparing the timing and area complexity. Conclusions with futural work are then drawn in Section V.

II. Previous Work

A. Arithmetic Weight Error Model

An n-bit 2's complemented binary-coded binary (BCB) number x can represented as

$$x=(x_i)_{i=0}^{n-1}=(x_{n-1}\cdots x_0)_2=\Sigma_{i=0}^{n-1}2^i\,x_i-2^n\,x_{n-1}, \qquad (1)$$

where $x_i \in \{0,1\}$. It can be also represented by a TCB number,

$$x=(t_i)_{i=0}^{n-1}=(t_{n-1}\cdots t_0)_{TCB}=\Sigma_{i=0}^{n-1}2^i\,t_i, \qquad (2)$$

where $t_i\in\{-,0,\ +\}$ and signs $\pm$ separately represent ±1. Here 2^i will be called a *digital weight* (DW) for distinguishing with *weights* in neural network (NN) and *arithmetic weight* (AW) of *x*. AW is defined as the minimum count of $\pm$ signs for representing, namely,

$$\boldsymbol{w}(x)=min\{w|x=\Sigma_{j=1}^{w}2^{i(j)}t_i,\ t_i\in\{\pm1\}\}. \qquad (3)$$

The AW distance between two integers *x* and *y*, $d_A(x,y)$ is defined as $\boldsymbol{w}(x-y)$, where $\boldsymbol{w}$ is the arithmetic weight.

B. AN Codes

The AN codes take the residue $AN\equiv\ 2^i(mod\ A)$ for error location. In a (n,k) AN codes, the message word N, $0\le N\le\lfloor(2^n-1)/A\rfloor=B$, is encoded to the product codeword *C=AN*, possibly infected by an AWE, e as $W=C+e=AQ+R$, and decoded by $N'=W/A$. where $0\le R<A$. As illustrated in Fig.2, the *w*-AWE, *e*, can be located as $\left|\Sigma_{j=1}^{w}(\pm2^{i(j)})\right|_A$, where $|x|_A$ denotes the residue of *x* mod *A*. Note that in the AWE model a single error can be $+2^i$ or -2^i at bit location i therefore a subgroup and a coset are generated by $\times$ 2 from $+1$ and $-1\equiv A-1$ for generating distinct residues for identifying the error location. For example, a check-bits 0011_2 locates an error at index 3 of a Hamming codeword, while a residue 2^3 (mod 29) denotes an addition error at location 3 of the 29N codes.

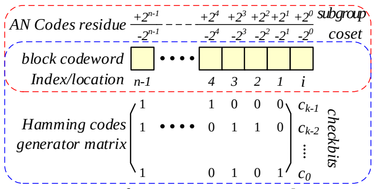


Fig. 2 Comparison of Hamming/AN Codes in error location.

If *A* is an odd prime number, and *2* is a primitive element for generating a multiplicative group G(*A*) of Galois field, GF(*A*), then G(*A*) is perfect to locate the AWE indices of a $n\le(A-1)/2$-bit *AN* codeword. Table I shows the searched AN-code multipliers A. Row $\boldsymbol{w}$(A) shows the AW of A. For example, since GF(29)={0; (0, 1, 2, 4, 8, 16, 3, 6, 12, 24, 19, 9, 18, 7, 14, 28, 27, 25, 21, 13, 26, 23, 17, 5, 10, 20, 11, 22, 15)}, 29 is a good multiplier for a perfect AN codes with single error correction (SEC) capability of a 14-bit codeword for $\mathrm{N}\le564$.

