**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. Despite previous studies [[1]](#ref1), no analytical decoding or systematic LUT-based approach for DEC has been explored in the past fifty years. 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 cryptographic applications such as RSA.

# Introduction

In modern cryptographic applications, ensuring data integrity is essential. Faults introduced in the encryption process may propagate, affecting final computations and compromising security [[2]](#ref2).

RSA encryption [[3]](#ref3) is a widely used asymmetric encryption scheme that involves computationally intensive operations [[4]](#ref4)[[5]](#ref5), such as long integer multiplication and modular arithmetic [[6]](#ref6). Processing data up to 2048 bits, it is necessary for secure communication. However, it remains vulnerable to fault injection attacks [[7]](#ref6), which can introduce critical errors, particularly arithmetic weight errors (AWE), potentially compromising the security and reliability of the encryption system [[2]](#ref2).

AN codes [[8]](#ref8), a type of arithmetic product code, offers error detection and correction capabilities while maintaining its functionality after arithmetic operations [[9]](#ref9). This distinguishes it from channel codes used in communication systems, such as BCH codes and RS codes [[10]](#ref10). However, conventional implementations are restricted to detecting single-bit AWE, limiting their reliability in encryption applications.

In this paper, we propose a method to correct 2-bit AWE in AN codes, enhancing reliability compared to existing implementations. In Section II with a review of 2-bit error-correcting channel codes and the current methods for encoding and decoding AN codes. Section III details the proposed technique. In Section IV evaluates the improvements and compares the results.

# Previous work

## Channel codes

Previous studies have demonstrated that channel codes, such as BCH and RS codes [[10]](#ref10), effectively correct transmission errors but fail to address arithmetic errors in computational operations. This limitation underscores the need for specialized techniques, such as AN codes, for effective arithmetic error correction.

## Arithmetic Weight Error Model

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

, (1)

where. It can be also represented by a TCB number,

, (2)

where and signs ± separately represent ±1. Here will be called a *digital weight* (DW) for distinguishing with *weights* inneural network (NN) and *arithmetic weight* (AW) of *x*. AW is defined as the minimum count of ± signs for representing, namely,

. (3)

The AW distance between two integers *x* and *y*, ​is defined as , where ***w*** is the arithmetic weight. A number with AW ≤ 2 (3) is called nice (light) number respectively. The AW-minimization algorithm [[11]](#ref111) reduces the AW distance efficiently.

## AN Codes

The AN codes take the residue for error location. In a () AN codes, the message word *N*, , is encoded to the product codeword *C*=*AN*, possibly infected by an AWE, as , and decoded by . where . As illustrated in Figure 1, the *w*-AWE, *e*, can be located as , where denotes the residue of *x* mod *A*. Note that in the AWE model a single error can be or at bit location *i* therefore a subgroup and a coset are generated by × 2 from +1 and −1 ≡ A − 1 for generating distinct residues for identifying the error location. For example, a checkbits locates an error at index 3 of a Hamming codeword, while a residue (mod 29) denotes an error at location 3 of the 29N codes.



1. 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 -bit *A*N codeword. Since it is easy for today’s computer, for applications only, we will search for a good multiplier number A for AN codes by an exhaustive searching scheme without too many complicated theories. Table I shows the searched AN-code multipliers A Row ***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 .

1. 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 |

To analyze the differences and applications of channel codes and AN codes, we propose an transmission fault module, as illustrated in Figure 2. After passing through the channel encoder, data undergoes 10 transmission stages, with errors potentially occurring at any stage. The error correction capabilities of channel codes and AN codes across these stages have been evaluated and compared in Table II.



1. Proposed transmission fault module.

As shown in Table II, AN codes can serve as an alternative to channel codes for error correction (i.e., ②). Additionally, integrating AN codes into the channel encoder and decoder enhances fault tolerance by preventing errors during the encoding and decoding processes (i.e., ①and ③).

1. an codes v.s. channel codes

|  |  |  |
| --- | --- | --- |
|  | Not enable to detect and correct errors. | Enable to detect and correct errors. |
| Channel Codes | ①③④⑤⑥⑦⑧⑨⑩ | ② |
| AN Codes | ④⑤⑨⑩ | ①②③⑥⑦⑧ |

Due to the limitations of Galois fields, the described AN code has a drawback: it can only detect and correct single-bit arithmetic weight errors and lacks the capability for double error correction (DEC), making it unable to handle two-bit or higher arithmetic weight errors. When the most significant bit (MSB) is prone to errors, the second MSB is often similarly vulnerable. To improve reliability, the implementation of DEC is essential.

