Greeting, everyone. I’m Yuan-Fu Liu from National Changhua University of Education.

Today, I will present our research on **Double Error Correction Arithmetic Product Codes (AN Codes)** for encryption systems, my advisor is Professor Tsung-Chu Huang.

Outline:

Here is the outline of my presentation:

* First, I’ll explain the motivation and background behind this work.
* Then, I’ll review related work on AN codes.
* Next, I’ll introduce our proposed technique.
* Finally, I’ll share the evaluation and experimental results that demonstrate the effectiveness of our method.

Introduction 1:

With the rapid development of hardware, both the volume and precision of data computation have grown significantly.  
During computation, **data integrity and confidentiality** have become critical concerns, particularly in security-sensitive applications.

Introduction 2:

Encryption technologies play a central role in ensuring data confidentiality.  
Among them, **RSA**, a widely used asymmetric encryption system, involves operations such as large integer exponentiation and modular arithmetic.  
As shown in this figure, these operations can process data up to 2048 bits.  
Any fault during this process could lead to **catastrophic error** of the encryption.

Introduction 3:

Since the connection between arithmetic codes and channel codes was first proposed in [10], little progress has been made toward **double error correction (DEC)** in AN codes.

While conventional AN codes can only correct **single arithmetic weight errors (AWE)**, our work proposes a **lookup-table-based approach for correcting double arithmetic weight errors (AWEs)**.

We also introduce a **hardware/software trade-off strategy** to mitigate the LUT area overhead, making the technique more practical for cryptographic applications.

Previous Work 1:

Before introducing AN codes, we first explain the arithmetic weight error model.

Arithmetic weight refers to the distribution of 1s and 0s in a binary representation of a number, when a number encounter an arithmetic weight error, its value changes by two to the power of k, where k is an integer.

For example, if the number 3 undergoes a plus (two to the power of one) AWE, its binary changes from 0011 to 0101, flipping two bits.

In contrast, a channel bit error flips only a single bit, such as from 0011 to 0001.

Previous Work2:

In AN codes, data is multiplied by a constant A to form a codeword ***C=A\*N***.

If an error ***e*** occurs during computation, the corrupted codeword becomes ***C′= C + e***.  
By computing ***C′ mod A***, we can determine whether an error occurred.  
If the remainder is 0, the output is error-free; otherwise, correction is performed by using techniques based on **Galois Fields**.  
However, conventional methods can only correct **single AWEs** due to field limitations.

Previous Work 3:

To further understand the differences between AN codes and conventional channel codes, we analyze a typical path with channel/memory and computations.

Channel codes are effective only during transmission and **lose error correction capability after arithmetic operations**.

In contrast, AN codes **retain fault tolerance during computation**. Moreover, AN codes can serve as substitutes for channel codes, and they can also be applied at the encoder/decoder stage for enhanced reliability.

Proposed Technique 1:

We propose an algorithm to find a suitable *A* value for correcting **double AWEs**.

* We first define the bit-width and multiply all values in the range by *A*
* Then, we inject single and double AWEs to generate 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 suitable for detecting and correcting single and double AWE in the specified bits.

We then build a 1-to-1 **lookup table** mapping all remainders to single and double AWEs.

This table consists of two parts:

1. A single AWE LUT

2. A double AWE LUT, which is **approximately N times larger** than the single AWE LUT.  
To address this area overhead, we propose a trade-off method described next.

Proposed Technique 2:

This figure shows how the relative size of A decreases as the bit width increases.

As bit widths grow, the overhead of A becomes negligible, making our technique highly suitable for **large-integer encryption systems**, such as RSA.

Proposed Technique 3:

To reduce area overhead, we propose a method based on linear decomposition. In this approach, a double arithmetic weight error remainder is decomposed into two single AWE remainders.

Then, using the pseudo code along with two LUTs — forward and reverse mappings, we correct the error through a for-loop search.

Since both two LUT match the size of a single AWE LUT, this method reduces area complexity by a factor of N, with a trade-off of increased time due to the for-loop search.

Experimental Results 1:

To evaluate fault tolerance, we injected **error-inducing noise** into multiplier circuits and simulated error rates under different correction techniques.

These graphs show results for 18-bits and 26-bits multipliers. Our proposed **double-AWE AN code (blue curve)** achieves the lowest error rate.

Experimental Results 2:

We compare three correction approaches:

* **Parallel (pure LUT-based)**
* **Sequential (software-only)**
* **Trade-off (hybrid approach)**

We assume a fixed total cost and evaluate area and time separately.

Area is synthesized using **Synopsys Design Compiler** on a 0.18µm process.

Execution time is measured using **Python on a Zynq-7000 platform**.

Since**Parallel** time cost and **Sequential area cost** are difficult to measure directly, we estimate these values through analytical approximation.

Experimental Results 3:

To visualize the differences, we normalize the area and time data using **Min-Max scaling** and plot the cost curves.

Both Parallel and Sequential approaches show **exponentially growing costs** in at least one cost.

In contrast, the Trade-off approach shows **balanced, linear growth** in both area and time.

This demonstrates that the Trade-off method provides efficient fault tolerance with practical hardware and software overhead, making it ideal for **large-integer encryption like RSA**.

These are the references cited in our work.

Thank you for your attention~~