

# REMOTE SENSOR SECURITY THROUGH ENCODED COMPUTATION AND CRYPTOGRAPHIC SIGNATURES

Ph.D. Dissertation Defense

By

Kevin Hutto

Advisor: Prof. Vincent J. Mooney

School of Electrical And Computer Engineering  
Georgia Institute of Technology  
Atlanta, GA, USA

23 April 2024

# Outline

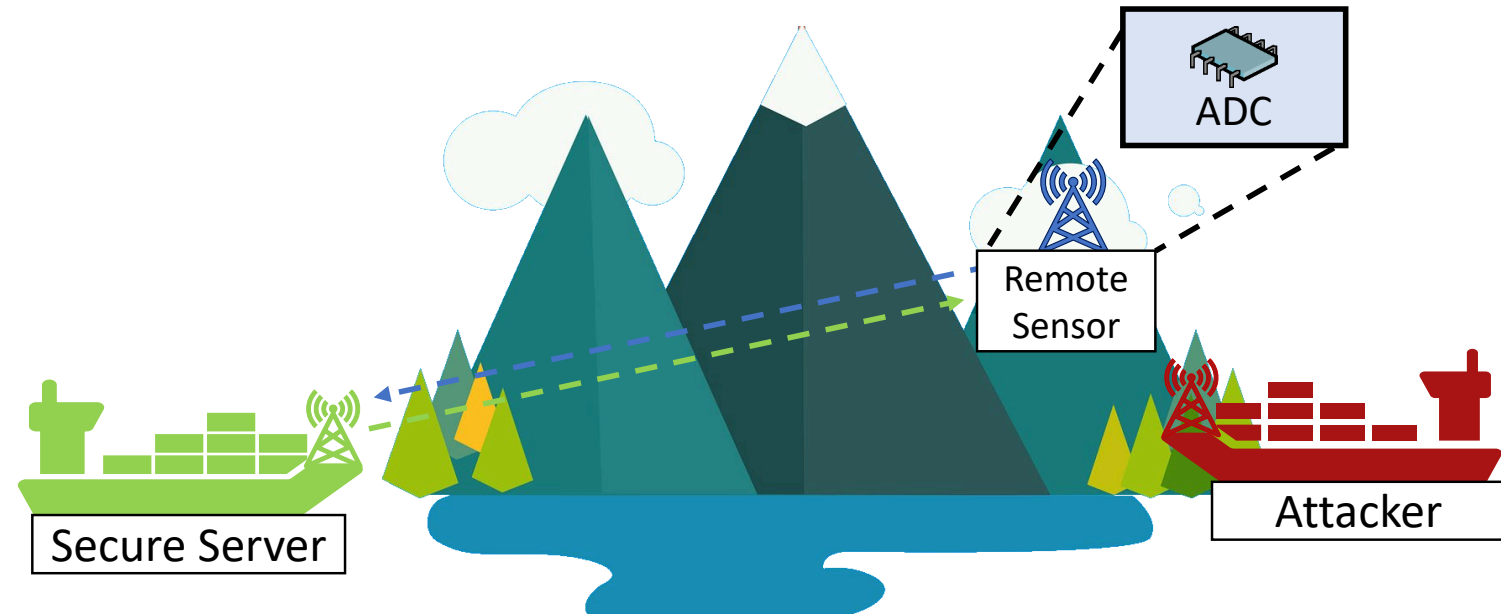
- **Introduction**
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Motivation

- Devices such as security cameras, sensors, and even household appliances today are increasingly connected to the internet, opening attack vectors to malicious entities
  - In 2017 attackers were able to exfiltrate data from a Casino by hacking into an internet-connected and controlled fishtank [1]
  - In 2021 attackers used default admin passwords to gain access to thousands of camera feeds running Verkada software [2]
  - In 2023 Akuvox smart intercoms were detected to have vulnerabilities allowing exfiltration of video feed data [3]
- Remote devices, such as IoT devices, have additional security considerations due to the potential ability of an adversary to gain access to the memory contents of a specific device

# Introduction

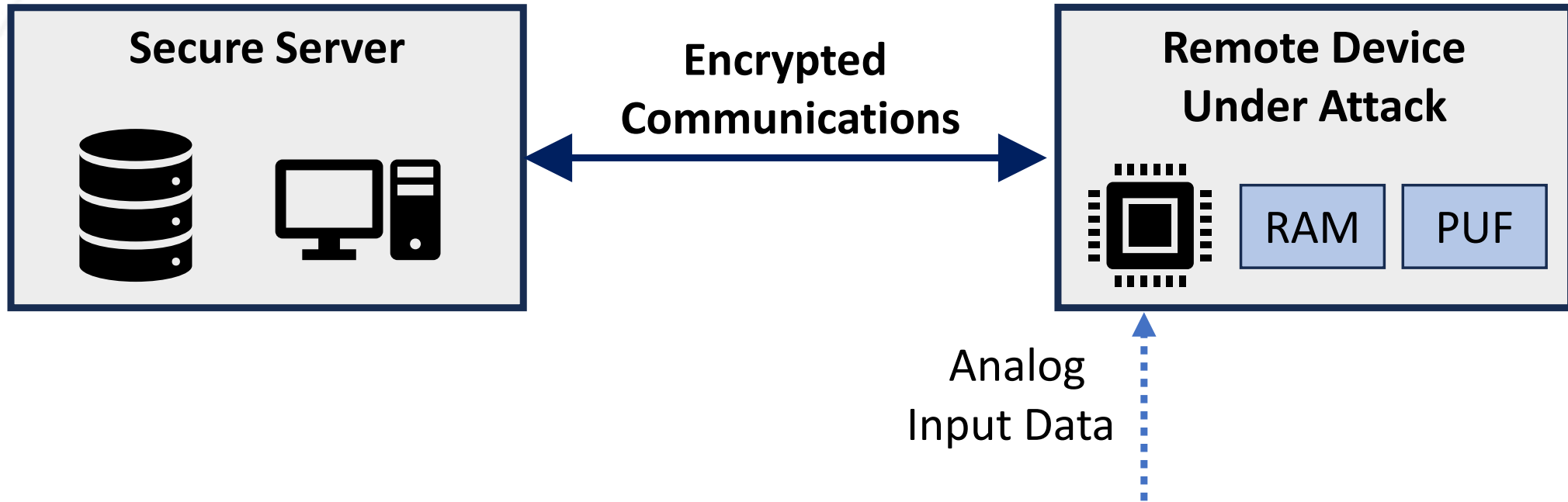
- Expedition collecting data using deployed remote sensors with analog-to-digital converters (ADCs) in an area of high physical vulnerability.
- The deployed remote sensors are communicating to a secure server utilizing standard networking protocols (e.g., TCP/IP)
- A non-participating party may steal and reverse engineer some of the remote sensor devices without detection, possibly leaking all on-chip and off-chip RAM, including, but not limited to, SRAM, Flash, DRAM, etc.



# Outline

- Introduction
- **Research Overview**
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Research Overview

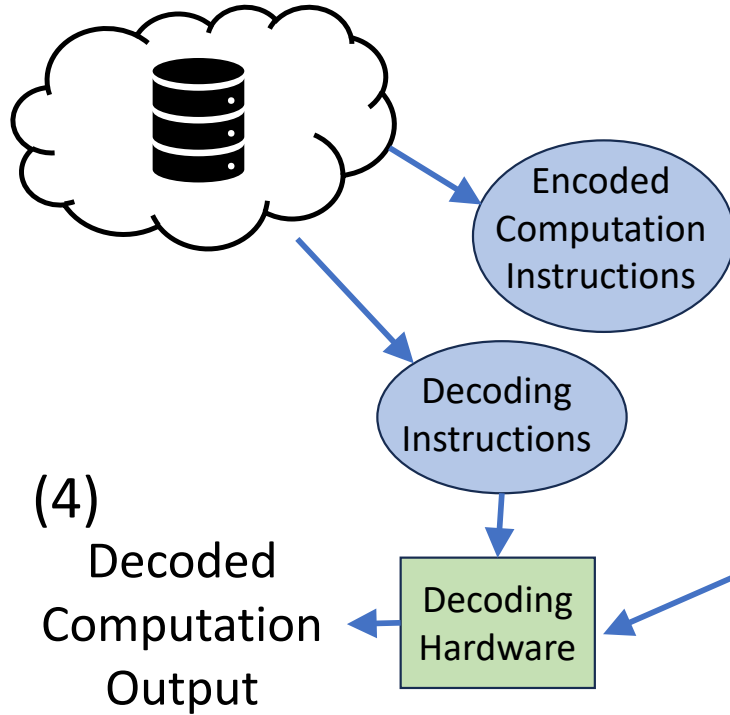


1. Security-Enhanced Analog-To-Digital Converter
2. Utilization of the ADC for a Privacy Homomorphism
3. Physical Unclonable Function (PUF) Based Authentication for Delivery of Software and Firmware Updates

# Research Overview

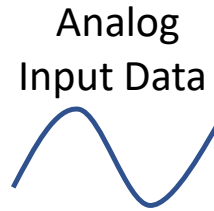
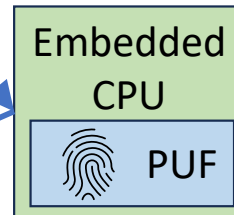
## Secure Server

(1)

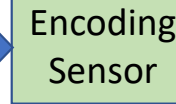


## Deployed Device

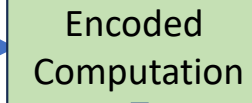
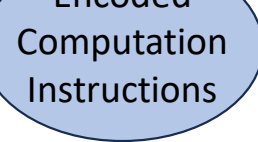
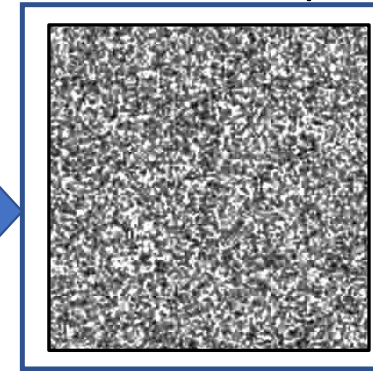
(5)



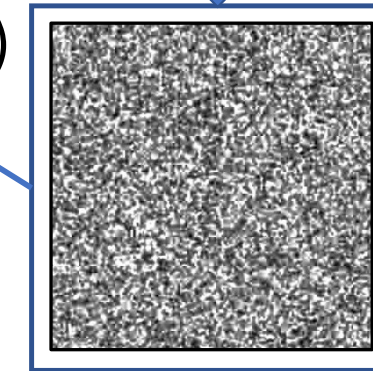
(2)



Sensor Output



(3)



Encoded Computation Output

# Outline

- Introduction
- Research Overview
- **Background and Prior Work**
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References



# Privacy Homomorphism

- In a seminal paper by Rivest, Adelman and Dertouzos in 1978 [4], the authors suggest the development of a cryptographic framework which would allow the performance of computation on encrypted data by a computer which cannot derive the unencrypted input data or the unencrypted output result of the computation

$$\phi(f^{-1}(\phi^{-1}(d_1), \phi^{-1}(d_2))) = f(d_1, d_2)$$

where  $\phi$  is the decoding function,  $\phi^{-1}$  is the encoding function,  $d_1$  and  $d_2$  are data,  $f$  is an operation in the decoded structure, and  $f^{-1}$  is an equivalent operation in the encoded structure

# Privacy Homomorphism

- The first published Fully Homomorphic Encryption (FHE), scheme was developed in 2009 by Craig Gentry, utilizing a multi-dimensional ideal lattice space [5]
- Numerous schemes have been proposed since Gentry's, with recent works, such as Brakerski-Gentry-Vaikuntanathan (BGV), mostly utilizing polynomial rings instead of lattices [6][7][8][9][10][11]
- Implementations such as HELib have large overheads compared to plaintext operations (potentially millions of times longer with state-of-the-art hardware such as BASALISC) [12][13][23]
- FHE schemes have large limitations such as no current support for division and square root directly (as specific operations) in publications and/or implementations

# Physical Unclonable Functions

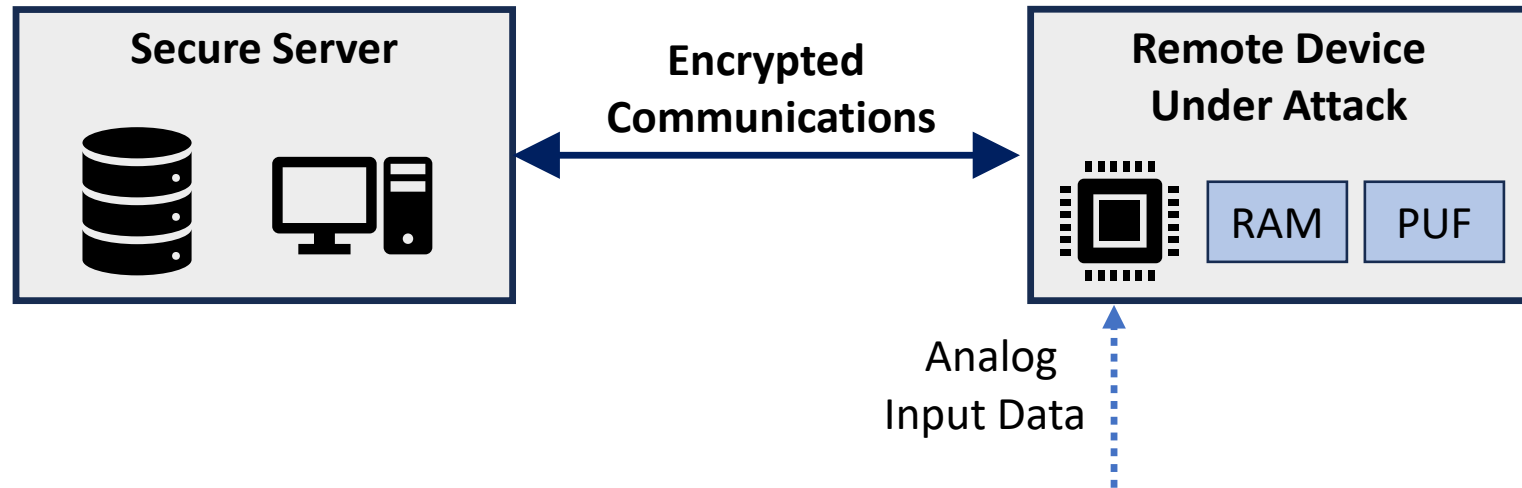
- A physical unclonable function (PUF) is a hardware security primitive that utilizes tiny manufacturing variations, typically in silicon, to produce a unique digital signature
- A PUF can function as a digital fingerprint and can be implemented on the same medium as digital circuits such as microprocessors
- Many different mechanisms have been proposed to create PUFs utilizing different variances affecting circuit features such as gate delay, resistance, and capacitance
- The most ubiquitous commercially available PUFs utilize the pseudorandom chance for an SRAM memory bit to initialize as either a '0' or '1'

# Outline

- Introduction
- Research Overview
- Background and Prior Work
- **Threat Model**
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Threat Model

- Two distinct entities: a secure server and a deployed device
- Assume adversary has potential to gain physical access to the deployed device
- With physical access, the attacker may be able to monitor on-chip memory (i.e., RAM, but not registers typically inaccessible to software)
- The attacker can perform simple data-collection techniques used for power side-channel analysis during operation [14]
- Assumed to have advanced hardware reverse-engineering techniques including decapsulating, delayering, and detection of logic values, but requires removal and destruction of the circuit
- We do not specifically address attacks on the secure server; rely on security provided by existing research



# Outline

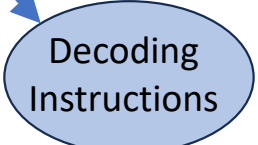
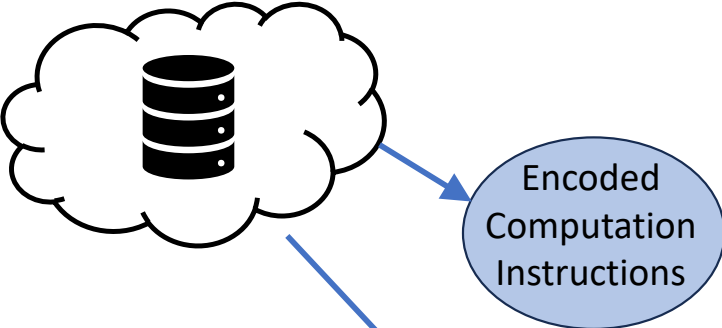
- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- **Security-Enhanced Analog-To-Digital Converter**
  - **Architecture**
  - Results
  - Security
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Security-Enhanced Analog-to-Digital Converter

Secure Server

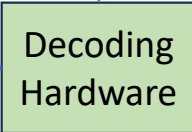
Deployed Device

(1)

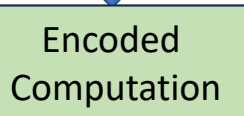
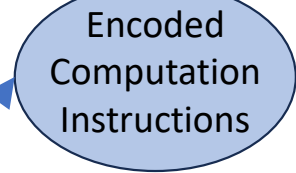
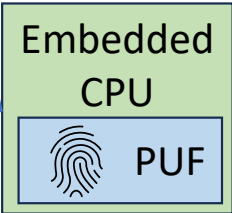


(4)

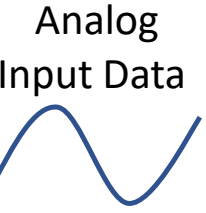
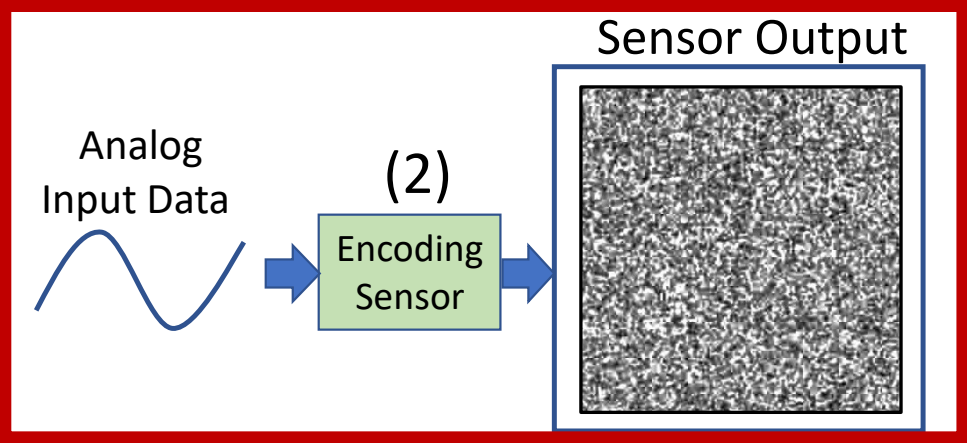
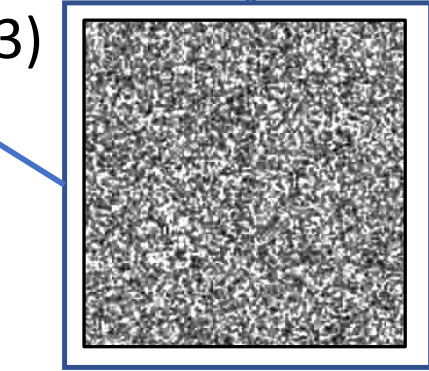
Decoded Computation Output



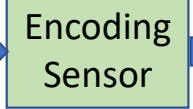
(5)



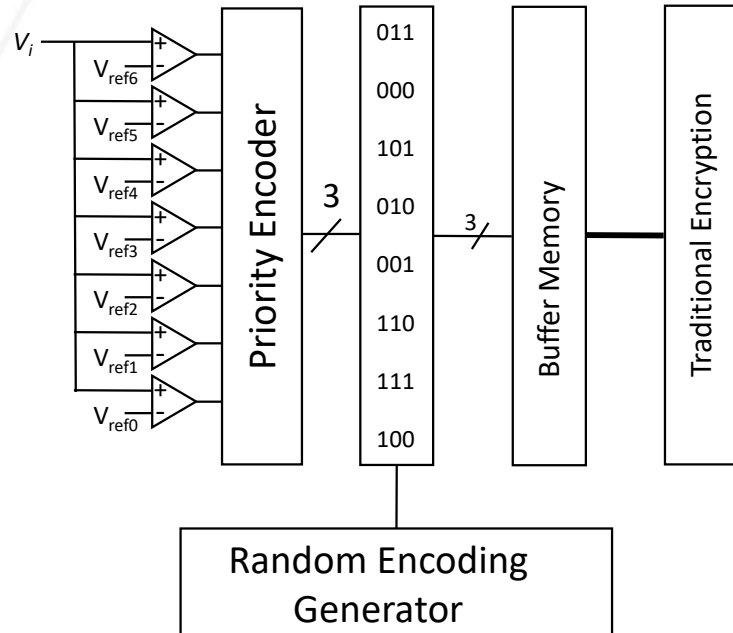
(3)



(2)

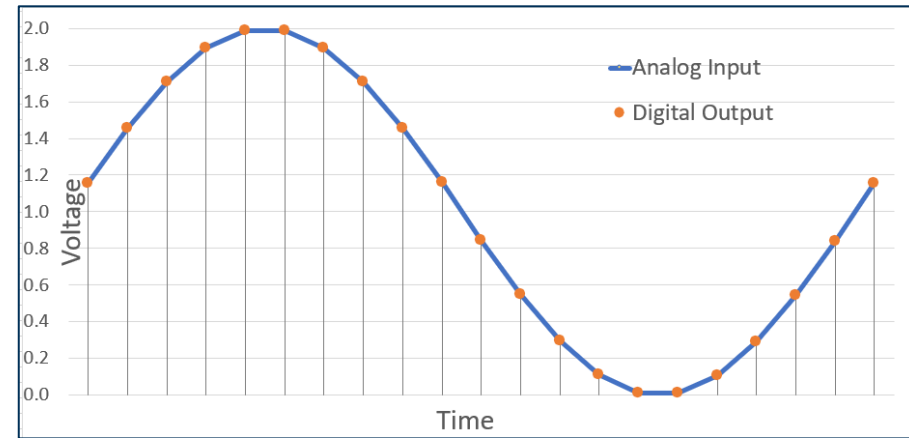


# Security-Enhanced Analog-to-Digital Converter Concept

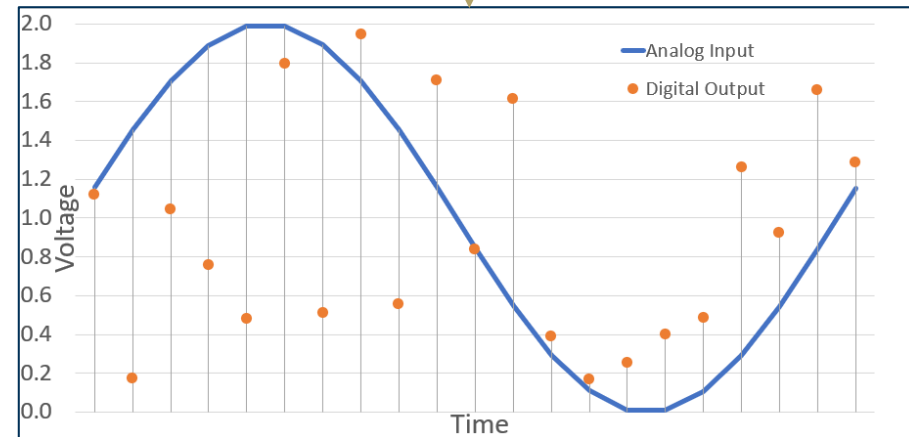


## Random Sensing with **RanCode** [15][16]

A **RanCode** ADC is implemented by directly encoding analog values as a function of both the analog input value and a pseudo-random permutation, shown here for a flash-ADC architecture