TABLE I. Multipliers for perfect AN codes

n	5	6	9	11	14	18	23	26	29	30	33	35	39	41	50	51	53	55	65
A	11	13	19	23	29	37	47	53	59	61	67	71	79	83	101	103	107	121	131
w(A)	3	3	3	3	3	3	3	4	3	3	3	3	3	4	4	4	4	3	3

Owing to the limitation of space, the introduction and implementation can be reviewed from [9]. Double AW errors generally denote two single AW error events. However, some double AW errors may result in a single AWE, and even a correct sum. In this paper, a double AW error is defined as the arithmetic error with an AW of 2. Additionally, someone may regard that the look-up table (LUT)-based is easy owing to its exhaustivity. However, almost all analytical algorithms of Hamming-code-based channel codes should also need the LUTs of GF(2^n) addition and multiplication. In this paper, the proposed algorithm of AN codes can spend two basic LUTs with the same complexity as that in analytical algorithms of Hamming-code-based channel codes.

III. Proposed Technique

A. Look-up table(LUT) based on DEC

For a b-bit received codeword W=AN+E, assume a double AW error, $E=s_1\cdot2^l+s_2\cdot2^m$, where *l* and *m* are location multiplicities in (0, 1,…*b-1*), and s_1 and s_2 in (-1, 0, +1). W is correct if both s_1 and s_2 are zeros and with a single error if only one of them is zero.

Considering the single-AWE with $\mathrm{s}_1=0$ at first, $\mathrm{W\%A}=\mathrm{r}_1=s_1\cdot2^l$, where the sign % means the modulo operation for finding the non-negative remainder. Since A is a primitive multiplier, r_1 and pair (s_1,l) will be 1-1 mapped. We can construct an *l*-LUT in an area complexity of $\boldsymbol{O}(n\cdot\log n)$ for looking the error location for correction [9]. Similarly, when $\mathrm{W\%A}=\mathrm{R}=s_1\cdot2^l+s_2\cdot2^m$, a 1-1 LUT can be also constructed for R and tuple $(s_1,\ l,s_2,m)$ for correcting a double AW error but with an area complexity of $\mathrm{O}(n^2\cdot\log n)$.

To evaluate the suitability of a number *A* as an AN codes capable of implementing DEC, Fig.3(a) illustrates the algorithm for searching a least multiplier for a DEC AN codes. First, the data of specific bits (*n*-bits) is multiplied by *A*. The data multiplied by *A* is referred to as W (i.e., AN). Next, single and double AWE are generated and added to the AN to simulate error data (i.e., W'). The error data is then divided by *A*, and the remainders are analyzed. If all remainders are distinct for the error data, *A* is deemed suitable for detecting and correcting single and double AWE in the specified bits.

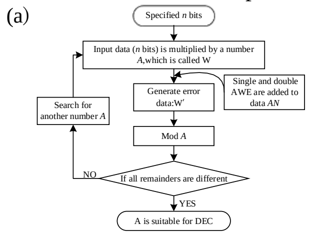
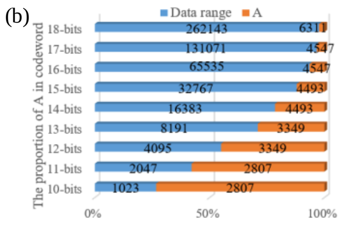

(a)
Specified *n* bits
Input data (*n* bits) is multiplied by a number *A*,which is called W
Generate error data:W'
Single and double AWE are added to data *AN*
Search for another number *A*
Mod *A*
NO
If all remainders are different
YES
A is suitable for DEC


Fig. 3 (a) The flowchart for searching *A*, (b) the proportion of *A*.

Fig.3(b) presents the data range corresponding to different bit widths along with the identified A values. The results indicate that as the bit width increases, the proportion of A within the data decreases, making its overhead negligible for large integers. This observation supports the feasibility of applying DEC to encryption technology that involve large-scale long-integer computations, such as RSA.

As illustrated in Fig.4, the *LUT* for DEC comprises two components: the single AWE *LUT* and the double AWE *LUT*. The single AWE *LUT* is capable of correcting 2b AWE, whereas the double AWE *LUT* can correct $4C_2^b$ AWE, where b represents the bit width of the codeword W (i.e., the bit width of AN). As a result, the size of the double AWE *LUT* is considerably larger than that of the single AWE *LUT*, posing significant area overhead concerns. To address this issue, the following section presents an alternative approach that achieves DEC using only the single AWE *LUT*, effectively reducing area costs.