# Proposed technique

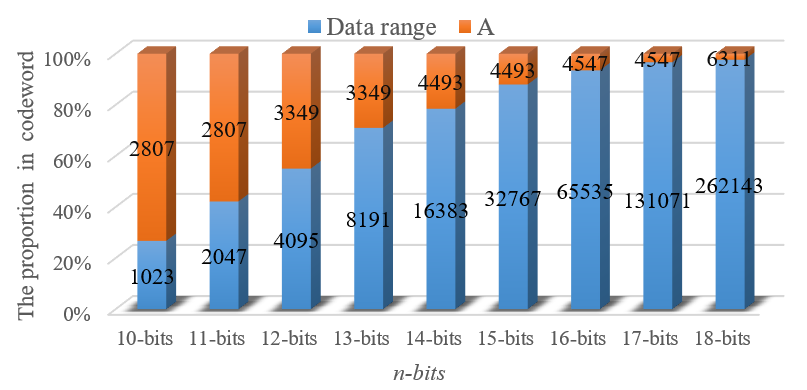
## Look-up table(LUT) based on DEC

To evaluate the suitability of a number *A* as an AN codes capable of implementing DEC, Figure 3 illustrates the proposed method for identifying a suitable AN codes to implement DEC. First, the data of specific bits (*n*-bits) is multiplied by *A*. The data multiplied by *A* is referred to as *AN* (*N*-bits). Next, single and double AWE are generated and added to the *AN* to simulate error data. 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.



1. The flowchart for searching n-bits DEC AN codes.

Figure 4 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 computations, such as RSA.



1. The proportion of A in codeword.

The identified *A* ensures distinct modulus values for any error data. Leveraging this property, a one-to-one error *LUT* can be constructed to correct single and double AWE, as demonstrated in Figure 5. As illustrated in the figure, the *LUT* for DEC comprises two components: the single AWE *LUT* and the double AWE *LUT*. The **single AWE *LUT*** is capable of correcting 2N AWE, whereas the **double AWE *LUT***can correct AWE, where *N* represents the bit width of the error data (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.



1. Look-up table (LUT) for DEC.

## 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. Alternatively, if error correction is performed without utilizing a *LUT* and instead relies on a purely software-based approach (called *Sequential*), the process would involve multiple iterations to compute, compare, and correct all possible AWE values. While this method minimizes area overhead, it significantly increases the search and computation time, making it inefficient for practical implementation. To achieve a balance between efficiency and resource utilization, we propose a *Trade-off* approach that combines software-based searching with a single AWE *LUT* for error correction, as illustrated in Figure 6



1. *Trade-off* algorithm.

In the proposed algorithm, **SEC\_LUT** represents the single AWE *LUT*. *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.

## 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, as shown in Figure 7



1. Synthesizer flow of LUT.

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.