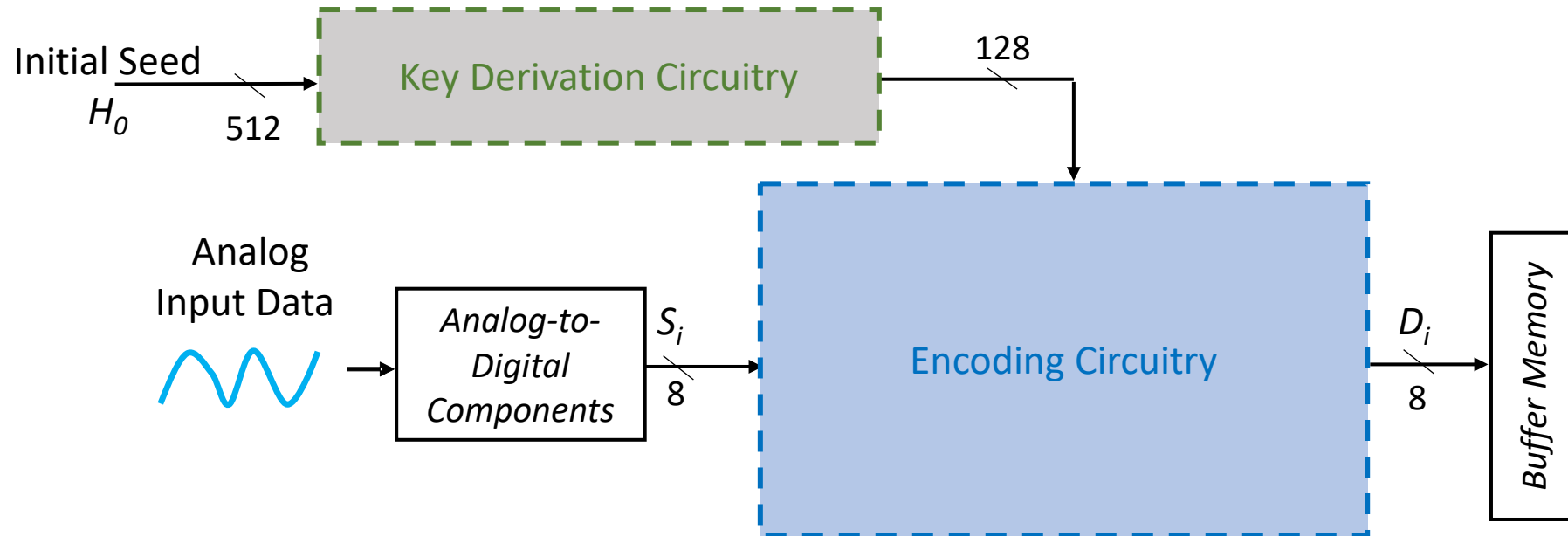
Standard ADC Encoding



Random ADC Encoding

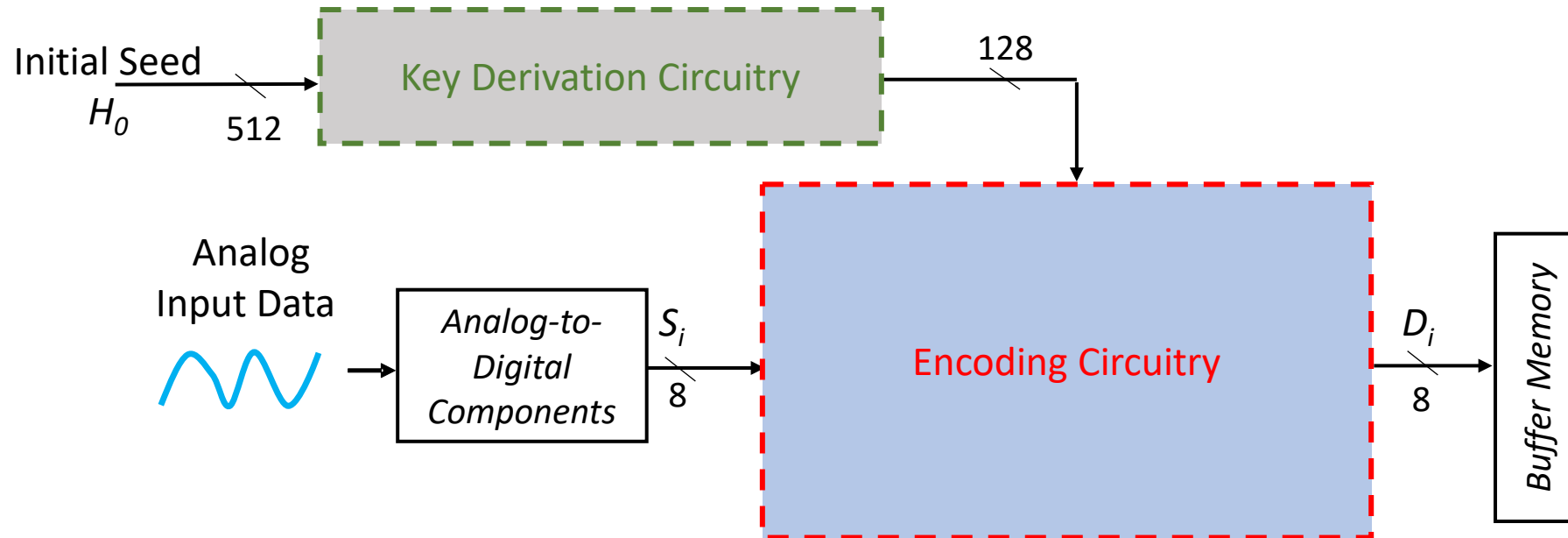


# RanCode High Level Architecture



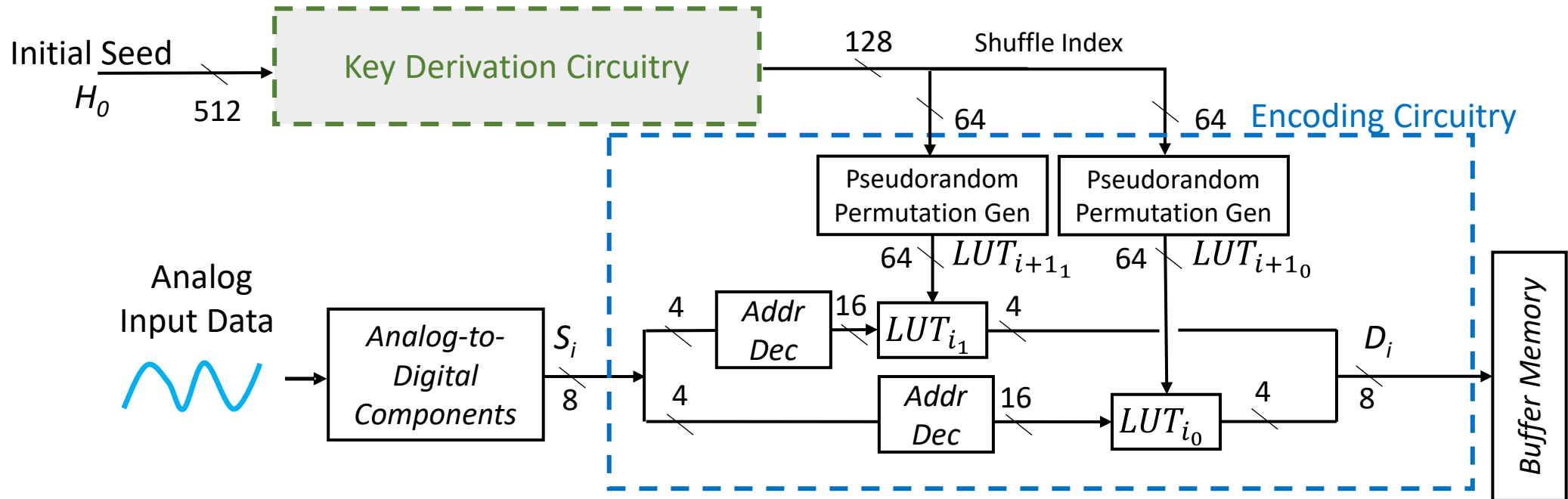
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data

# RanCode High Level Architecture



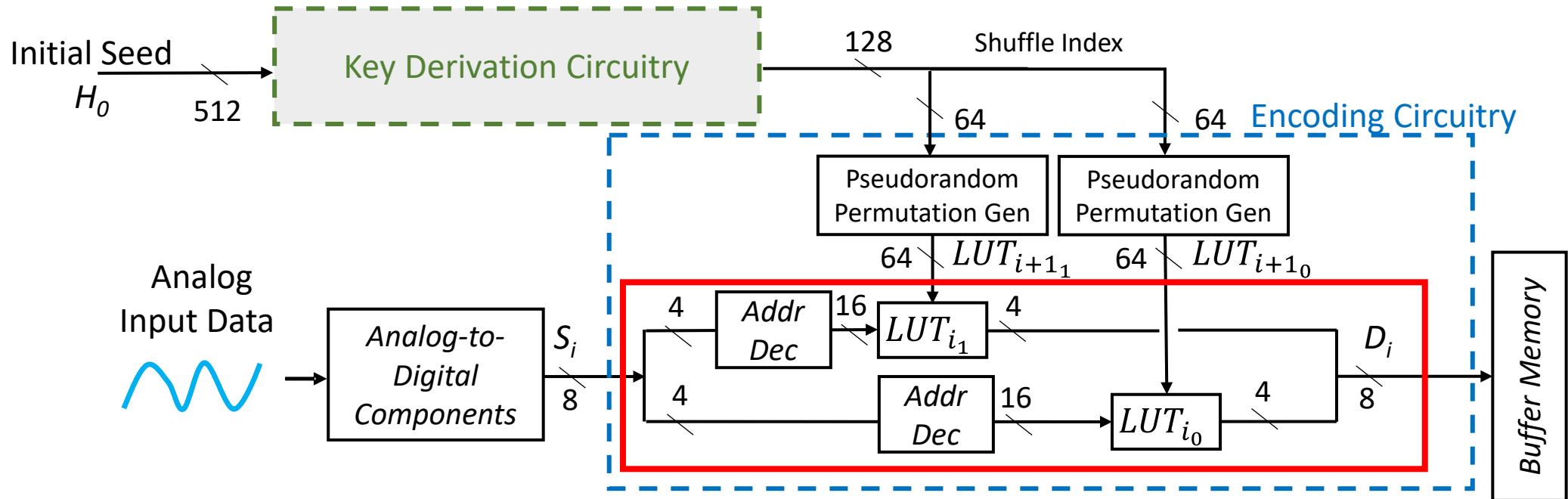
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data

# RanCode Encoding Circuitry



$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 LUT $_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Encoding Circuitry



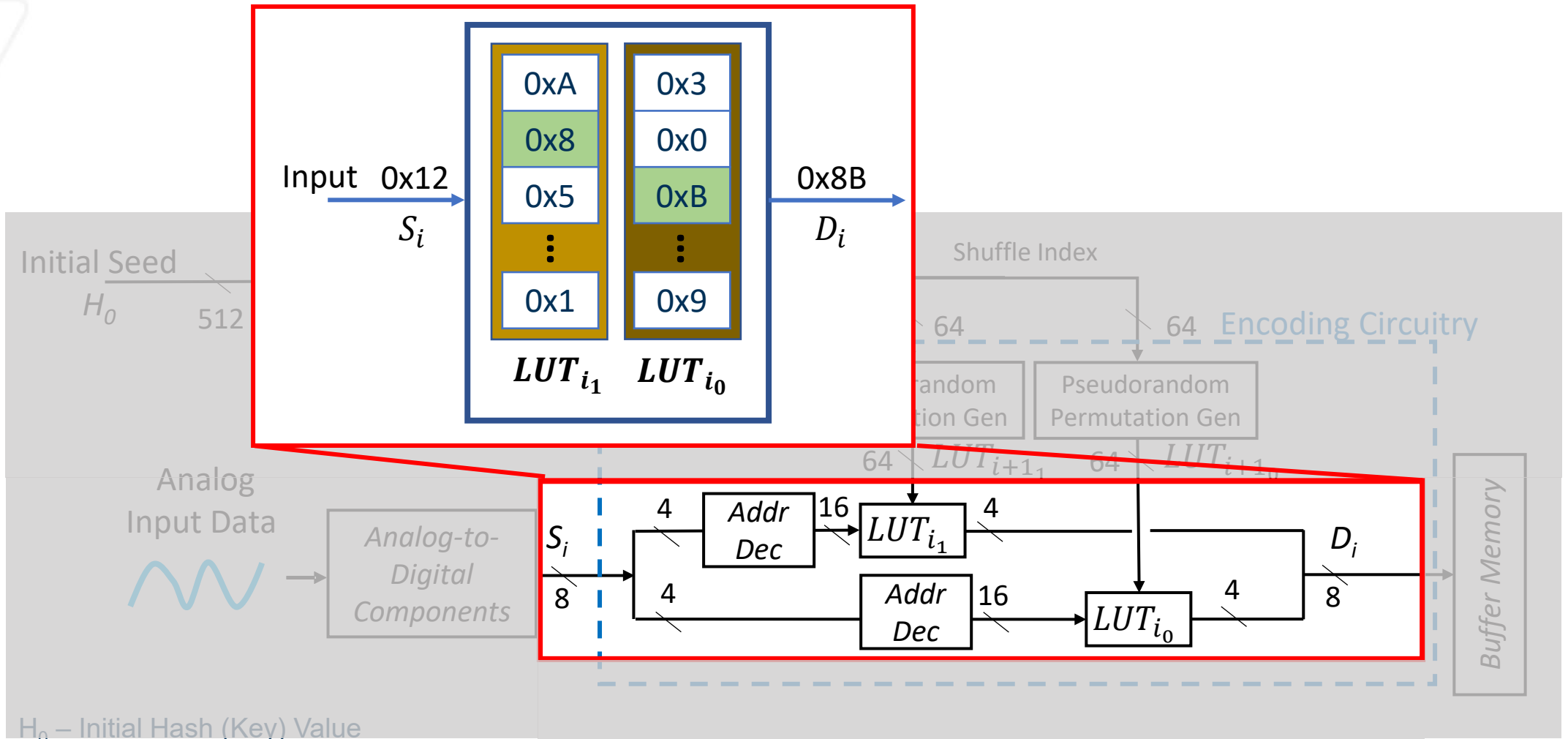
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 LUT $_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding







# RanCode Encoding Circuitry



$H_0$  – Initial Hash (Key) Value

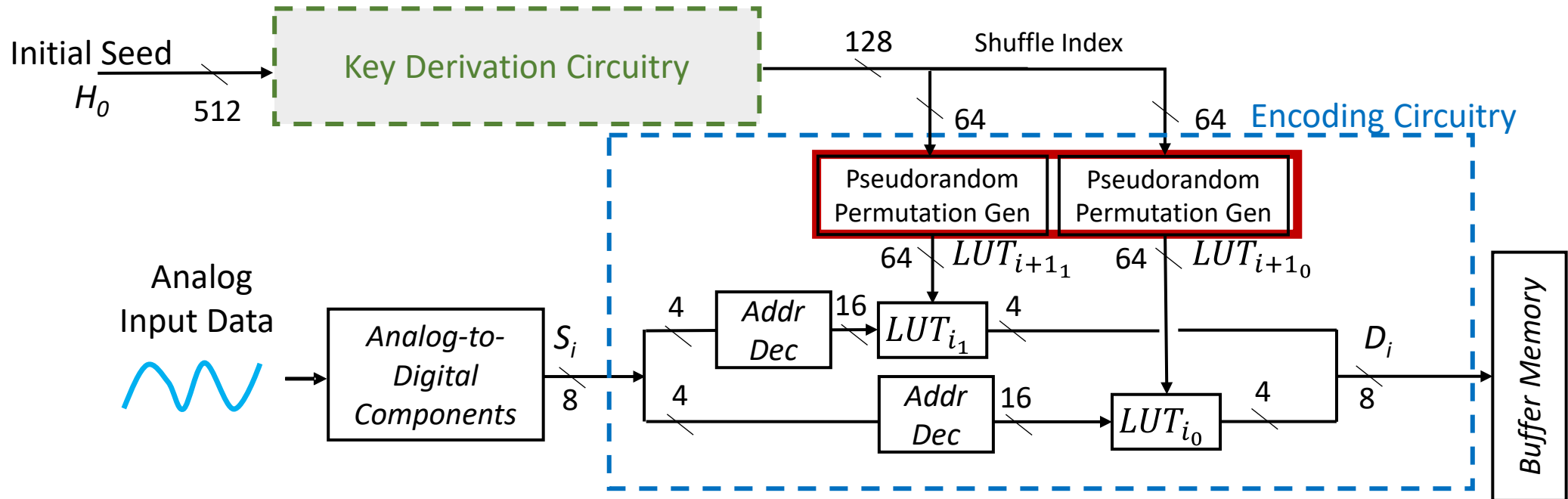
$S_i$  – Sensed Unencoded Data

$D_i$  – Encoded Sensor Output Data

$LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

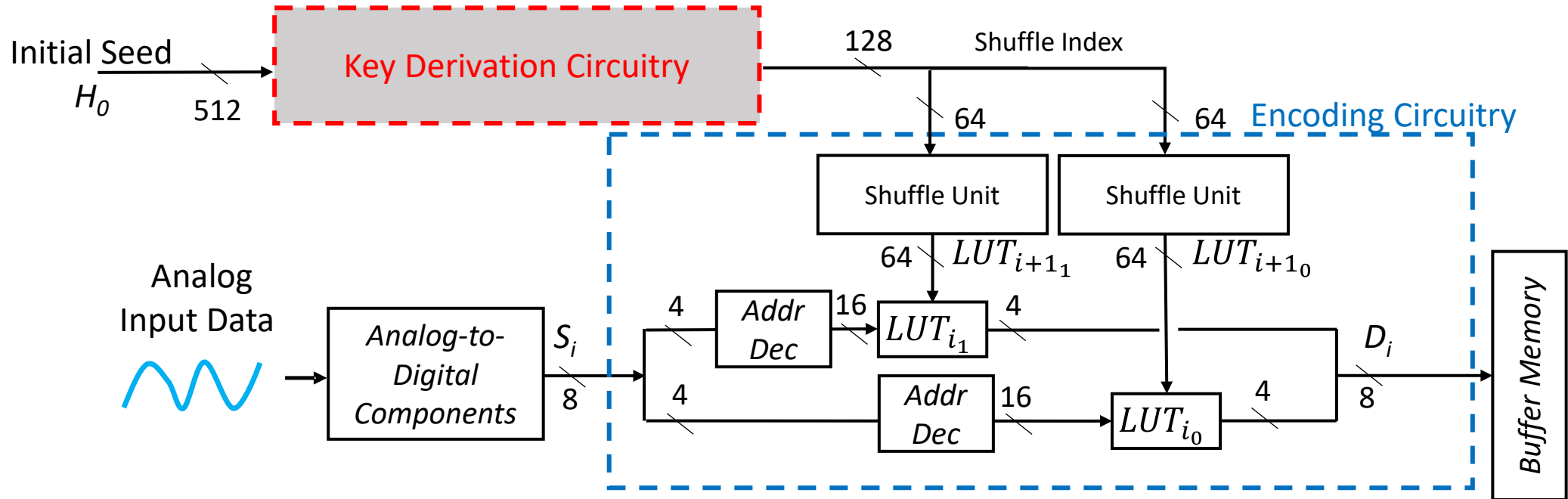


# RanCode Encoding Circuitry



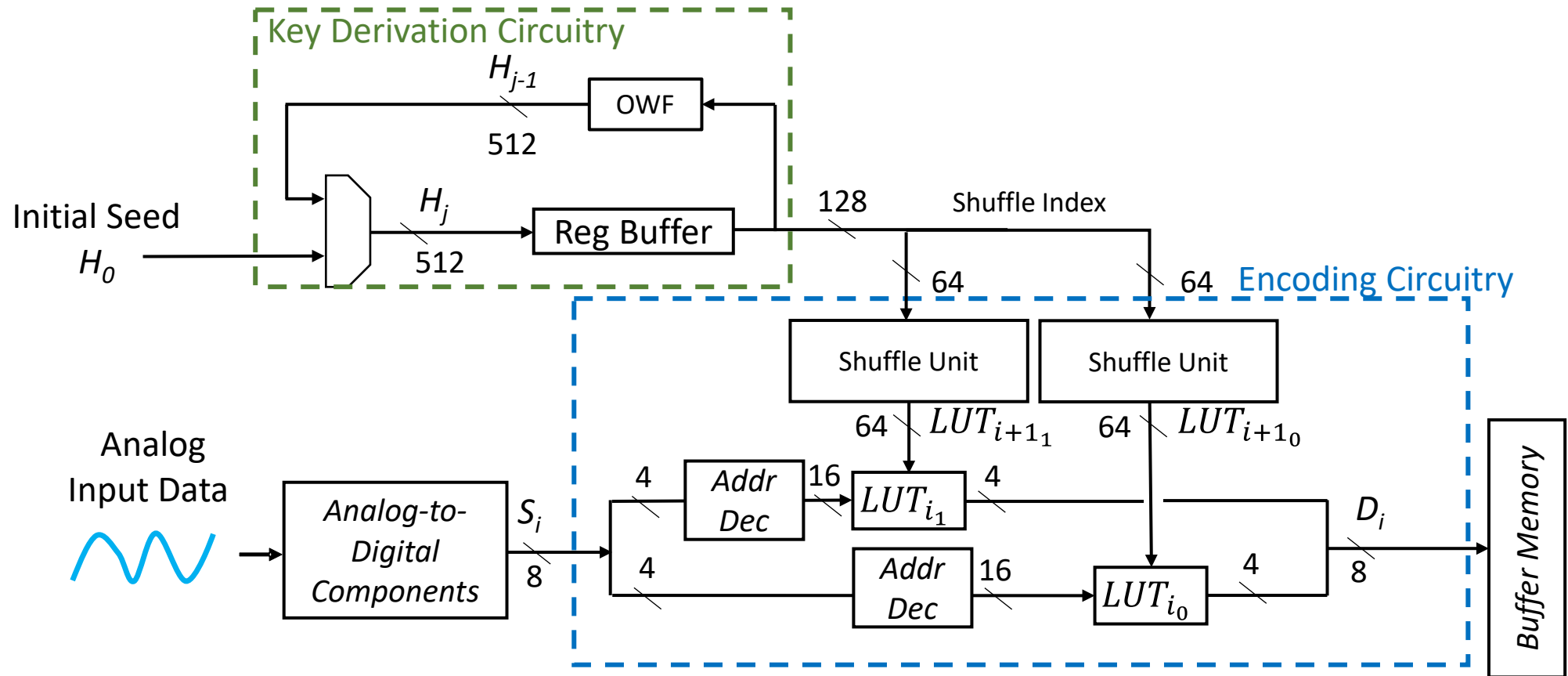
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Key Derivation Circuitry