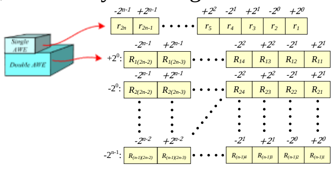


Fig. 4 Look-up table (LUT) for DEC.

B. *A Trade-off Method For DEC*

A *LUT-based* approach (called *Parallel*) for DEC provides fast and efficient error correction but incurs excessive area overhead, making it unsuitable for VLSI implementation. Since the error accumulation is addition and/or subtraction, it is easy to prove that the error location function DE(R) = E(r_1+r_2) = E(r_1) + E(r_2) = SE(r_1) + SE(r_2), where the DE and SE are double/single error location functions. This makes the space (area)-time complexity of the AN decoder can be hold and the area complexity can be approximately substituted by time complexity. To achieve a balance between efficiency and resource utilization, we propose a *Trade-off* approach that combines software-based searching with single AWE *LUT* for error correction, as illustrated in Fig.5.

```
Given: multiplier A, received codeword W with length b
R = W % A; Q = W // A
if R == 0: output Q
else:
    for l in range(b):
        for s1 in {-1, 1}:
            lookup r1 and ΔQ in l-LUT
            r2 = R - r1
            if r2 == 0: output Q - ΔQ
            else if r2 < 0:
                r2 += A
            else:
                lookup s2 and m in r-LUT
                if found: break
        if found: break
    if found: output (W - s1*2^l - s2*2^m) % A
    else: multiple-AW errors are seldom occurred
```

Fig. 5 *Trade-off* algorithm for DEC.

In the proposed algorithm, the ***l*-table** is a *LUT* that maps single-error weight positions and signs to their corresponding remainders, while the **r-table** performs the inverse mapping, retrieving single-error weight positions and signs from a given remainder. *Trade-off* approach leverages only the single AWE *LUT*, significantly reduces the *LUT* size and the area required for synthesis, thereby enhancing feasibility for VLSI implementation. Moreover, this method eliminates the need for extensive loop iterations and per-AWE computations, effectively reducing computational time overhead.

C. *Automatic Synthesizer*

Building on the *Trade-off* method, a Python-based synthesizer was developed to generate *LUT* for different AWE. This synthesizer converts the *LUT* from a *.py (python)* file into a *.v (verilog)* file. After implementing the required *LUT* functionality in Python, the synthesizer automatically generates the corresponding Hardware Description Language (HDL) code, facilitating seamless hardware synthesis. This enables the rapid development of fault-tolerant hardware designs with DEC capabilities, making them readily deployable on FPGA or ASIC platforms in the future.

IV. Evaluation and Experimental Results

To evaluate the cost of the three methods-*Parallel*, *Sequential*, and *Trade-off*-we assume the total product of time and area is a fixed value. The area cost was obtained through synthesis using Synopsys Design Compiler in a 0.18μm CMOS technology. The time cost represents the execution time required for software processing. To ensure consistency in evaluation, all execution time measurements for the different methods were conducted using Python on the Pynq-z2 platform**.** The summarized results are presented in Table III**.**

TABLE II. The cost table of each method

			Parallel		Sequential		Trade-off	
n	*N*	*A*	*Area:* (um^2)	*Time: Est*	*Area: Est*	*Time:* (s)	*Area:* (um^2)	*Time:* (s)
4	14	665	17743.6	0.00029	1259.65	0.004	10699.3	0.00048
8	19	1939	34210.3	0.00036	1473.39	0.008	18339.5	0.00066
12	24	3349	54434.0	0.00042	1628.87	0.014	28513.8	0.00081
16	29	4547	76838.8	0.00046	1765.41	0.020	34779.0	0.00102
20	33	6311	95569.9	0.00049	1768.39	0.026	40531.8	0.00116
24	38	13837	121430.7	0.00053	1850.08	0.035	51121.7	0.00127
28	43	17619	165939.4	0.00055	1954.94	0.045	63374.0	0.00141