# Evaluation and Experimental Results

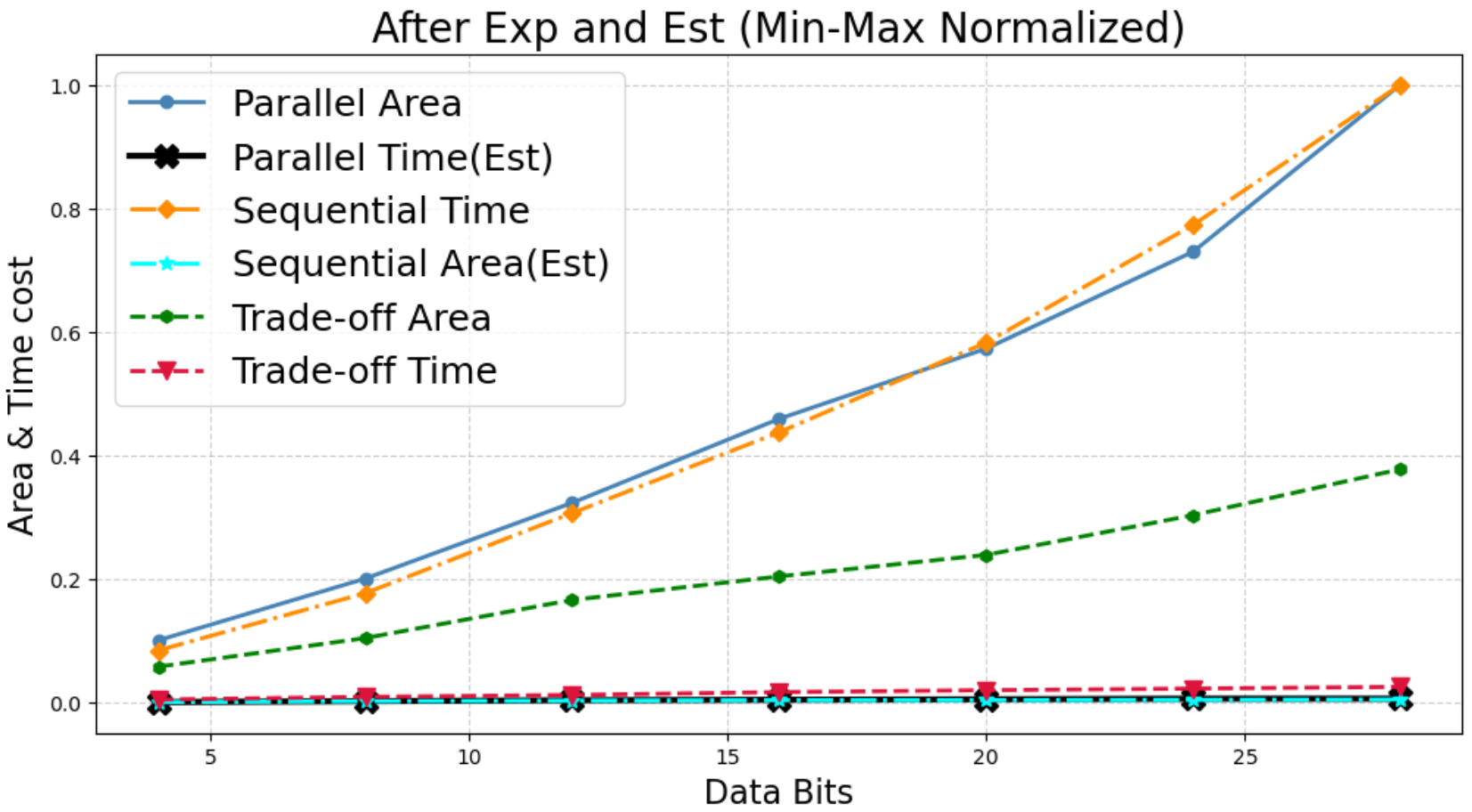
To evaluate the cost of the three methods-*Parallel, Sequential*, and *Trade-off*-we assume the total product of time and area is a fixed value. The area cost was obtained through synthesis using **Synopsys Design Compiler** in a **0.18μm CMOS technology**. The **time cost** represents the execution 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.**

1. The cost table of each method

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | **Parallel** | | **Sequential** | | **Trade-off** | |
| ***n*** | ***N*** | ***A*** | ***Area:***  () | ***Time:***  *Est* | ***Area:***  *Est* | ***Time:***  (s) | ***Area:***  () | ***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 **Figure 8**. The horizontal axis represents the bit width, while the vertical axis denotes the corresponding cost.



1. 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.

##### References

1. C.K. Liu. "CKLiu1972\_Error Correction Codes in Computer Arithmetic," technical report, University of Illinois at Urbana-Champaign, 1972.
2. D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in \*Advances in Cryptology - EUROCRYPT '97\*, Lecture Notes in Computer Science, vol. 1233, W. Fumy, Ed. Berlin, Germany: Springer-Verlag, 1997, pp. 37–51.
3. W. Stallings, "Cryptography and Network Security: Principles and Practice," 7th ed., Pearson, 2016.
4. S. Y. Sun Yile and W. X. Wu Xingjun, "An area efficient modular arithmetic processor," 2003 5th International Conference on ASIC Proceedings, Beijing, China, 2003, pp. 1273-1276 Vol.2
5. Xin Zhou and Xiaofei Tang, "Research and implementation of RSA algorithm for encryption and decryption," Proceedings of 2011 6th International Forum on Strategic Technology, Harbin, Heilongjiang, 2011, pp. 1118-1121
6. R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, no. 2, pp. 120–126, 1978.
7. M. Joye and M. Tunstall,"Fault Analysis in Cryptography," Springer, 2012.
8. J. M. Diamond, "Checking codes for digital computers," Proc. IRE, vol. 43, pp. 487-488, Apr. 1955.
9. 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
10. R. W. Hamming, "Error Detecting and Error Correcting Codes," Bell System Technical Journal, vol. 29, no. 2, pp. 147–160, 1950.
11. C.-D. Tsai, T.-Y. Chen, H.-W. Fu and T.-C. Huang. "TCBNN: Error-Correctable Ternary-Coded Binarized Neural Network," 2021 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS2021), virtual, June 7, 2021.