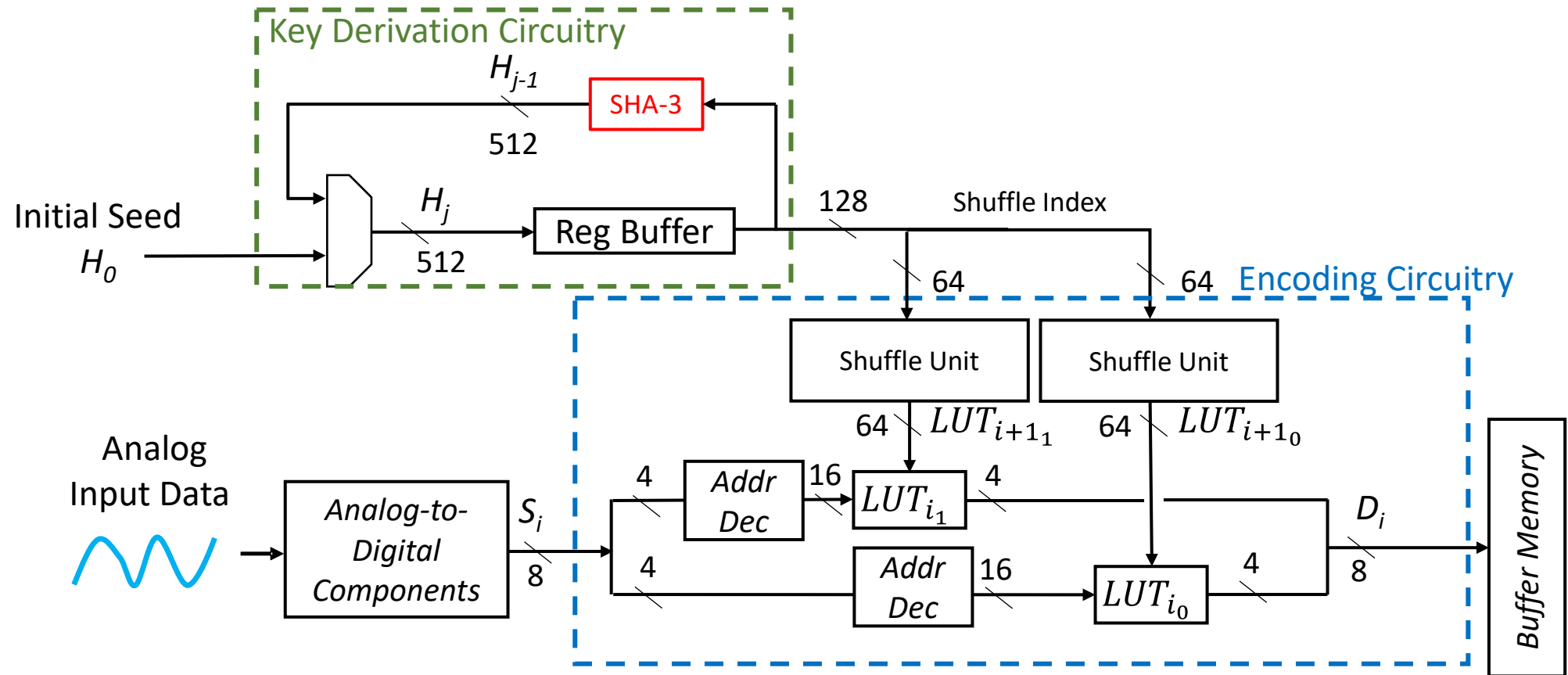
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Key Derivation Circuitry



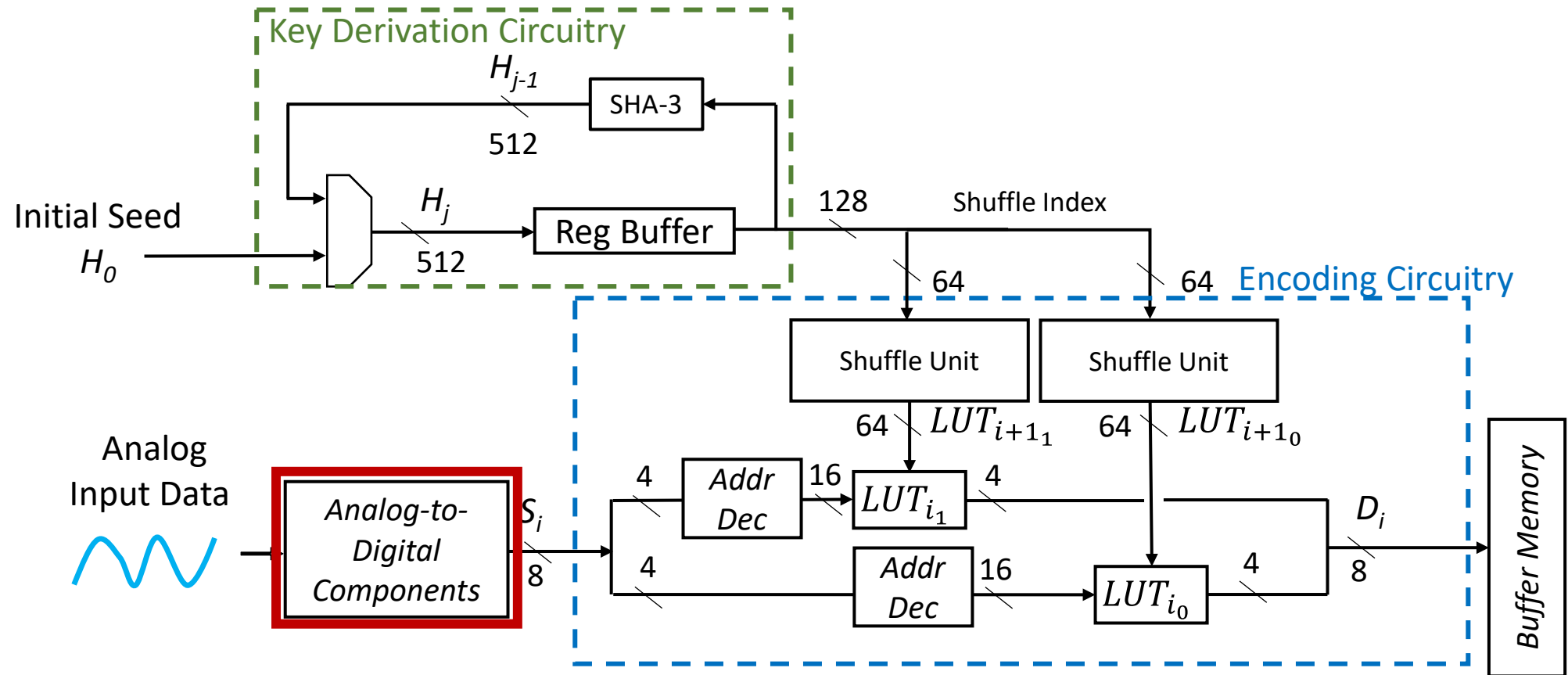
- $H_0$  – Initial Hash (Key) Value
- $S_i$  – Sensed Unencoded Data
- $D_i$  – Encoded Sensor Output Data
- $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Key Derivation Circuitry



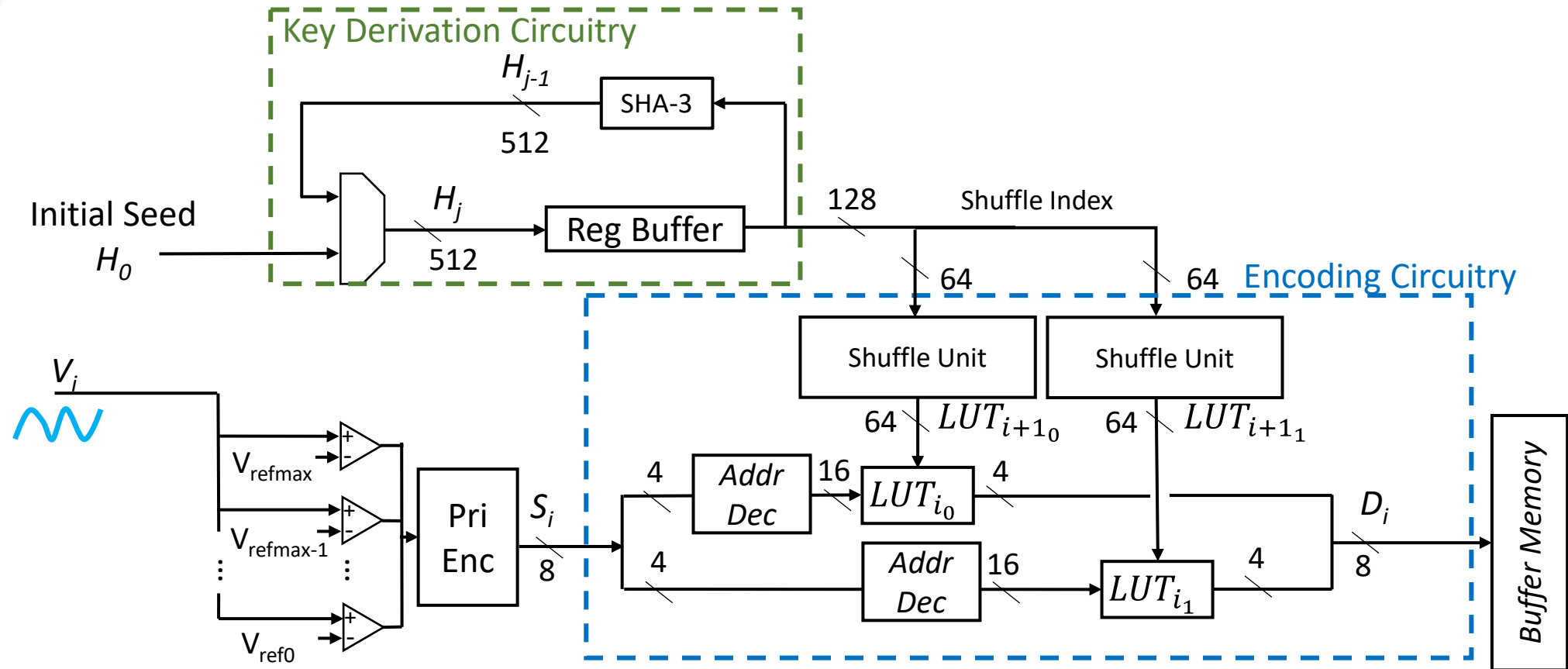
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Analog Circuit



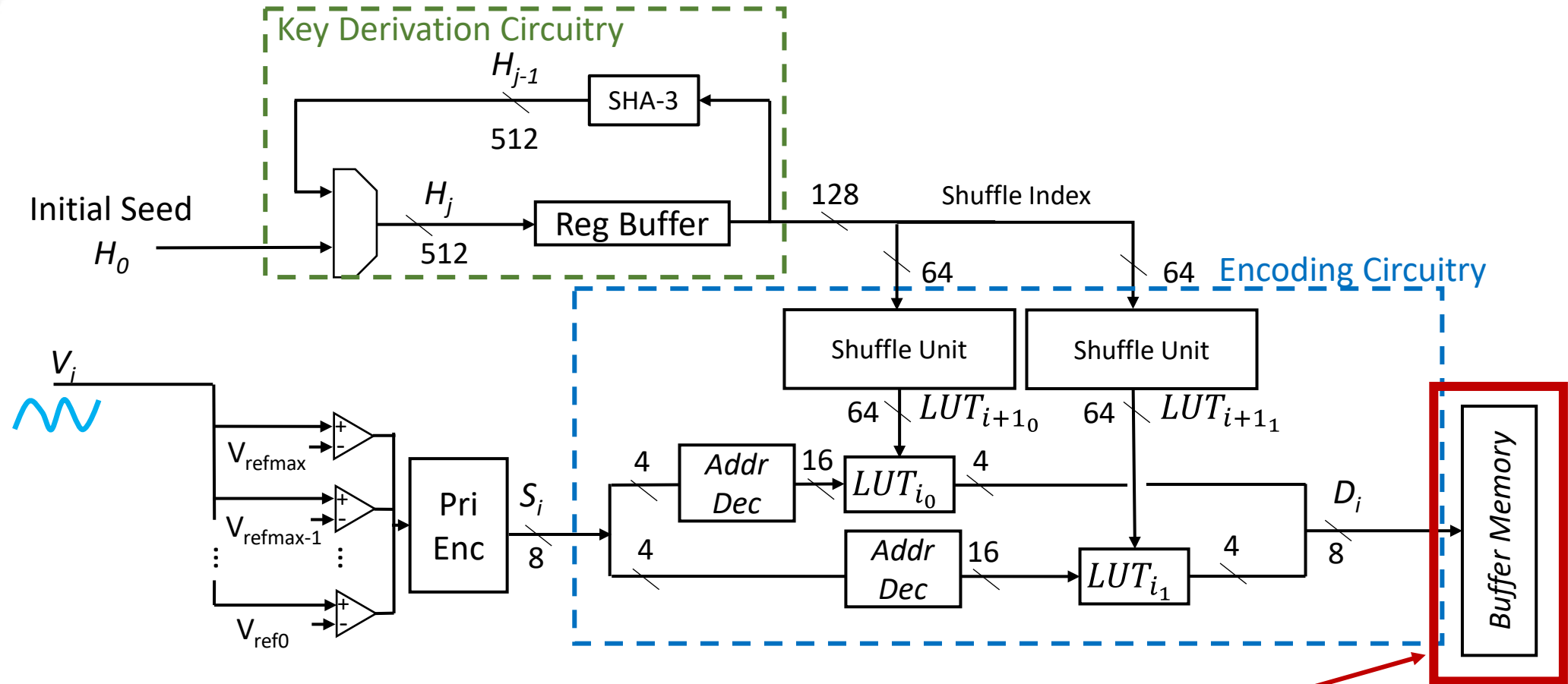
$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Analog Circuit



$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

# RanCode Analog Circuit



$H_0$  – Initial Hash (Key) Value  
 $S_i$  – Sensed Unencoded Data  
 $D_i$  – Encoded Sensor Output Data  
 $LUT_i$  – Look Up Table for the  $i^{\text{th}}$  Encoding

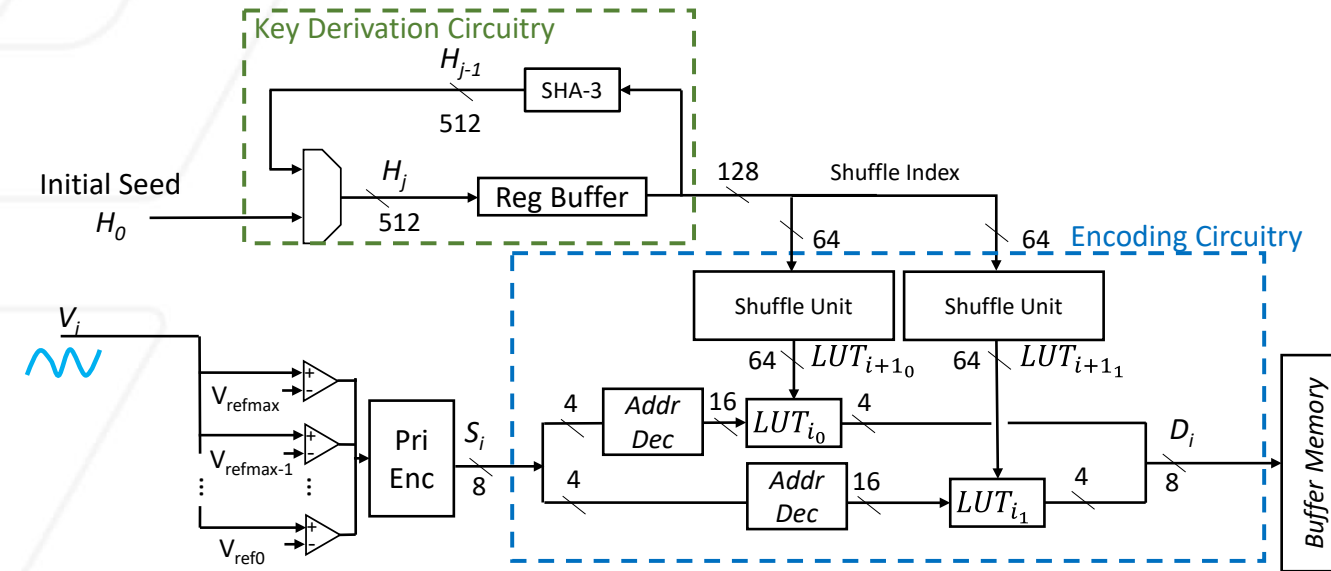
First location where encoded input data is stored in any form of memory

# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- **Security-Enhanced Analog-To-Digital Converter**
  - Architecture
  - **Results**
  - Security
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References



# Synthesis Results



- Synthesis conducted targeting the 28nm Cyclone V 5CSXFC6D6F31C6
- Three ADC architectures supported:
  - Flash
  - Successive-Approximation Register
  - Integrating
- Decoding architecture allows server to retrieve unencoded sensor values

Shuffle Unit Area Utilization

	Area (Square Microns)	Area (kGE)	Max Clk Freq
3-bit Shuffle Unit	1191	0.6	>550 MHz
4-bit Shuffle Unit	2794	1.5	>400 MHz
5-bit Shuffle Unit	8912	4.6	>150

Area and clock frequency results for Shuffle Unit synthesis.

FPGA Utilization for Flash 16-bit RanCode and Decode Circuits

Circuit	(A)	(B)	(C)	(D)
RanCode Encoding Circuit	4044	3694	44	>180
Decode Circuit	4088	3461	44	>200
All Shuffle Units	1346	0	44	>200
Single Shuffle Unit	418	0	11	>200

Integrating ADC FPGA Utilization

Circuit	LEs	Registers	DSPs	MHz
Encoding Circuit	2,499	2,486	22	157

SAR ADC FPGA Utilization

Circuit	LEs	Registers	DSPs	MHz
Encoding Circuit	2,538	2,568	22	157

# Outline

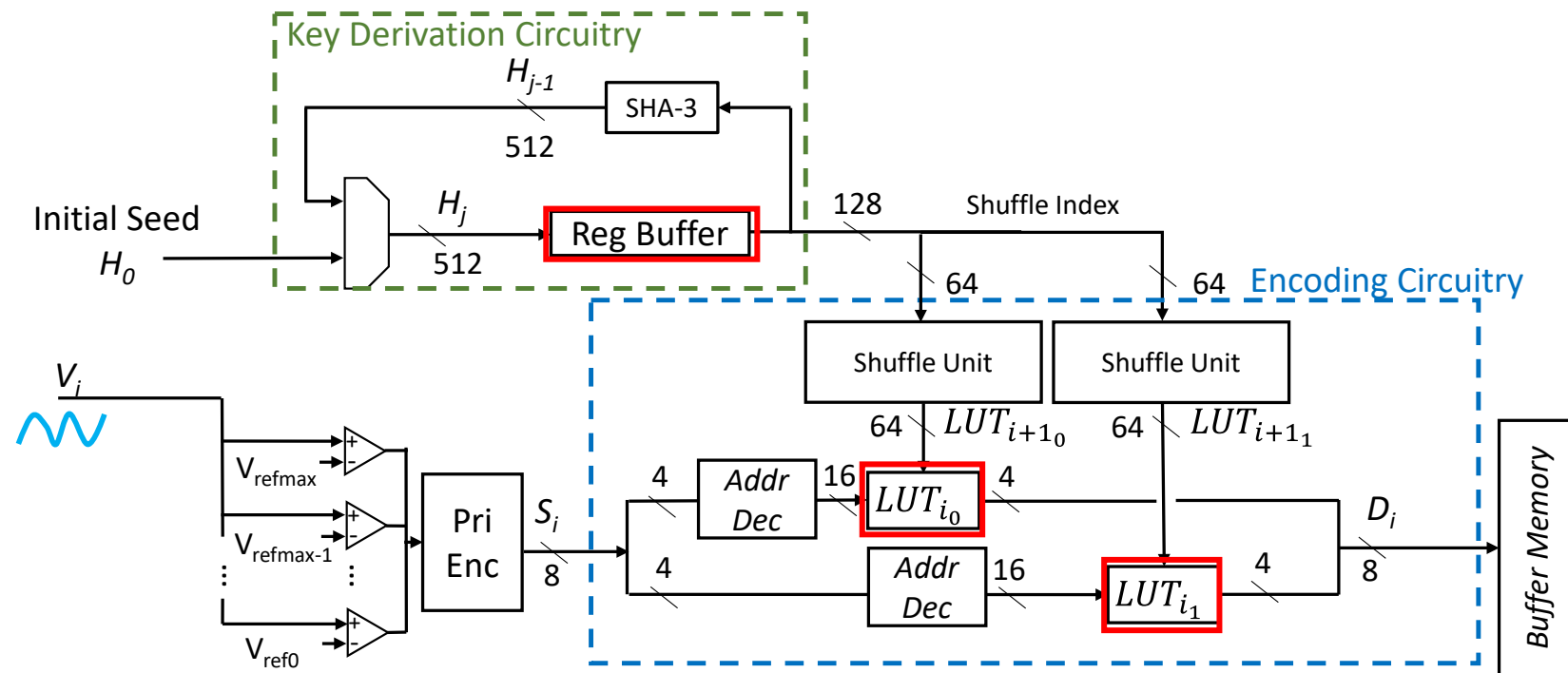
- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- **Security-Enhanced Analog-To-Digital Converter**
  - Architecture
  - Results
  - **Security**
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Security of the RanCode Circuit

- We are concerned with an adversary's ability to interpret the encoded data produced by the RanCode circuit
- We will show that without the knowledge of the RanCode key,  $H_j$ , the adversary has an exponentially small chance of successfully interpreting the data
- Keep in mind that our attack surface excludes on-chip registers including look-up tables; only the large on-chip memories such as SRAM cache blocks (including buffer memory to store the digital images) are provided to the adversary