In the Parallel Time and Sequential Area columns of Table III, as the experiment is still ongoing, we estimate the values of these two columns based on the principle that the total product of time and area remains a fixed value. As shown in the table, the Trade-off approach achieves a more balanced distribution of time and area costs, preventing extreme cost values that would otherwise pose implementation challenges.

To further highlight the cost differences among these methods, the data from Table III was normalized using Min-Max scaling and visualized as a curve plot in Fig.6. The horizontal axis represents the bit width, while the vertical axis denotes the corresponding cost.

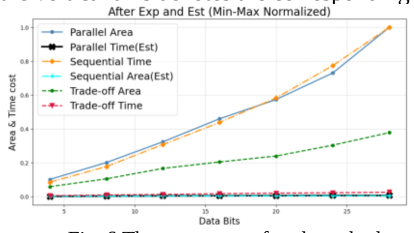


Fig. 6 The cost curve of each method.

As shown in the results, the Parallel Area cost and Sequential Time cost (blue and orange lines) exhibit a progressively steeper cost increase as the bit width grows. In contrast, the Trade-off approach (green and red lines) maintain a slow linear growth in both area and time costs. Specifically, the slope of Trade-off Area is approximately half that of Parallel Area, while Trade-off Time remains nearly constant. This balanced approach optimizes computational efficiency and hardware resource utilization, making it highly suitable for cryptographic applications involving large integer computations, such as RSA. It achieves fault tolerance while maintaining cost-effectiveness.

V. Conclusions

For DEC of AN codes, no one has proposed an analytical method yet, but the table lookup method suffers an area complexity of $\boldsymbol{O}(n^2 \cdot \log n)$, which is actually very expensive. This paper proposes a more feasible algorithm with a time complexity of $\boldsymbol{O}(n)$ and a space complexity of $\boldsymbol{O}(n \cdot \log n)$, which makes long integer calculations more reliable.

References

[1] D. Boneh, *et. al.*, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology - EUROCRYPT '97*, Lecture Notes in CS, vol. 1233, Springer-Verlag, 1997, pp. 37–51.

[2] W. Stallings, "Cryptography and Network Security: Principles and Practice," 7th ed., Pearson, 2016.

[3] Y. Sun, *et. al.*, "An area efficient modular arithmetic processor," *2003 5th Int'l Conf. on ASIC*, Beijing, vol.2, pp. 1273-1276.

[4] R. Rivest, *et. al.*, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications, ACM, 21(2), pp. 120–126, 1978.

[5] M. Joye and M. Tunstall,"Fault Analysis in Cryptography," Springer, 2012.

[6] J. M. Diamond, "Checking codes for digital computers," Proc. IRE, vol. 43, pp. 487-488, Apr. 1955.

[7] D. T. Brown, "Error detecting and error correcting binary codes for arithmetic operations," IRE Trans. Electron. Comput., vol. EC-9, pp. 333-337, Mar. 1960

[8] S. Lin and D. Costello. Error Control Coding: Fundamentals and Applications. Prentice-Hall, Englewood Cliffs, NJ, 2004.

[9] C.-D. Tsai, T.-Y. Chen, H.-W. Fu and T.-C. Huang. "TCBNN: Error-Correctable Ternary-Coded Binarized Neural Network," 2021 IEEE Int'l Conf. on AI Circuits and Systems (AICAS2021), virtual, June 7, 2021.

[10] C.-K. Liu. *Error Correction Codes in Computer Arithmetic*, technical report, University of Illinois at Urbana-Champaign, 1972.