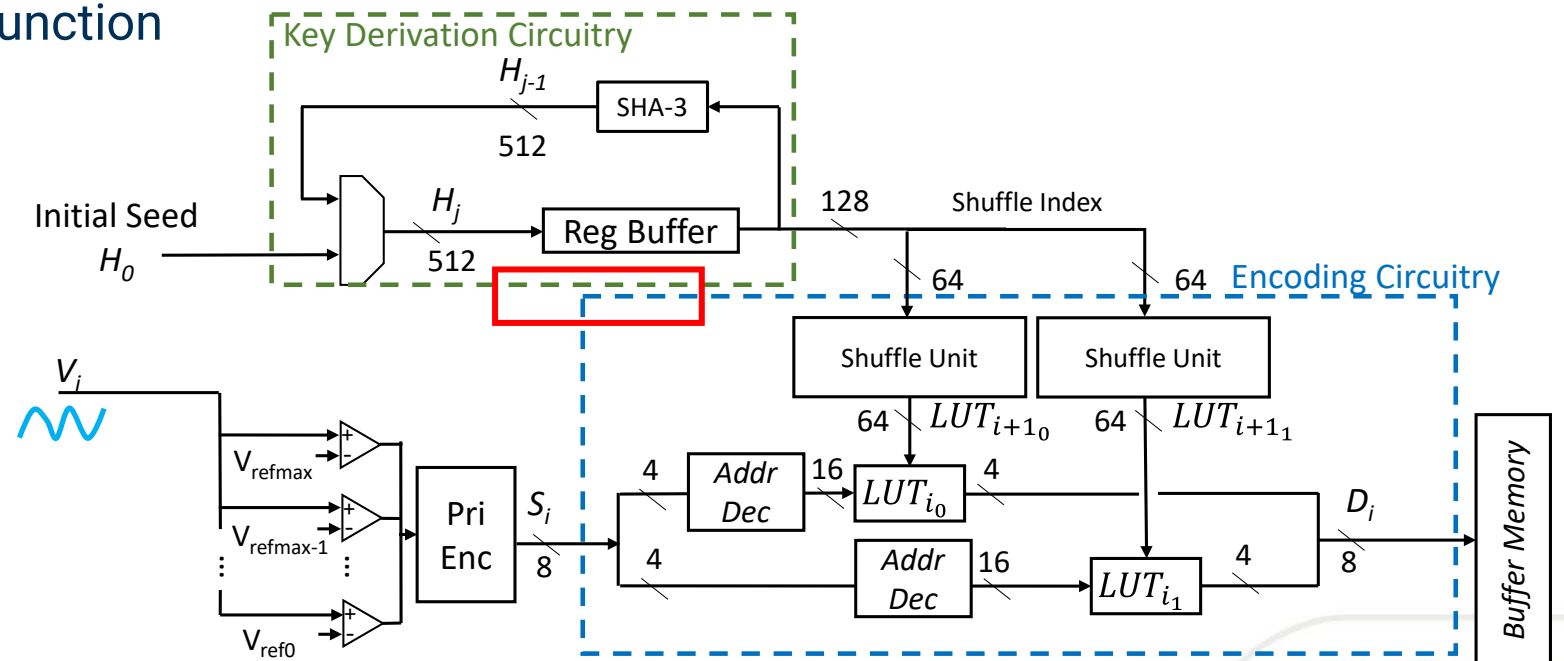
# Security of the RanCode Circuit

- To interpret encoded data the adversary has a few potential methods
  - Discover the key  $H_j$
  - Discover the mappings in each  $LUT_i$
- If the adversary can achieve the determination of a correct  $H_j$ , then determining  $LUT_i$  is trivial to achieve as the hardware architecture is known by the adversary.



# Security of the RanCode Circuit

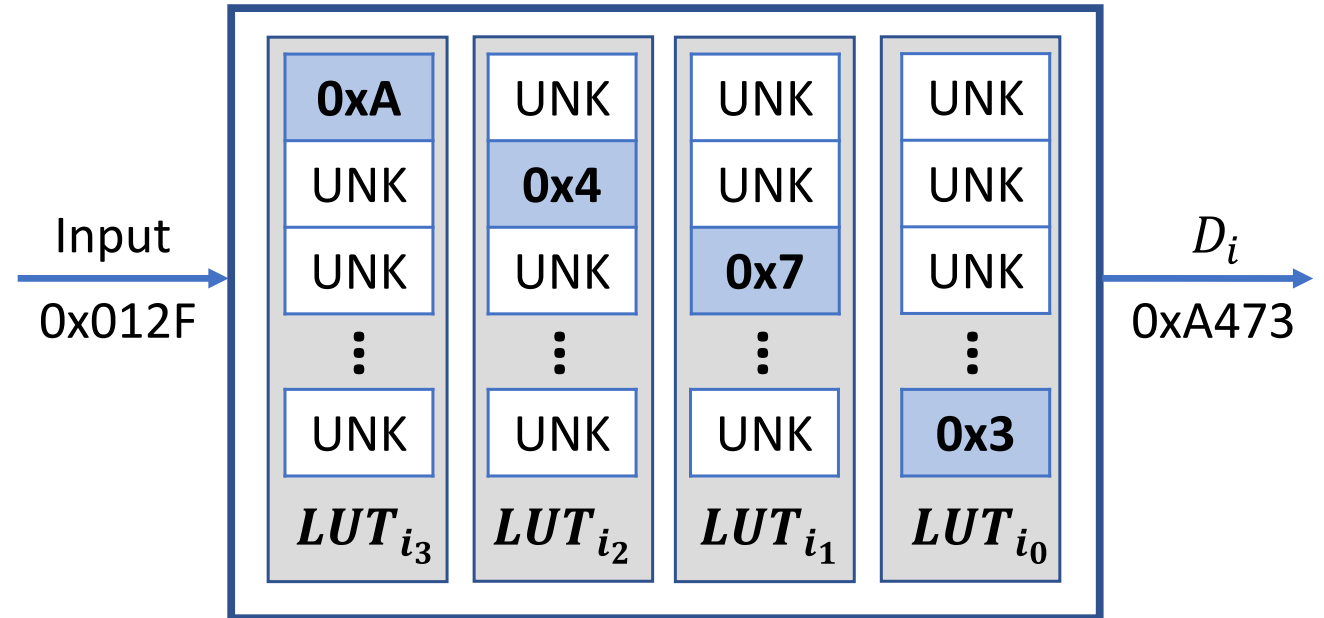
- Can  $H_j$  be determined?
- For  $H_j$ , no read mechanism is provided, and furthermore, there is no device interface which allows access of any intermediate value from the encoding method
- After encoding (permuting), once the key  $H_j$  is overwritten by SHA-3, there is no way on the remote device's microchip to go backwards in time
  - SHA-3 implements a one-way function
- By utilizing SHA-3, each subsequent encoding (derived from a SHA-3 output) is effectively independent of the previous encoding
- $H_j$  is 512 bits long – infeasible for brute force search





# Attempted Attack

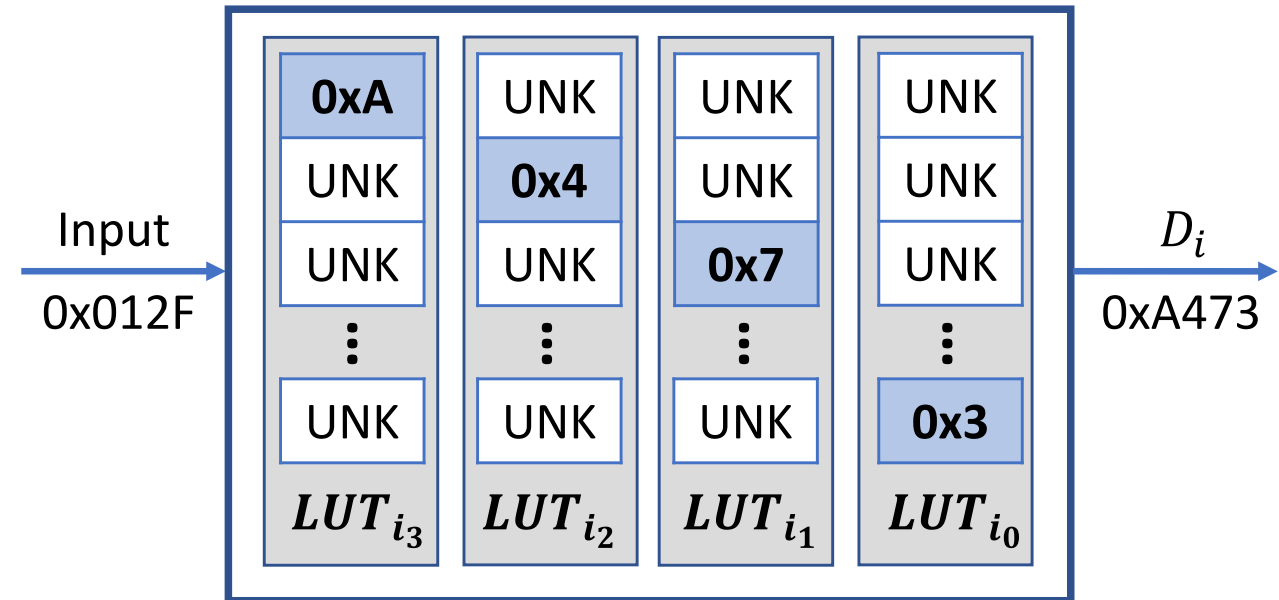
- Attacker cannot brute force all  $H_j$  values, so an attempt is made to prune potential values without evaluating each  $H_j$
- Attacker can see the encoded outputs stored in RAM on the device
- Attacker attempts to correlate plaintext inputs to outputs by guessing input values
  - E.g., at night the pixels may be black
- Assume the input guesses made by the adversary are accurate for a limited number of samples such that the exact voltage encoded by the ADC is known
- For the guessed values, the attacker inverts the permutation function to retrieve  $H_j$  values



UNK: Values unknown to attacker

# Attack Cont.

- If the input guesses are correct, a subset of the LUT values is determined
- With the known LUT subsets, the adversary attempts to determine future complete mappings
- The known input to output mappings allow the adversary to determine one location in each of the four  $LUT_{i_j}$  values
- The adversary knows any hash value input to the shuffle circuitry which does not result in the discovered partial mapping must be wrong
- How far does this partial known mapping lead to a reduced search space?

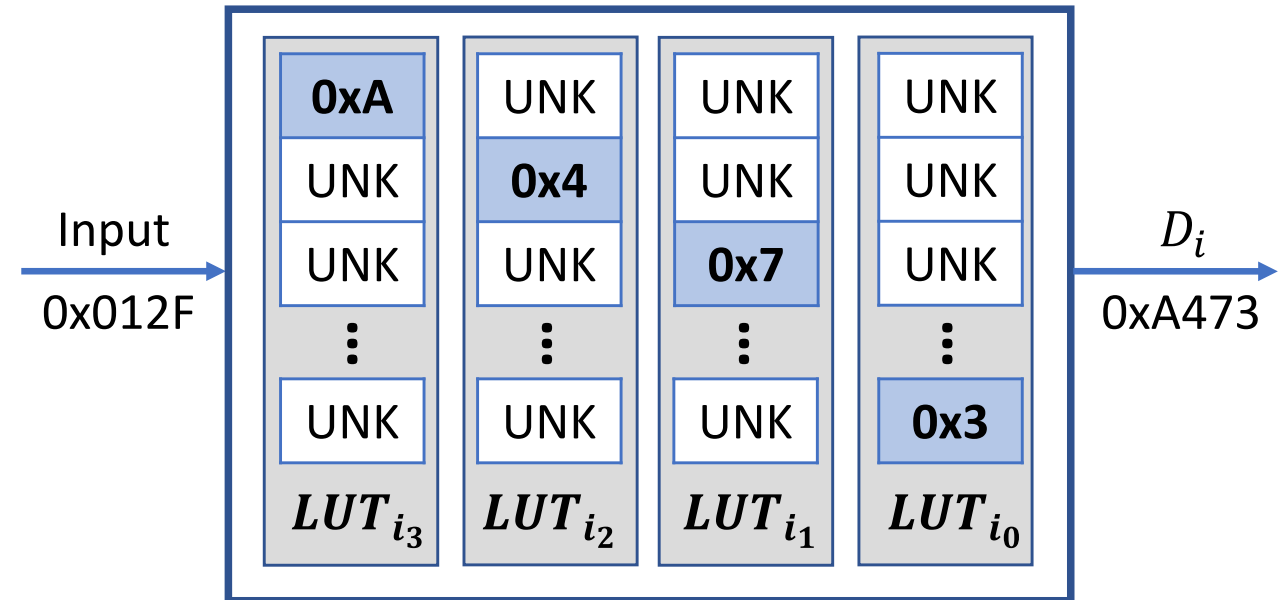


UNK: Values unknown to attacker



# Attack Cont.

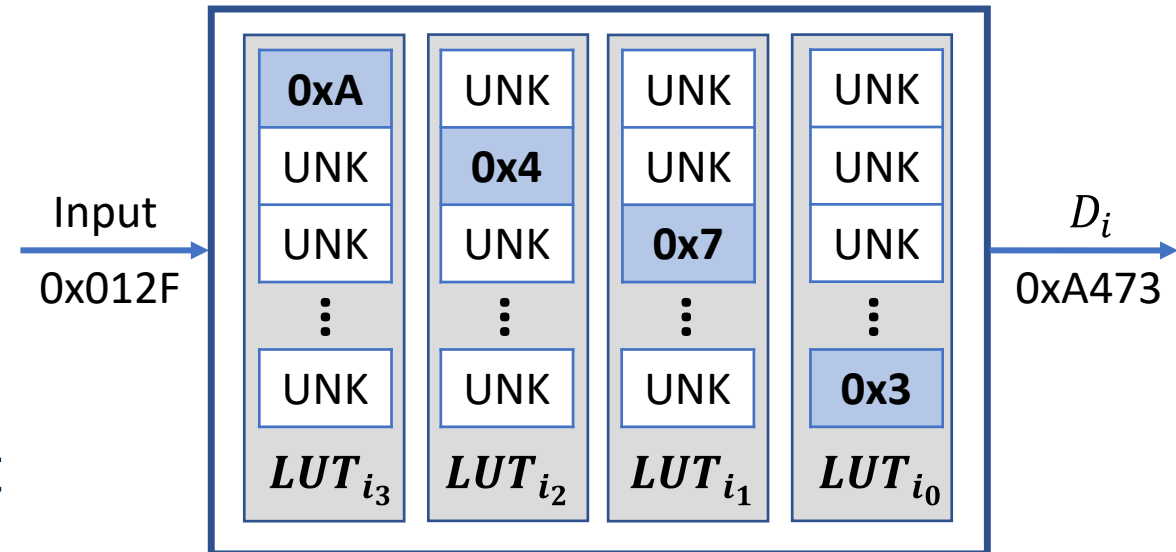
- The Pseudorandom Permutation (Knuth Shuffle Algorithm [17]) is reversible
  - Given a known arrangement of set elements and a known output, it can be easily determined what index was used in the algorithm
- With only a partial knowledge of the shuffle output, a subset of possible indices can be disregarded



UNK: Values unknown to attacker

# Attack Cont.

- The partially known LUT values remove 1/16 of the possible  $2^{512}$   $H_j$  values, leaving  $2^{496}$   $H_j$  values
- Each of the  $2^{496}$  possible  $H_j$  values will provide a permutation which maps the known four locations in the four LUTs
- Only one of these possible  $H_j$  values matches the actual internal value of  $H_j$
- If the wrong  $H_j$  value is used, the follow-on calculated values  $LUT_{i+1}$  from  $H_{j+1}$  will not match the actual RanCode mappings for subsequent encoding

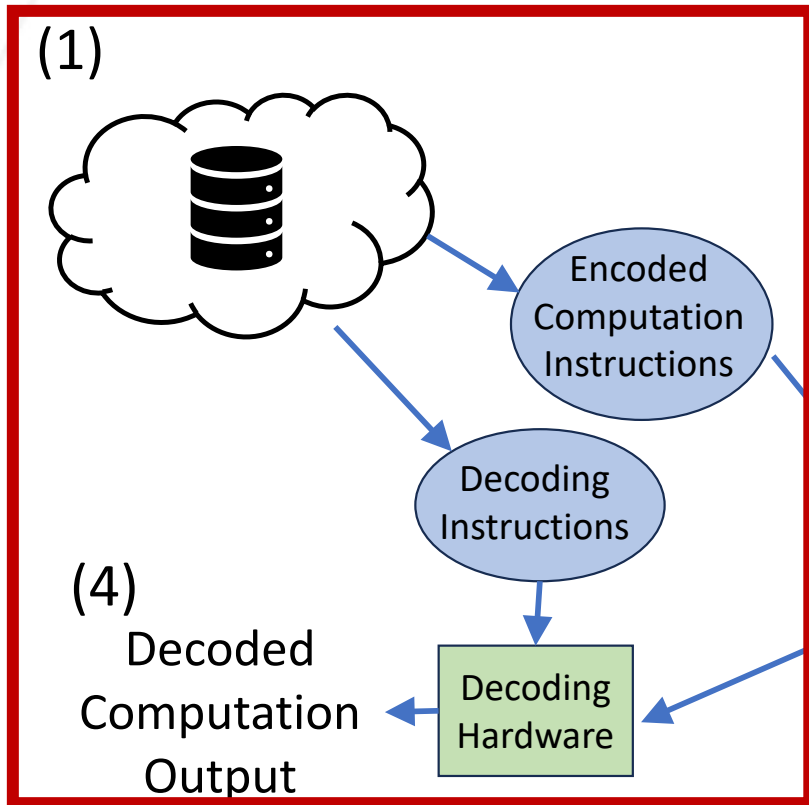


# Outline

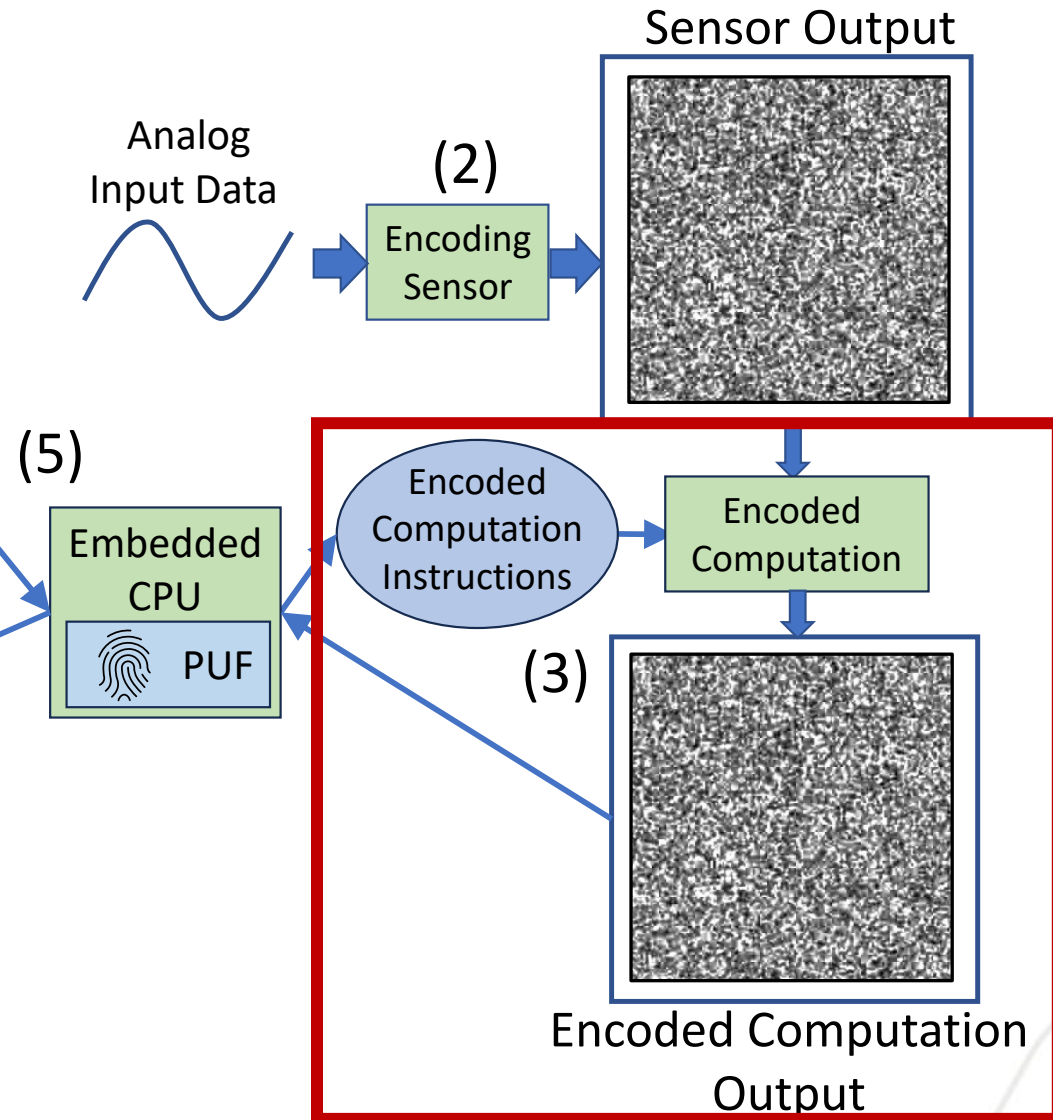
- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- **Implementing a Privacy Homomorphism With a Security-Enhanced ADC**
  - **Privacy Homomorphism Scheme**
  - Scheme Extensions
  - Architecture for Edge Detection
  - Security
  - Comparison to Fully Homomorphic Encryption
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Implementing a privacy homomorphism with RanCode

## Secure Server



## Deployed Device



# Recall What is a Privacy Homomorphism

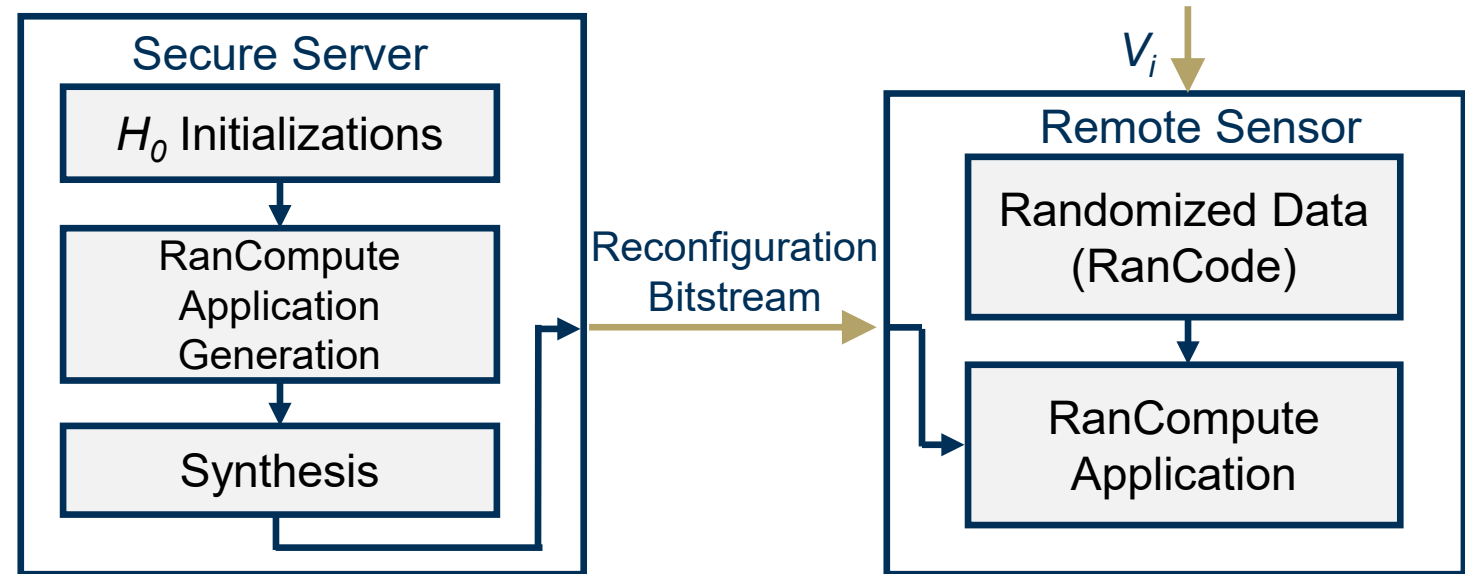
- In a seminal paper by Rivest, Adelman and Dertouzos in 1978 [4], the authors suggest the development of a cryptographic framework which would allow the performance of computation on encrypted data by a computer which cannot derive the unencrypted input data or the unencrypted output result of the computation

$$\phi(f^{-1}(\phi^{-1}(d_1), \phi^{-1}(d_2))) = f(d_1, d_2)$$

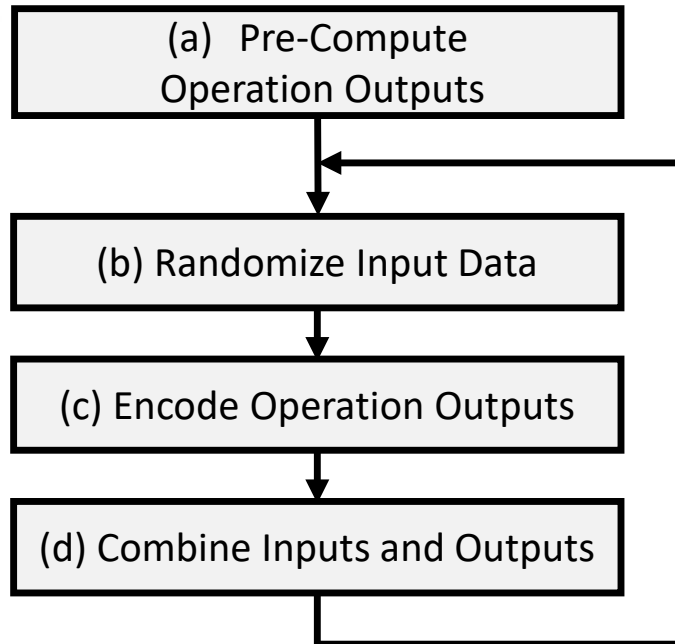
where  $\phi$  is the decoding function,  $\phi^{-1}$  is the encoding function,  $d_1$  and  $d_2$  are data,  $f$  is an operation in the decoded structure, and  $f^{-1}$  is an equivalent operation in the encoded structure

# RanCompute, a Privacy Homomorphism

- The developed technology is referred to as **RanCompute** [18][19][20]
- **RanCompute** implements a privacy homomorphism
- Two entities are involved
  - The secure server
  - The remote device
- The secure server contains the  $H_0$  initialization values for remote sensors implementing a **RanCode** architecture
- The remote sensor shown has two components, a **RanCode** implementation and a **RanCompute** application on an FPGA

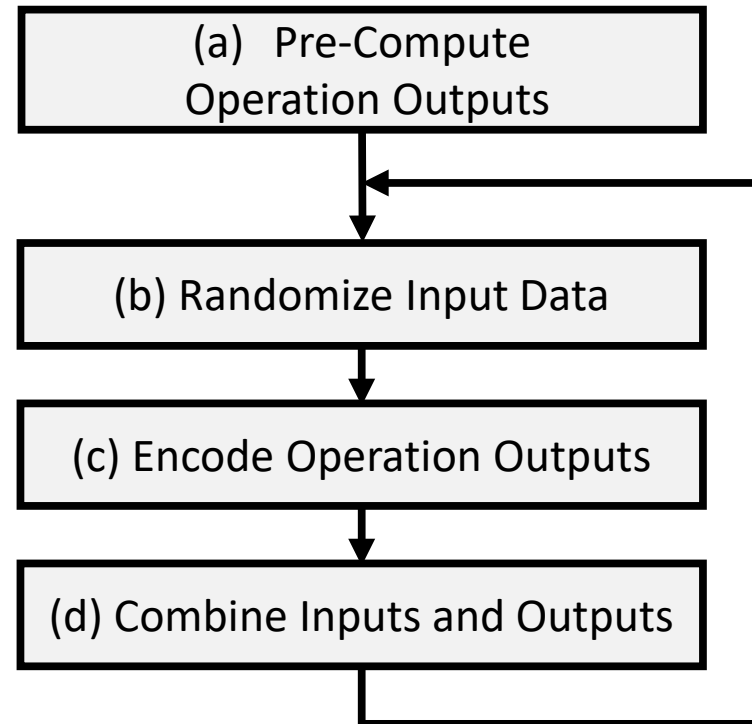


# RanCompute Privacy Homomorphism



- **RanCompute** creates two tables to facilitate a privacy homomorphism:
  - **Computation Table** - Used by the device.
    - Contains mappings from **RanCode** encoded inputs to encoded computation outputs
  - **Decode Table** - Used by the server.
    - Contains mappings from encoded computation outputs to unencoded outputs

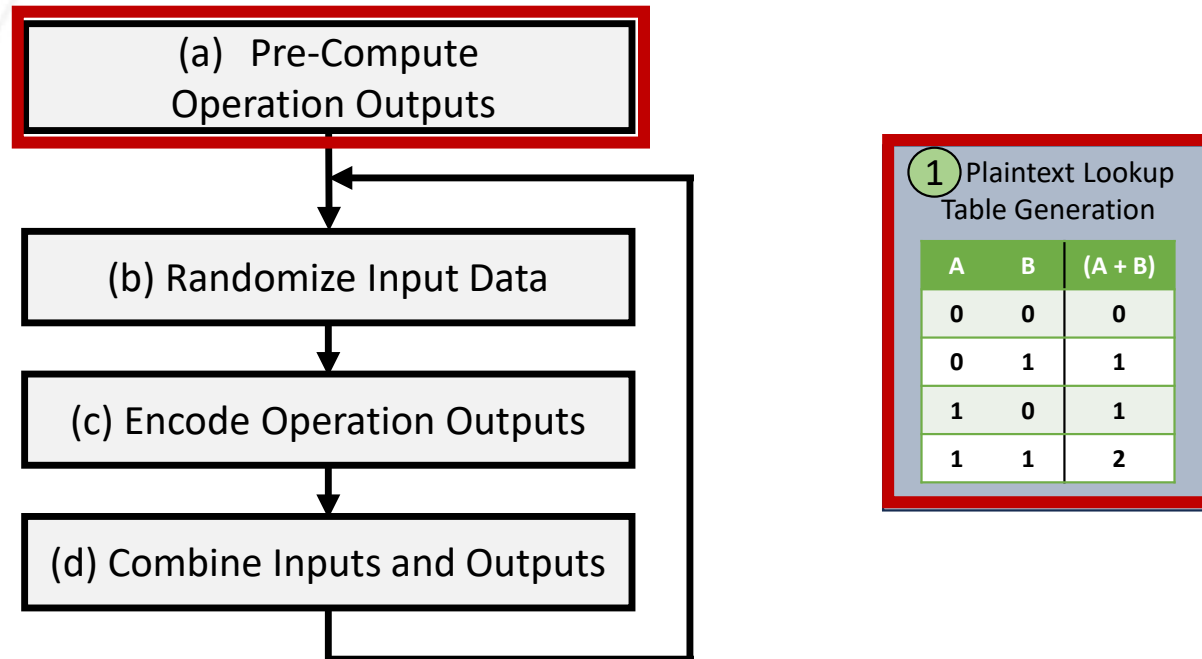
# RanCompute Privacy Homomorphism



$$\phi \left( f^{-1} \left( \phi_1^{-1}(d_1), \phi_2^{-1}(d_2) \right) \right) = f(d_1, d_2)$$

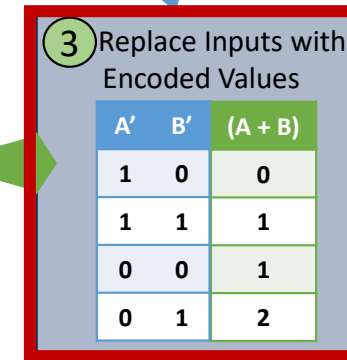
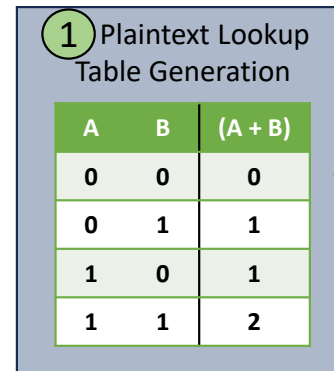
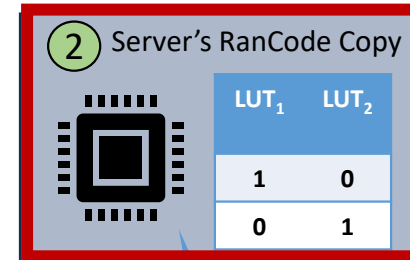
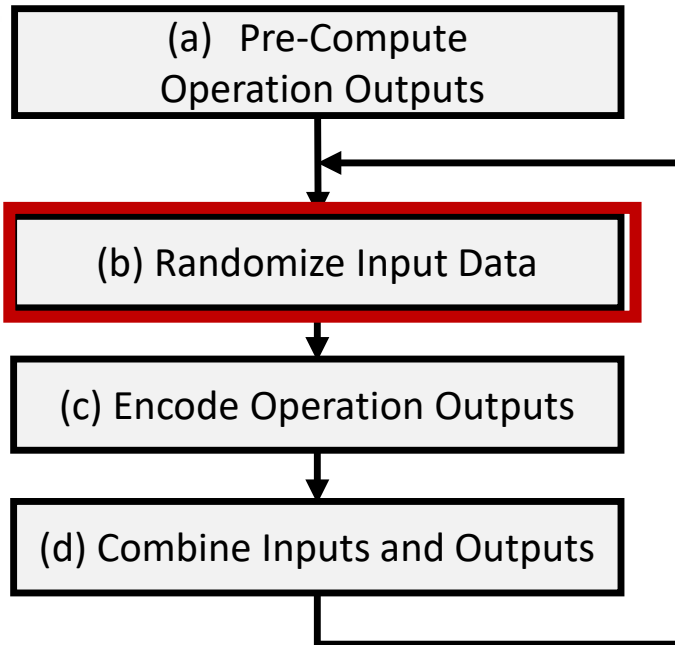


# RanCompute Plaintext Table Generation



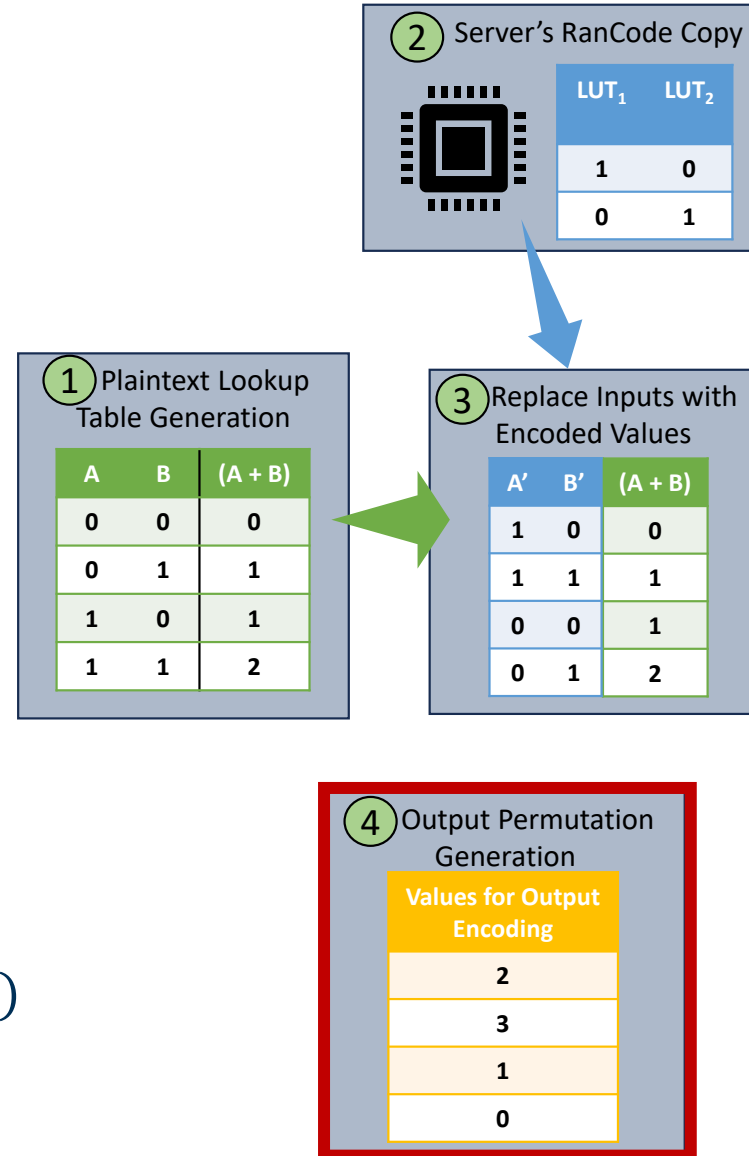
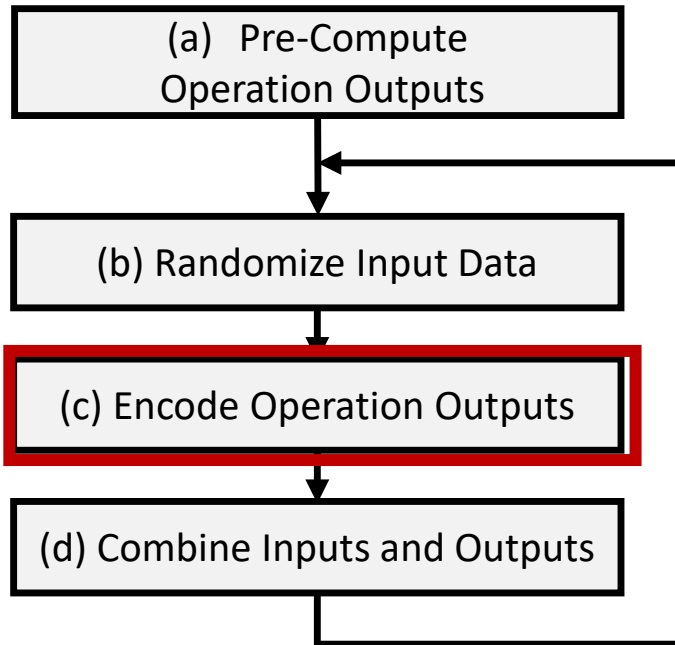
$$\phi \left( f^{-1} \left( \phi_1^{-1}(d_1), \phi_2^{-1}(d_2) \right) \right) = f(d_1, d_2)$$

# RanCompute Input Encoding



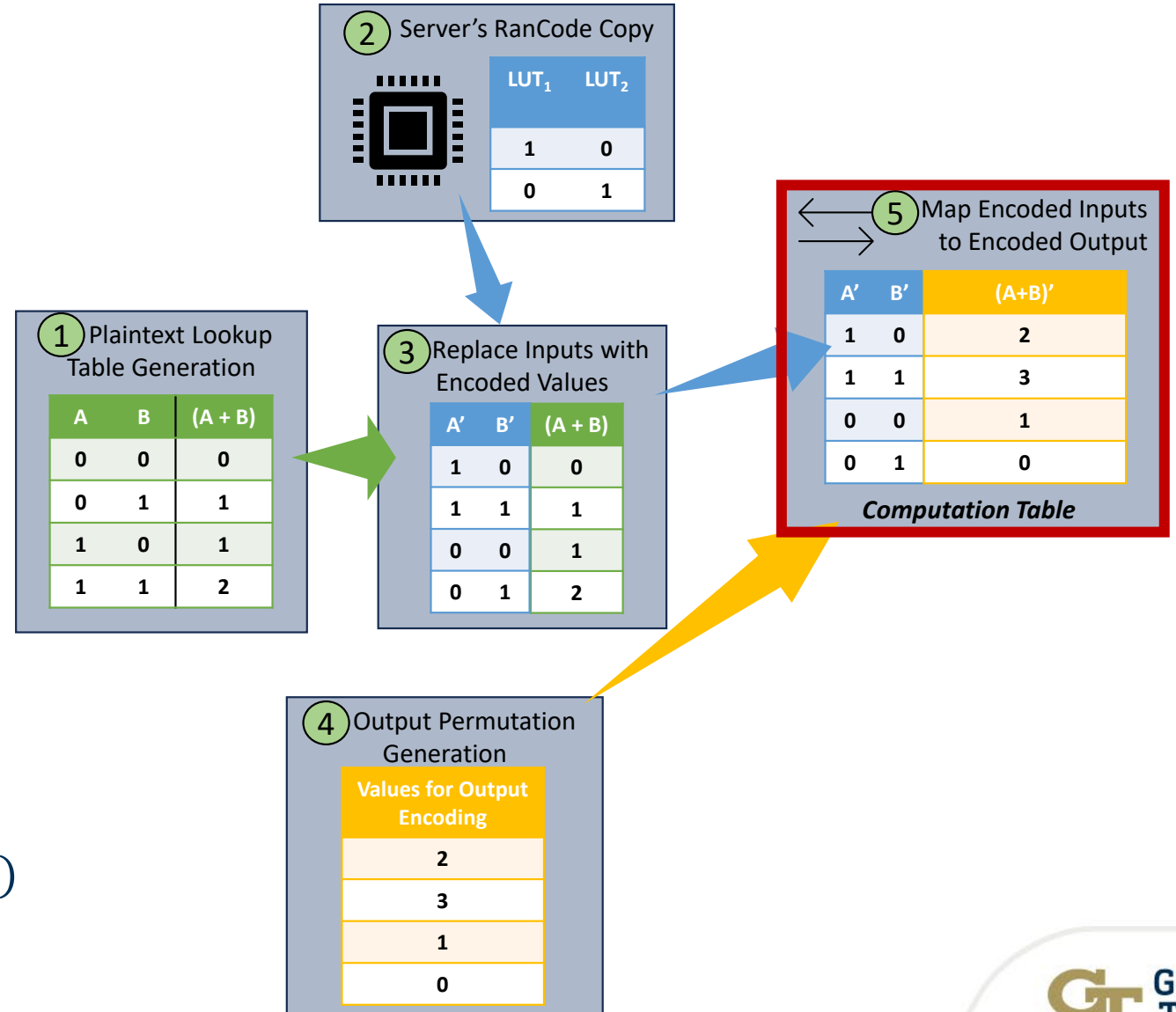
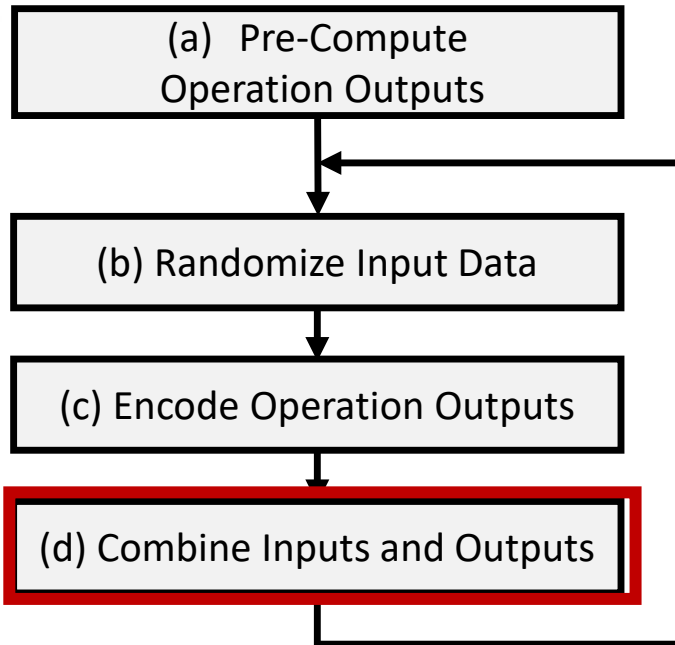
$$\phi \left( f^{-1} \left( \phi_1^{-1}(d_1), \phi_2^{-1}(d_2) \right) \right) = f(d_1, d_2)$$

# RanCompute Output Encoding



$$\phi \left( f^{-1} \left( \phi_1^{-1}(d_1), \phi_2^{-1}(d_2) \right) \right) = f(d_1, d_2)$$

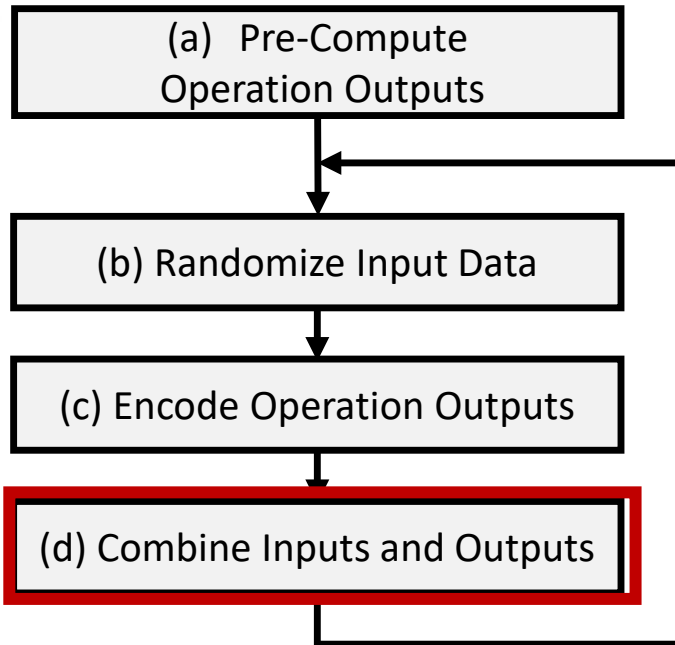
# RanCompute Computation Table



$$\phi \left( f^{-1} \left( \phi_1^{-1}(d_1), \phi_2^{-1}(d_2) \right) \right) = f(d_1, d_2)$$

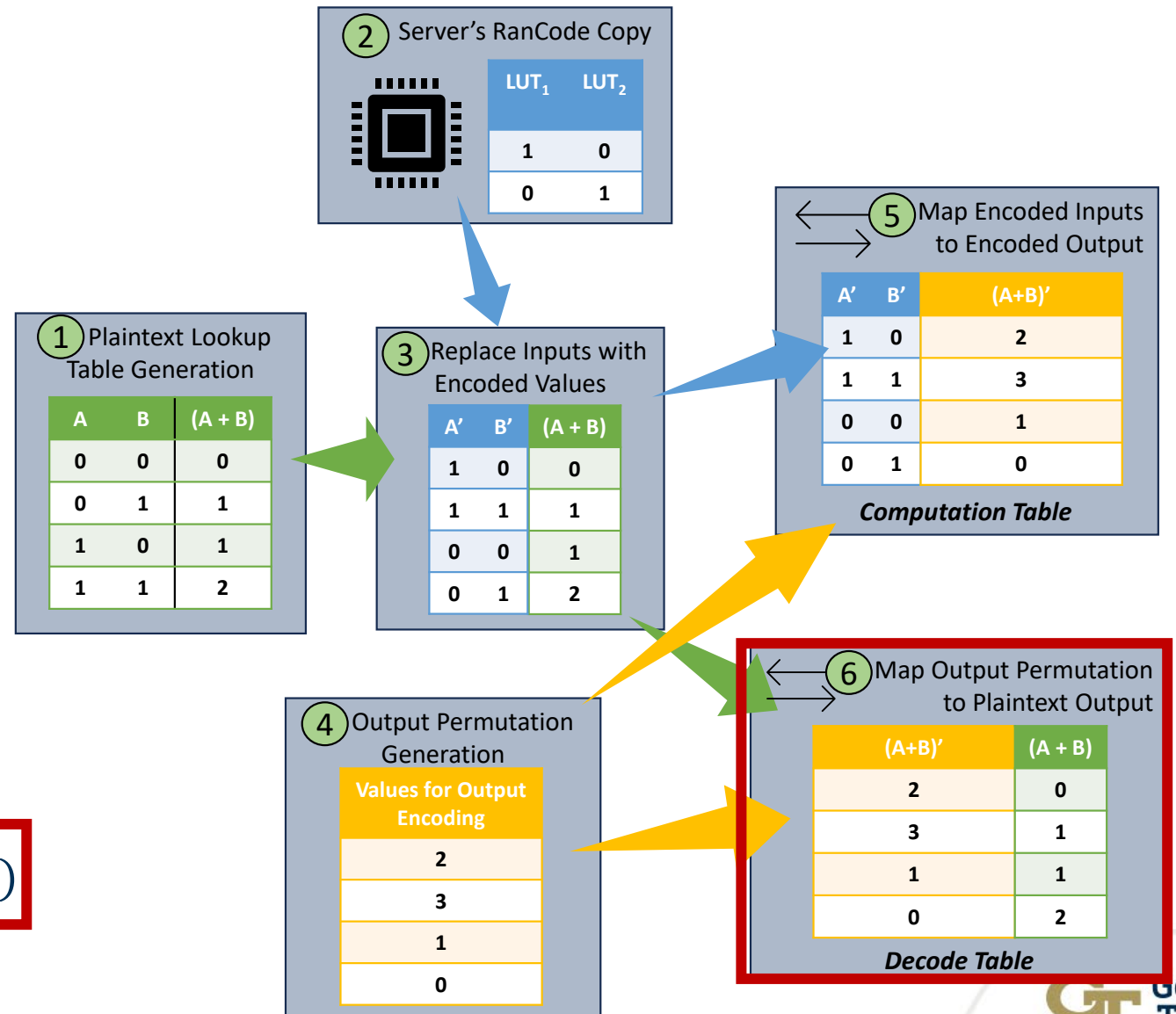
**“Computation Table”**

# RanCompute Decode Table



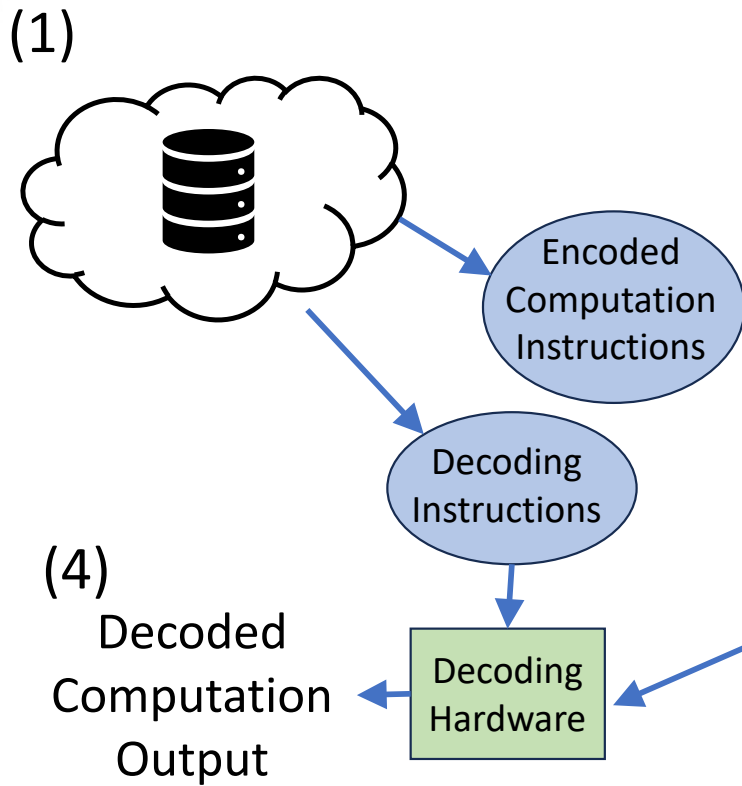
$$\phi \left( f^{-1} \left( \phi_1^{-1}(d_1), \phi_2^{-1}(d_2) \right) \right) = f(d_1, d_2)$$

“Decode Table”

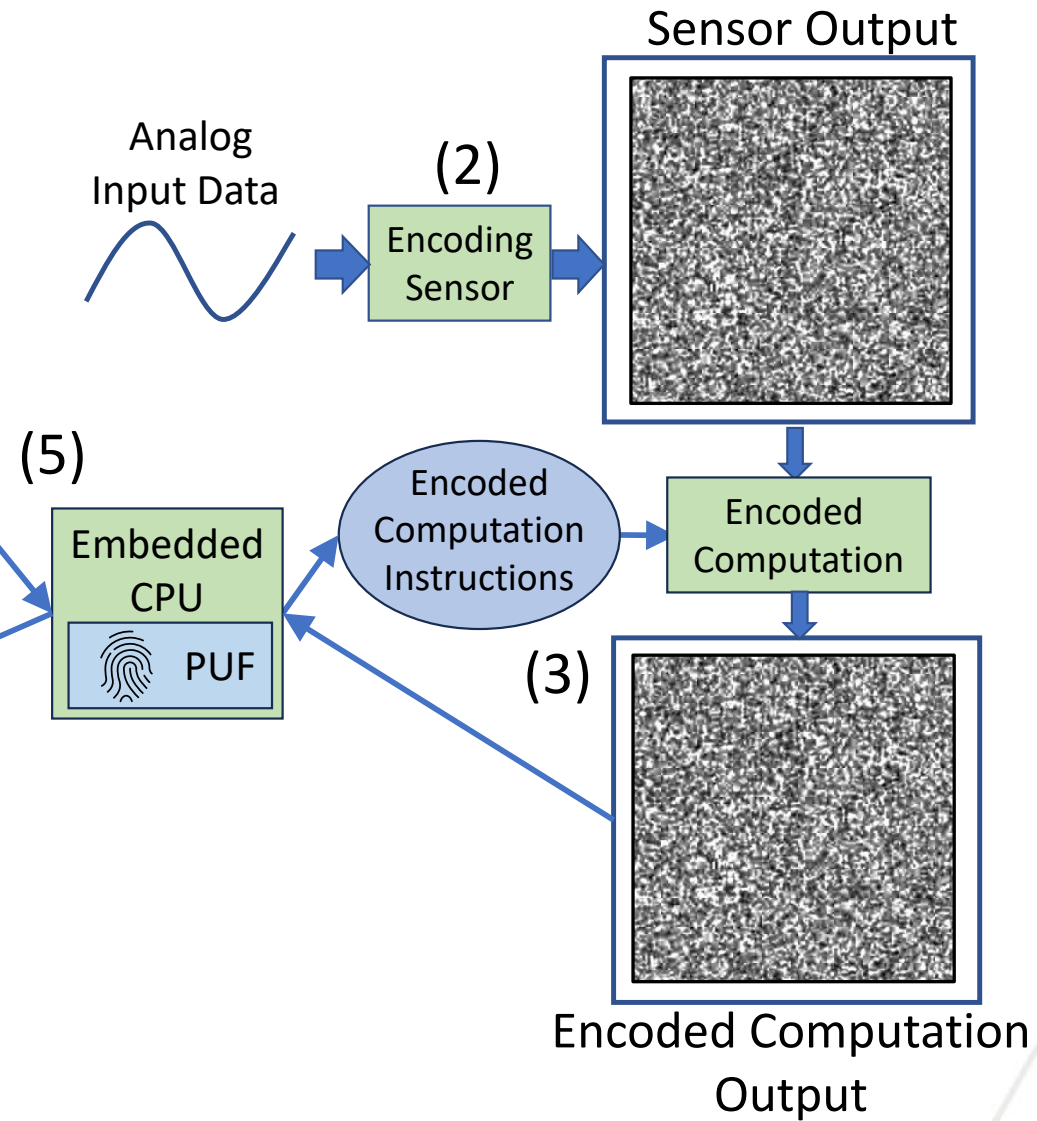


# RanCompute Usage

## Secure Server

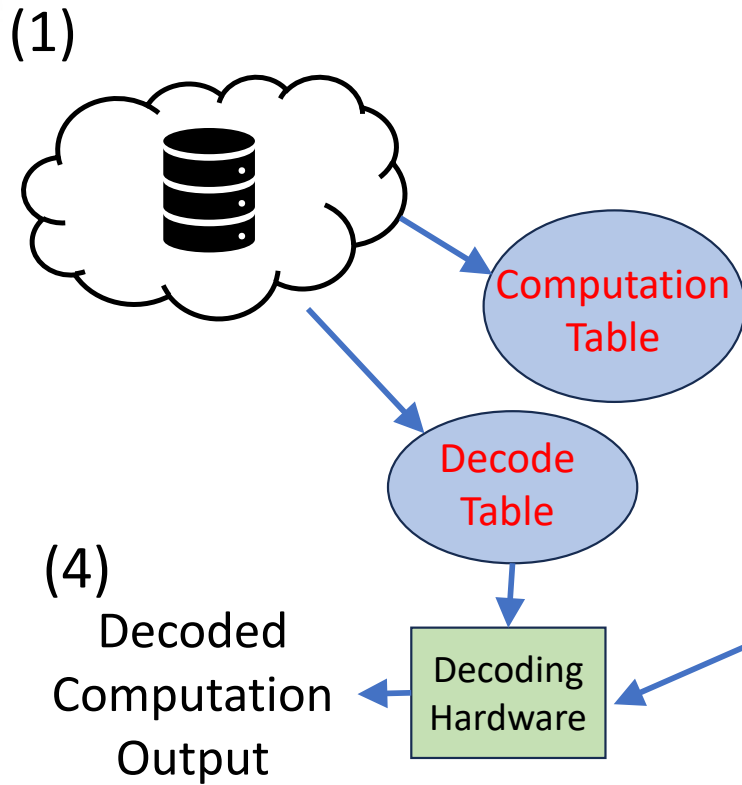


## Deployed Device

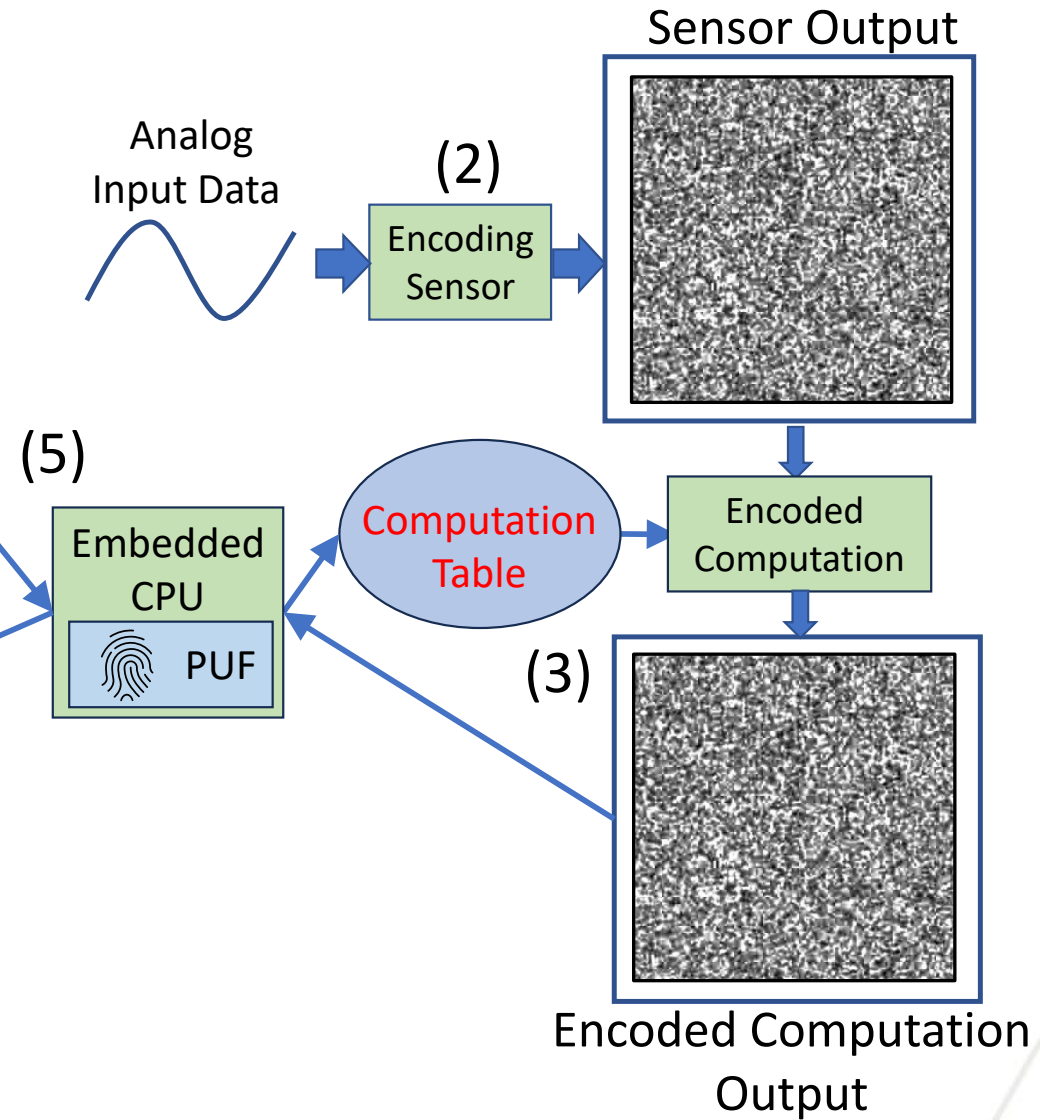


# RanCompute Usage

## Secure Server

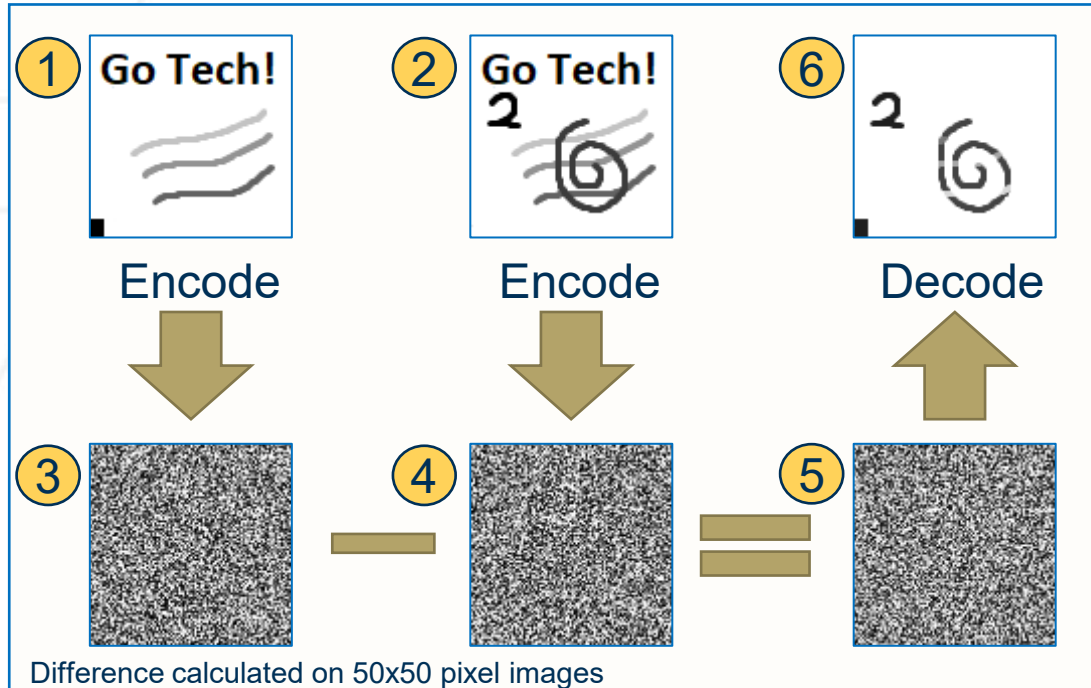


## Deployed Device

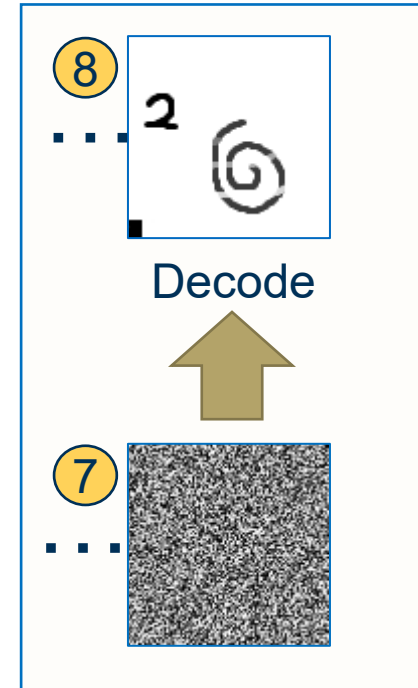


# RanCompute – Difference Calculation

## Lossy Precision



## Full Precision



Computation Table Data Usage

	Per Pixel		Per 720p Image	
	Lossy	Full	Lossy	Full
Comp Tables	1	1	921,600	921,600
Total Comp Table Rows	512	65,536	4.7e8	6.0e10
Total Comp Table Bytes	720,896	131,072	471 MB	120 GB

- With a high-performance SHA-3 core and proven dynamic FPGA technology, image difference can be calculated for 720p (720 pixels by 1280 pixels) images at 20fps

Image Difference: measure of the difference of pixel values between two images:

$$\text{Full Precision: } \forall i \rightarrow \text{Diff}_i(t, T) = |I_{t-T}(i) - I_t(i)|$$

$$\text{Lossy Precision: } \forall i \rightarrow \text{Diff}_i(t, T) = |I_{t-T}(i)[7:4] - I_t(i)[7:4]| \& |I_{t-T}(i)[3:0] - I_t(i)[3:0]|$$

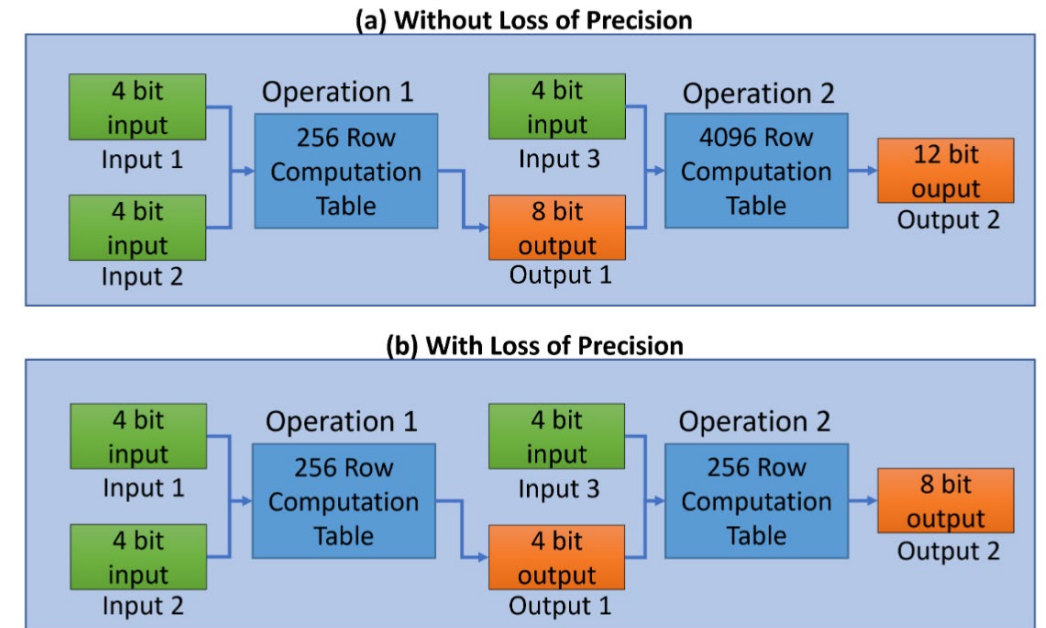


# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- **Implementing a Privacy Homomorphism With a Security-Enhanced ADC**
  - Privacy Homomorphism Scheme
  - **Scheme Extensions**
  - Architecture for Edge Detection
  - Security
  - Comparison to Fully Homomorphic Encryption
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# Extending Depth of Computations

- To accommodate many inputs, a single Computation Table will become very large as the number of inputs increases
- To perform an increased depth of number of sub-computations to compose a larger computation, we limit the number of possible output values (**output binning**) so that we can limit the size of our look up tables



A	B	Unencoded Output Values (Addition)	Encoded Table Values
0	0	0	10
0	1	1	11
1	0	1	01
1	1	2	00

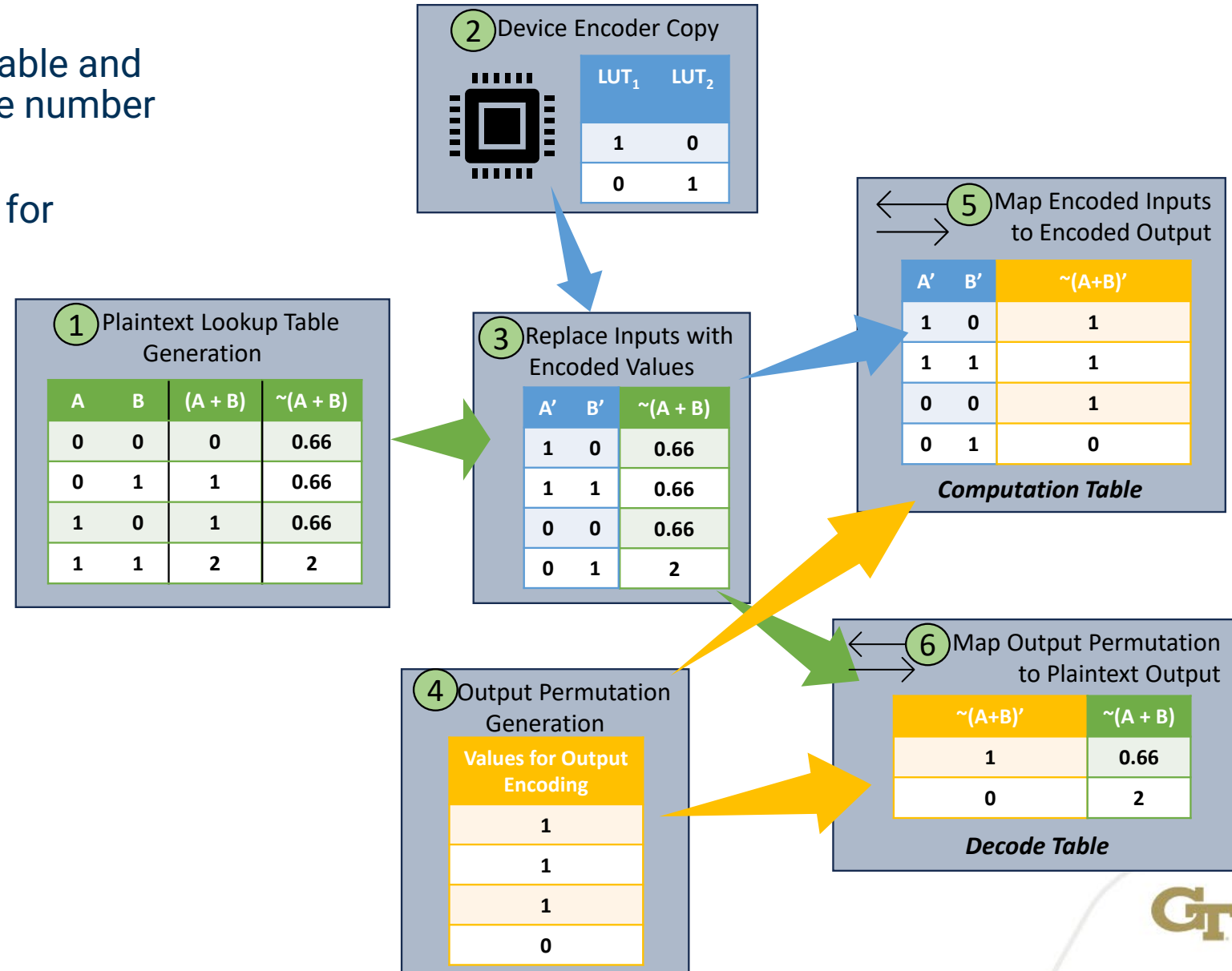
(a) Full Precision Computation Table

A	B	Unencoded Output Values (Addition)	Encoded Table Values
0	0	0.66	1
0	1	0.66	1
1	0	0.66	1
1	1	2	0

(b) Reduced Precision Computation Table

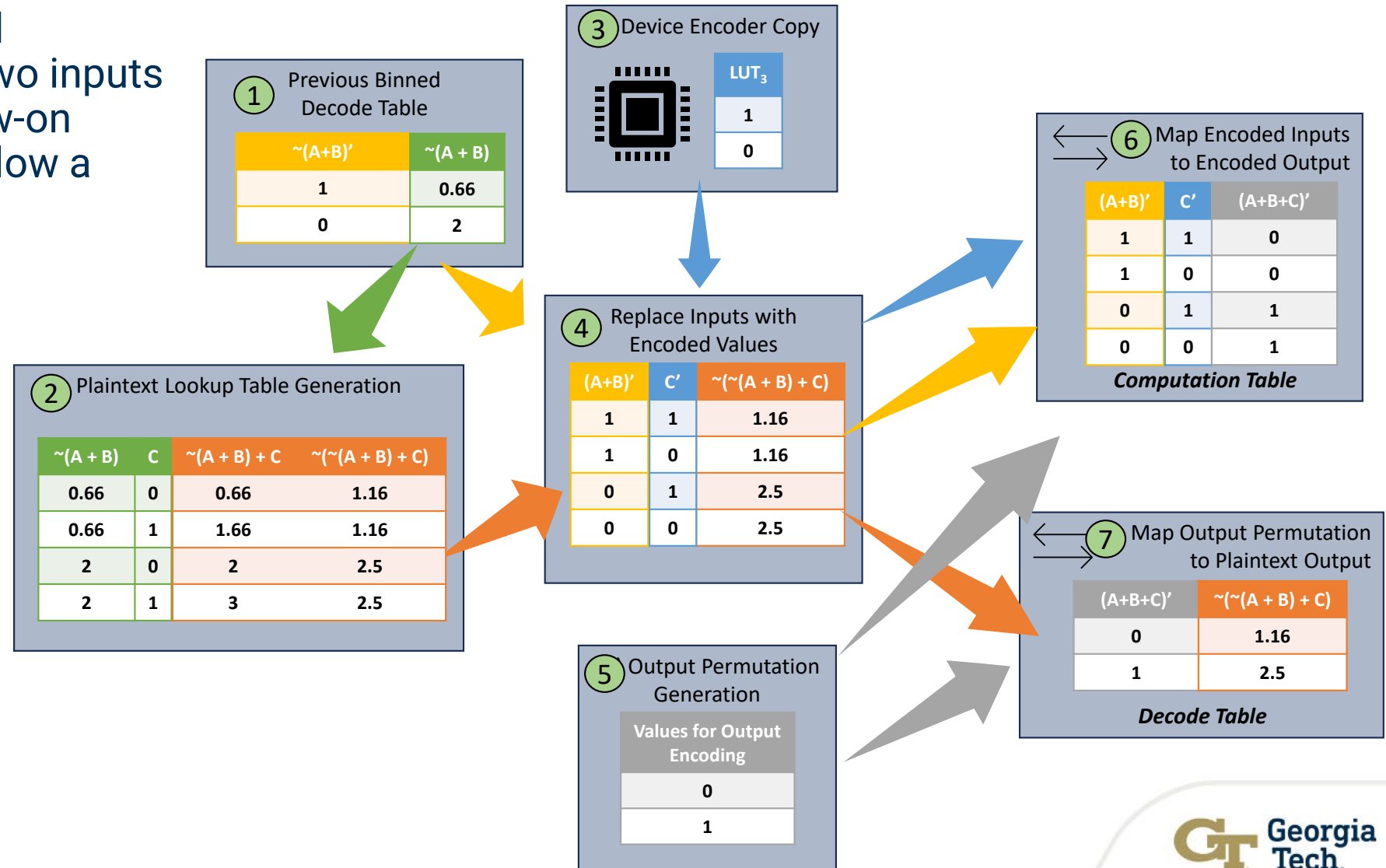
# Limiting Computation Output Possibilities

- Creation of the Computation Table and Decode Table when limiting the number of output possibilities
- Calculation precision is traded for reduced table sizes



# Chaining Computation Tables

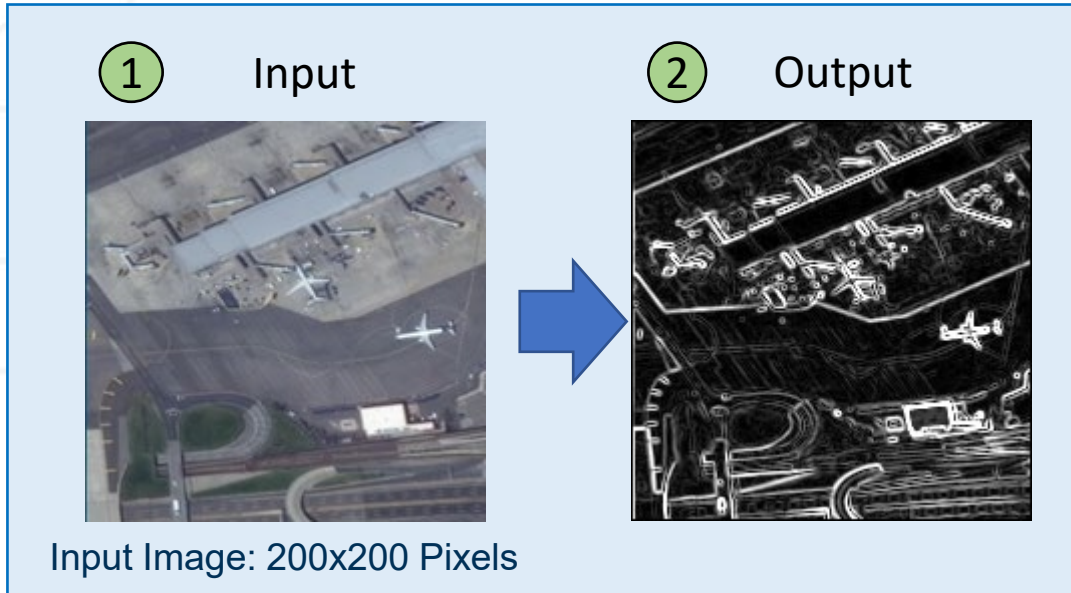
- The previously produced Computation Table on two inputs is used to create a follow-on Computation Table to allow a third input



# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- **Implementing a Privacy Homomorphism With a Security-Enhanced ADC**
  - Privacy Homomorphism Scheme
  - Scheme Extensions
  - **Architecture for Edge Detection**
  - Security
  - Comparison to Fully Homomorphic Encryption
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

# RanCompute – Edge Detection



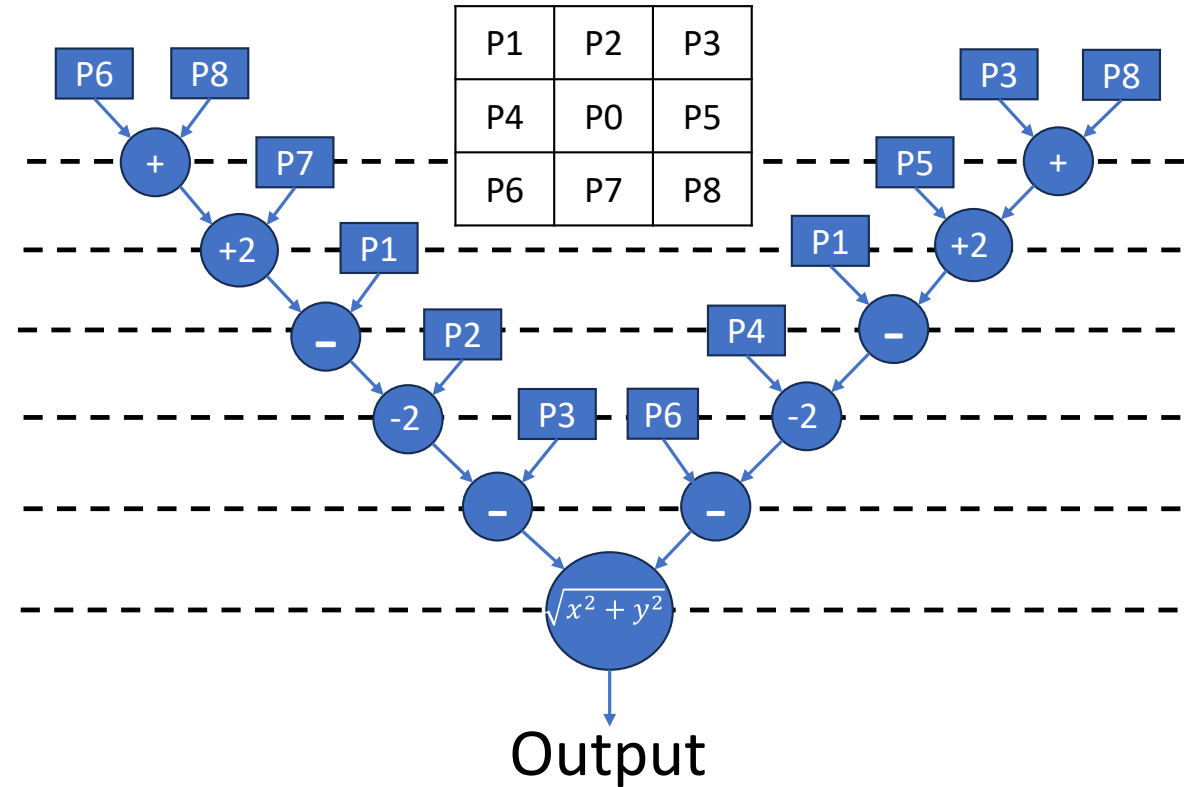
Edge Detection: image processing technique to extract object boundaries from an image

## Canny Edge Detection [21]

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * I$$

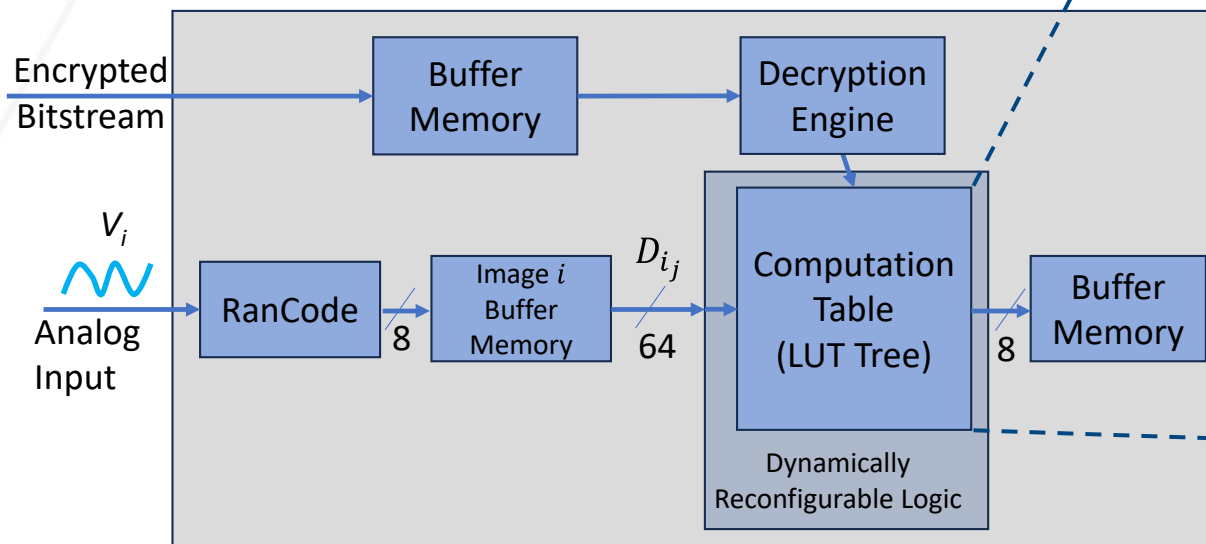
$$G = \sqrt{G_x^2 + G_y^2}$$

Directed Acyclic Graph for Each Pixel Computation

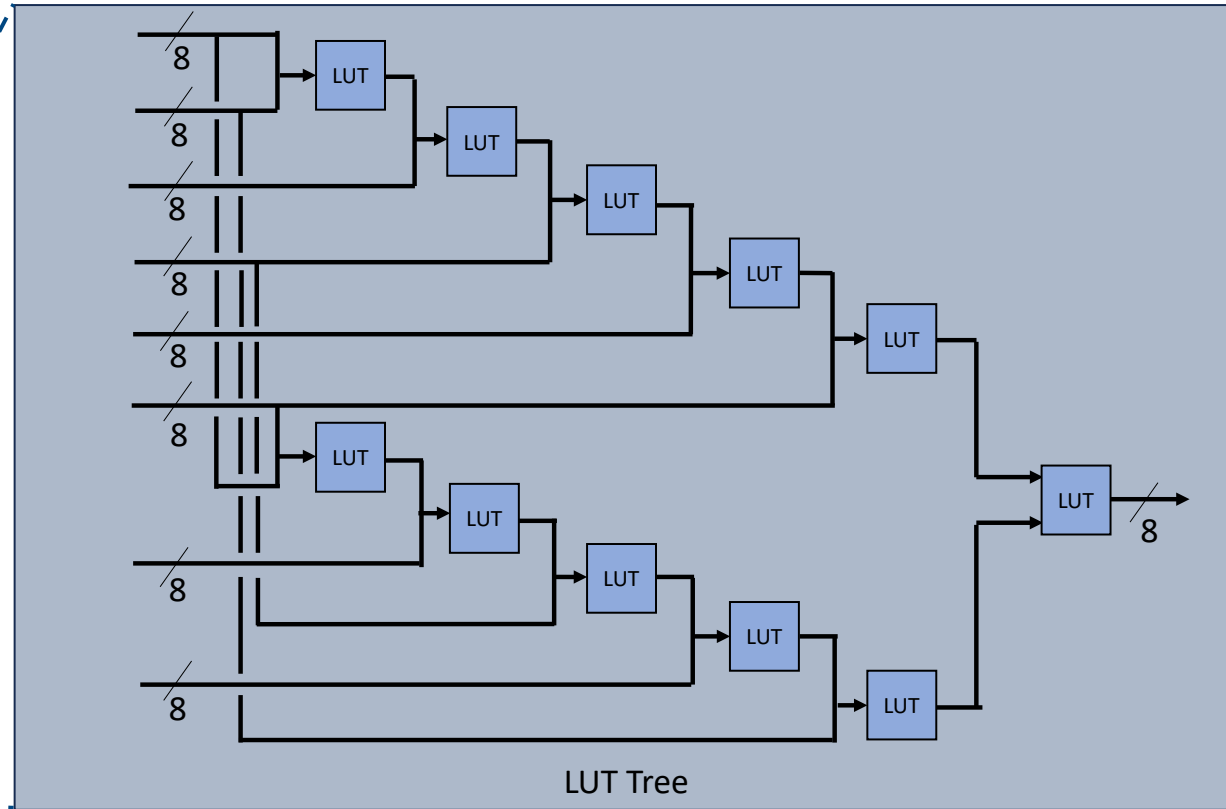


# Device Architecture – Edge Detection

- Architecture receives encrypted bitstream which is loaded onto reconfigurable logic
- Inputs are provided by RanCode through a buffer
- Each LUT tree performs one pixel computation



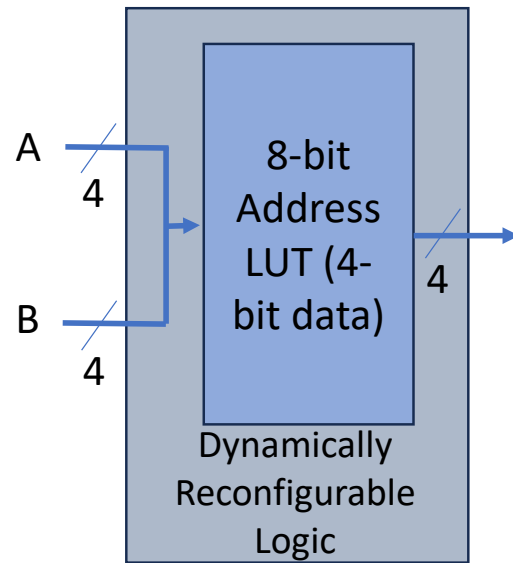
Deployed Device



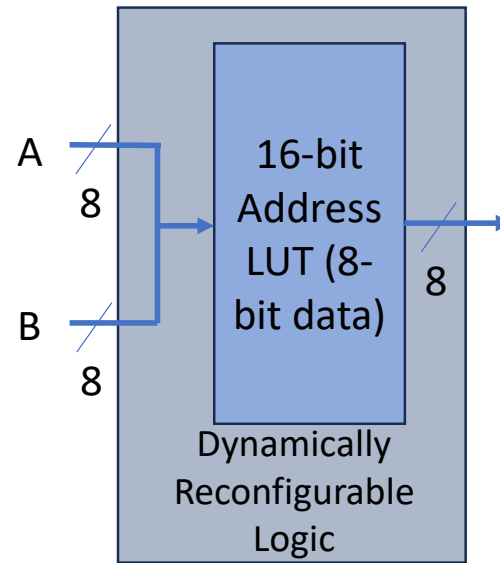
# Device Architecture – Edge Detection

Three architectures implemented:

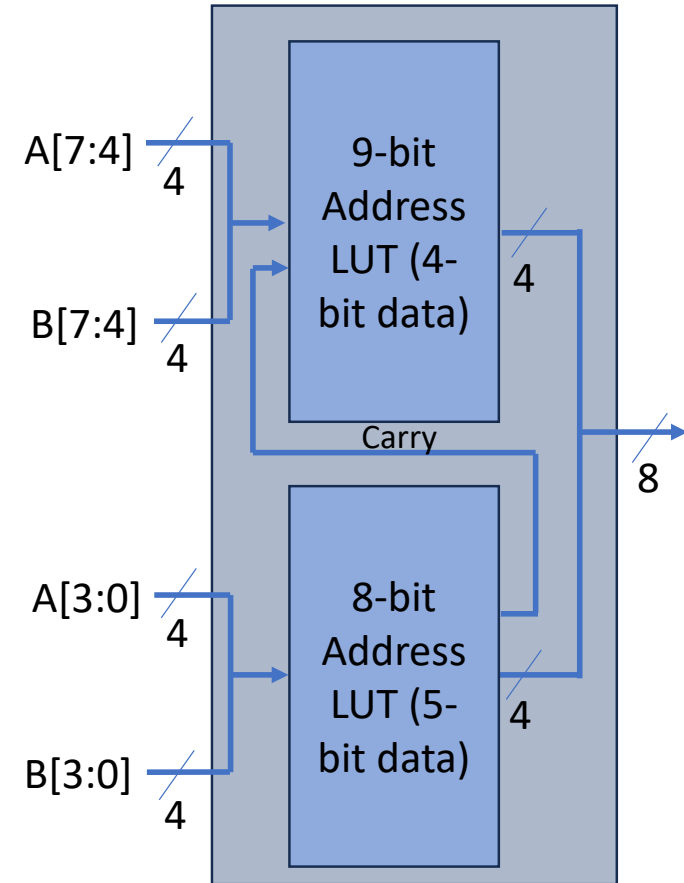
- 4-bit image with binned outputs
- 8-bit image with binned outputs
- 8-bit image with binned outputs and a carry



(a) Binned, 4-bit Image



(b) Binned LUT



(c) Binned with Carry



# Results

- All three encoded computations retain enough resolution to clearly distinguish the features of the original image
- 8-bit binned too large to fit on our FPGA

## Synthesis Results

	LEs	Registers	MHz
<b>4-bit binned</b>	3,753	2,574	380
<b>8-bit binned with carry</b>	46,897	2,624	380

## Computation Table Data Usage

	Per Pixel			Per 720p Image		
	Binned	Binned with Carry	4-bit Binned	Binned	Binned with Carry	4-bit Binned
<b>Comp Tables</b>	11	21	11	10,137,600	19,353,600	10,137,600
<b>Total Comp Table Rows</b>	720,896	80,896	2,816	6.6e11	7.5e10	2.6e9
<b>Total Comp Table Bytes</b>	720,896	73,856	1,408	664 GB	68.1 GB	1.3 GB



① Plaintext



② Binned Outputs



③ Binned with Carry



④ 4-bit Binned

# Results

- Data requirements for the Computation Tables for binned, 4-bit image computation are within the bandwidth rates of 5G transmissions for **one frame of edge detection every second**

Computation Table Data Usage

	Per Pixel			Per 720p Image		
	Binned	Binned with Carry	4-bit Binned	Binned	Binned with Carry	4-bit Binned
Comp Tables	11	21	11	10,137,600	19,353,600	10,137,600
Total Comp Table Rows	720,896	80,896	2,816	6.6e11	7.5e10	2.6e9
Total Comp Table Bytes	720,896	73,856	1,408	664 GB	68.1 GB	1.3 GB



① Plaintext



② Binned Outputs



③ Binned with Carry



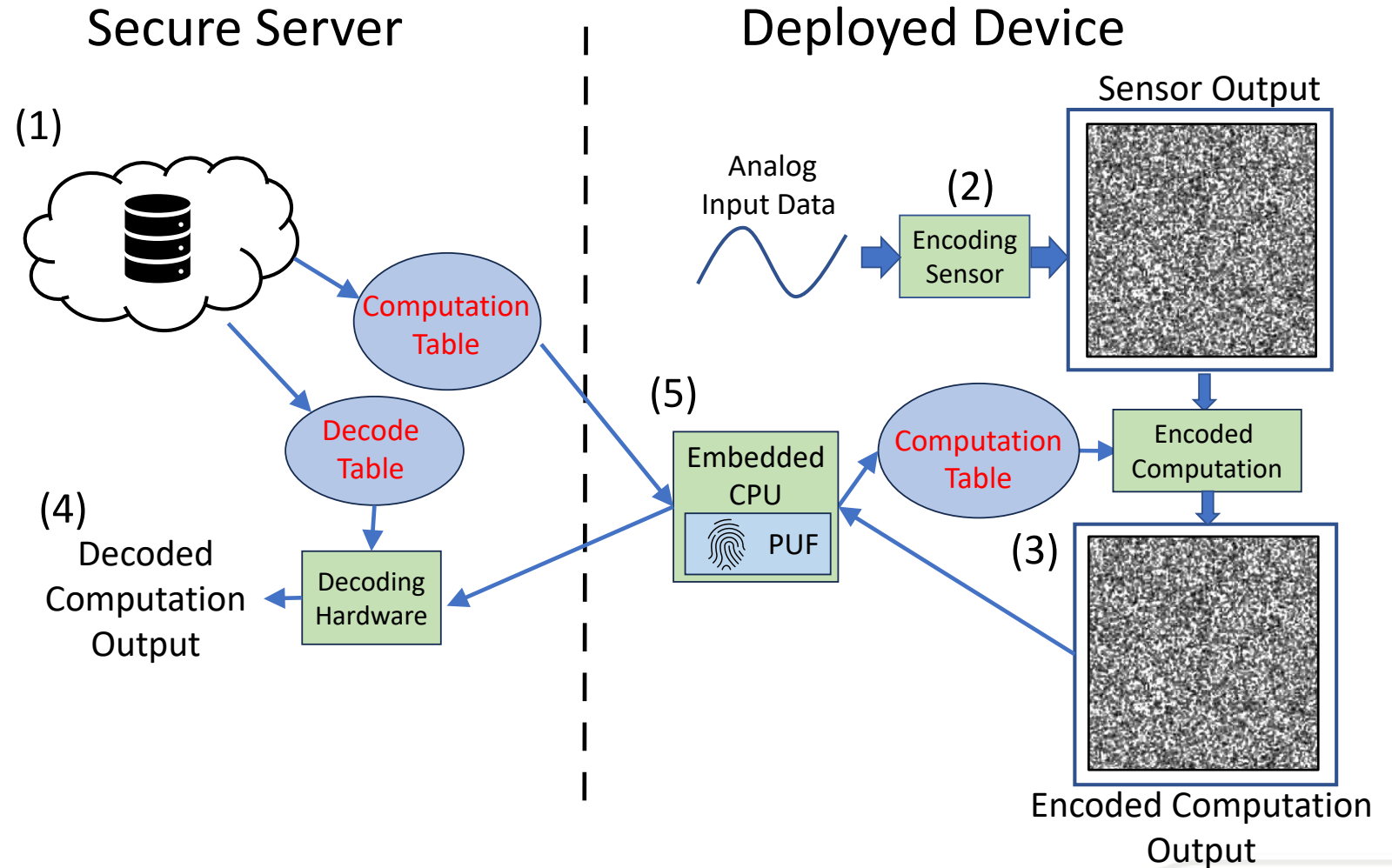
④ 4-bit Binned

# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- **Implementing a Privacy Homomorphism With a Security-Enhanced ADC**
  - Privacy Homomorphism Scheme
  - Scheme Extensions
  - Architecture for Edge Detection
  - **Security**
  - Comparison to Fully Homomorphic Encryption
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

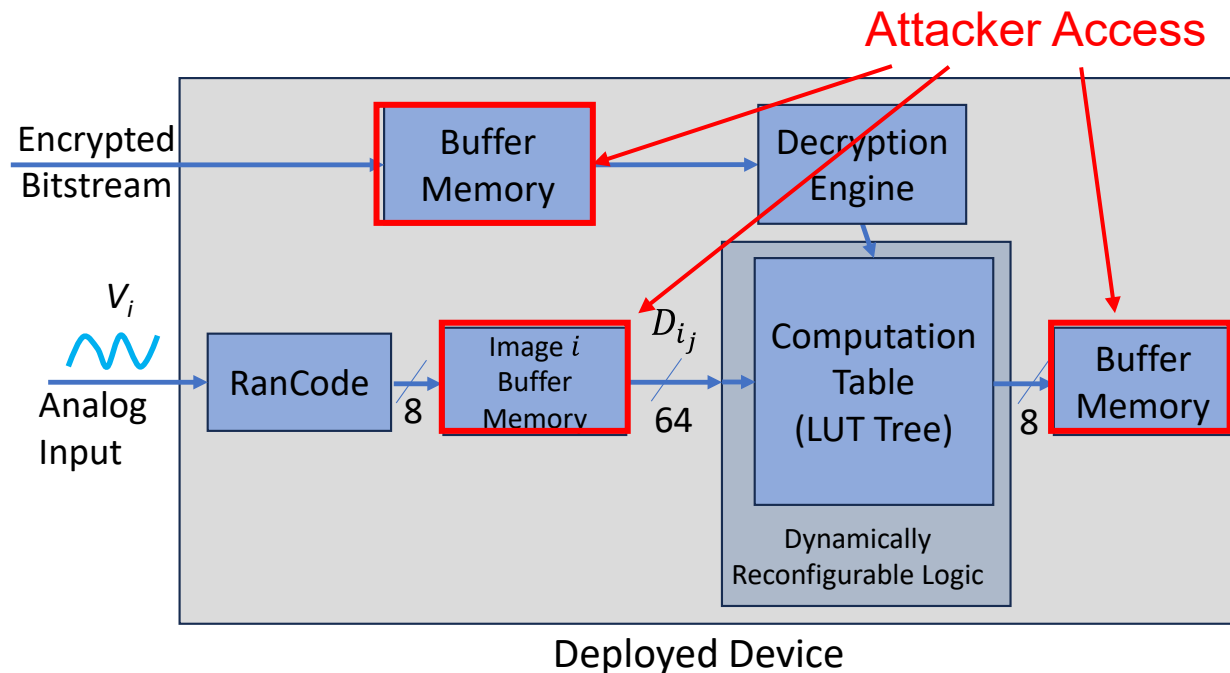
# Security Analysis

- RanCompute has three main components:
  - RanCode
  - Computation Table(s)
  - Decode Table
- RanCode was discussed previously, so we will now discuss the security of the Computation Table(s) and the Decode Table in a RanCompute application



# Security Analysis – Attack Model

- An actor can read run-time data stored in general purpose memory
  - Cannot read microarchitectural units such as registers holding partial results of operations
- The attacker cannot precisely replicate the raw data values (analog or digital) sensed in the remote environment
- The attacker can take the device offline at some point in time to attempt reverse engineering of the device through destructive methods
- The attacker's goal is to eavesdrop the operation of the device and the executed computation such that the attacker **gains knowledge of either the unencoded sensor input data, an unencoded version of the computation output, or both over a reasonably long period of time (including the past)**



# Security of the Computation Table

- Each Computation Table output is obtained from a **unique random permutation** generated by the secure server
- Each new output seen by the adversary will be **unrelated to the previous output** even if both outputs were obtained from equivalent input values
- The RanCode input encoding permutations are completely distinct from the permutations used for the Computation Table output encodings

← → Map Encoded Inputs to Encoded Output

A'	B'	(A+B)'
1	0	2
1	1	3
0	0	1
0	1	0

*Computation Table*

# Security Analysis – Output Binning

- Output binning affects the distribution and frequency of encoded symbols in a Computation Table output column
  - In full precision table, each element in the output column appears once
  - In the binned table, '1' appears 3 times while '0' appears 1 times
- Frequency analysis may lead to the ability to decode Computation Table outputs if the entire Computation Table is obtained
- To ensure protection, the Computation Tables are sent in an **encrypted format** (i.e., an encrypted FPGA bitstream) and then decrypted when being loaded

A'	B'	(A+B)'
1	0	2
1	1	3
0	0	1
0	1	0

Full Precision Computation Table

Each encoded output appears with **equal frequency**

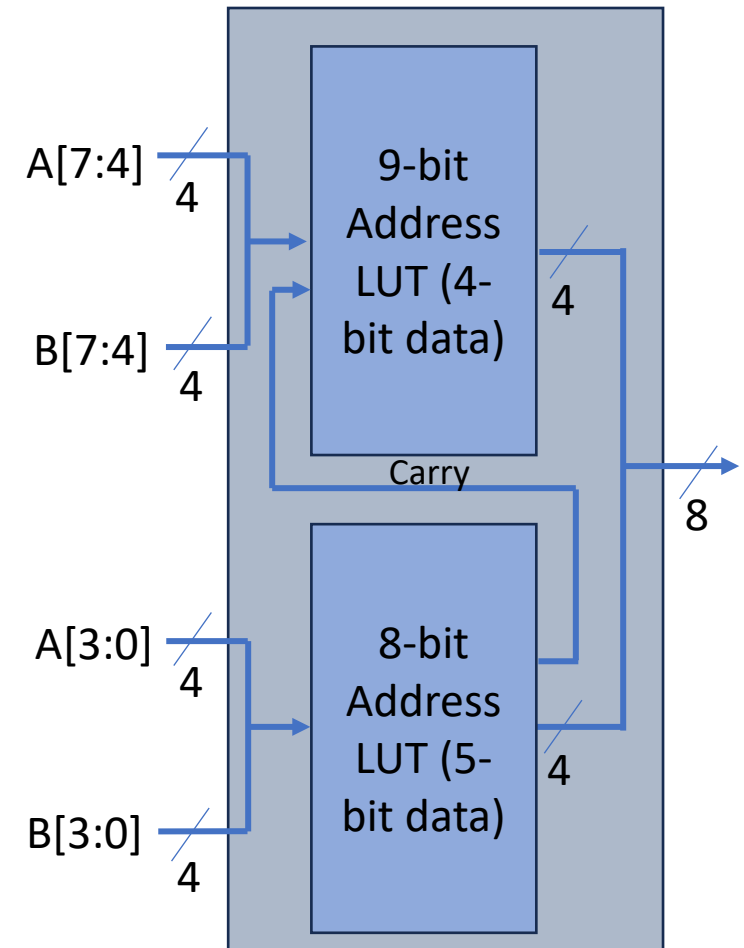
A'	B'	$\sim(A+B)'$
1	0	1
1	1	1
0	0	1
0	1	0

Output Binned Computation Table

Each encoded output appears a **unique number of times**

# Security Analysis – Carry Bit

- The carry bit in an addition or subtraction operation may reveal some information about the magnitudes of the two inputs if observed in plaintext
- The carry bit is encoded randomly as a '0' or '1' for each table
  - Observation of the carry bit (which we assume adversary cannot do) by an adversary would not reveal anything as each Computation Table is only used once
- The distribution of the frequency of appearance of the carry bits in the Computation Tables are inaccessible to the adversary



(c) Binned with Carry



# Security of a Many-input Computation

- Multiple iterations of a run of Computation Tables utilize overlaps of the pixels previously used
  - One encoded edge output relies on some overlap of encoded pixels from another encoded edge output)
- However, each output is encoded via a new random permutation
- To empirically check if an overlap of encoded inputs causes correlated outputs in our scheme, we performed NIST randomness testing on the encoded computed output images

NIST Randomness Testing Results

	Binned	Binned with Carry	4-bit Binned
Monobit Frequency	100	99	98
Block Frequency	100	100	99
Cumulative Sums	100	99	97
Runs	97	99	100
Longest Run	99	99	98

Results for randomness tests from the NIST Statistical Test Suite [22]. Results are individual passes out of 100 runs, with 96 considered as passing the test for randomness overall.

	Full Precision			Lossy Computation		
	Enc Image 1	Enc Image 2	Enc Diff	Enc Image 1	Enc Image 2	Enc Diff
	<b>Different Images</b>					
Avg. Entropy	7.901	7.904	7.922	7.901	7.877	7.892
Std. Dev	0.010	0.007	0.007	0.006	0.005	0.005
Variance	5.344e-5	9.453e-5	5.368e-5	3.966e-5	2.786e-5	2.878e-5
	<b>No Difference</b>					
Avg. Entropy	7.900	7.904	7.925	7.901	7.878	7.878
Std. Dev	0.004	0.009	0.007	0.006	0.006	0.006
Variance	1.948e-5	8.329e-5	5.2889e-5	3.953e-5	3.914e-5	3.509e-5
	<b>Max Difference</b>					
Avg. Entropy	7.769	7.869	7.924	7.837	7.901	7.928
Std. Dev	0.007	0.007	0.008	0.007	0.006	0.004
Variance	2.763e-5	5.468e-5	6.759e-5	4.601e-5	3.324e-5	1.337e-5

Shannon Entropy of the encoded images and computation results for 100 runs of three image pairings.

Entropy Results for Edge Detection Calculation

	Binned		Binned with Carry		4-bit Binned	
	Encoded Input	Encoded Output	Encoded Input	Encoded Output	Encoded Input	Encoded Output
Avg. Entropy	7.99	7.99	7.99	7.99	7.99	3.99
Std. Dev	0.05E-2	0.05E-2	0.06E-2	0.04E-2	0.05E-2	0.04E-2
Variance	3.4E-7	1.84E-7	4.76E-7	1.92E-7	3.41E-7	1.75E-7

# Security of the Decoding Table

- The Decoding Table resides only on the secure server
  - Can exclude direct adversarial access as a possible attack vector
- The Decoding Table input is derived from a pseudorandom permutation produced independently from the RanCode mappings
- The adversary can view a single output value from the Computation Table, corresponding to a single Decode Table input
- No information about the plaintext output value obtained from the Decode Table appears to be available only by providing one row of the Computation Table

← → Map Encoded Inputs to Encoded Output

A'	B'	(A+B)'
1	0	2
1	1	3
0	0	1
0	1	0

*Computation Table*

For each computation, adversary only sees a single row

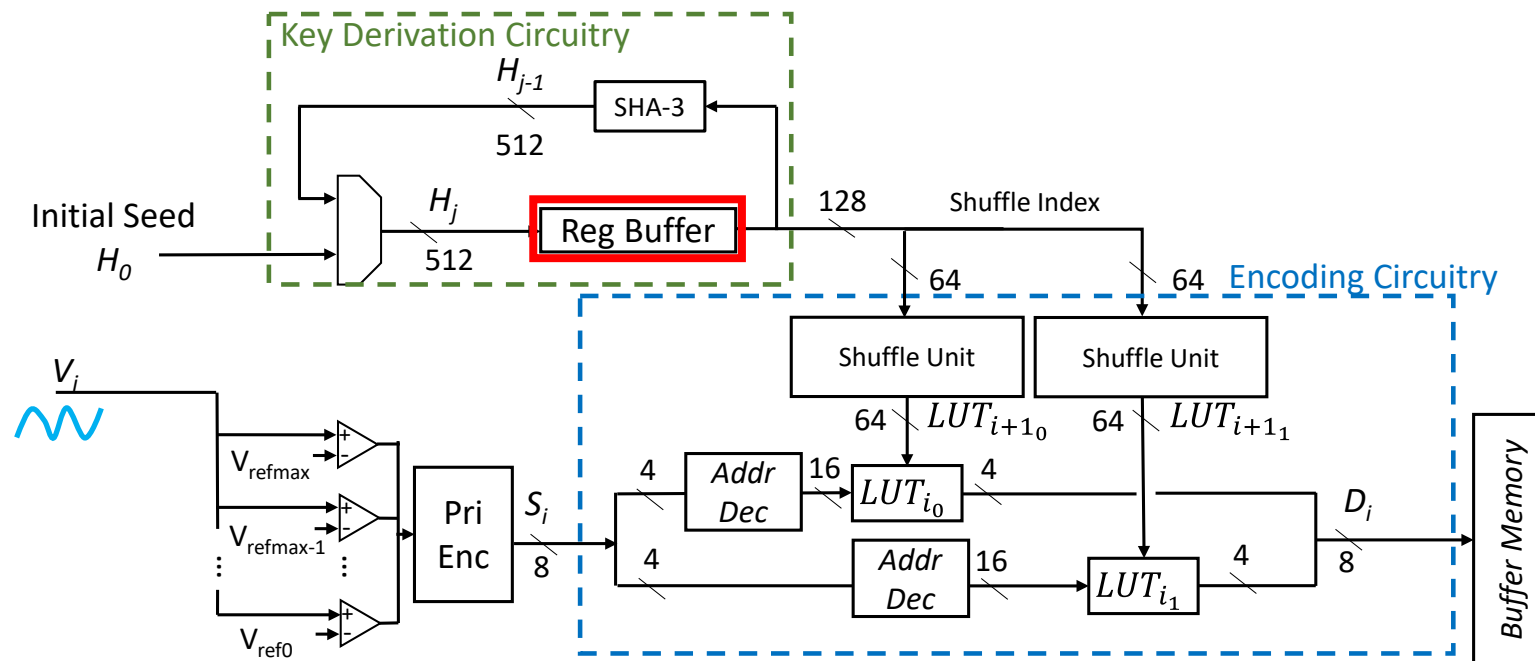
← → Map Output Permutation to Plaintext Output

(A+B)'	(A + B)
2	0
3	1
1	1
0	2

*Decode Table*

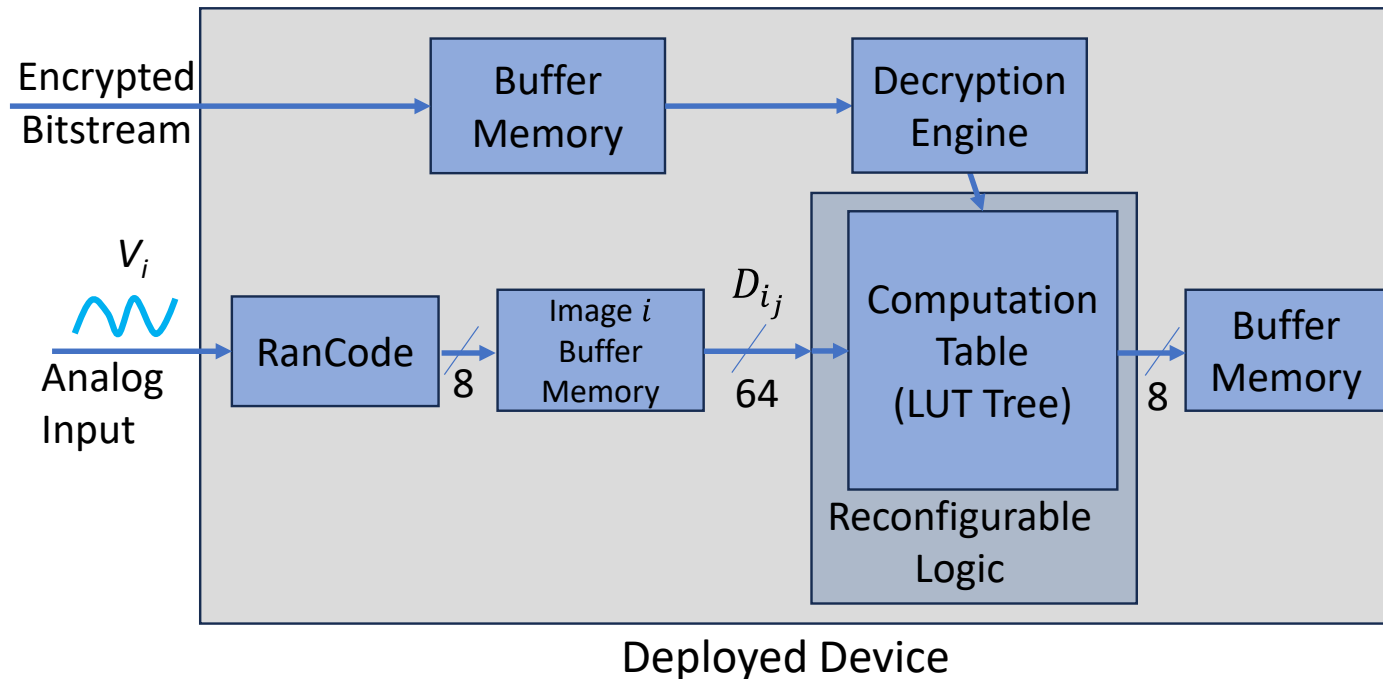
# Security after Reverse Engineering

- The encoding scheme is reliant on a pseudorandom permutation derived each iteration from a 512-bit vector  $H_j$
- If the circuit is reverse engineered to obtain the most recent  $H_j$ , the adversary can determine the encoding used for the last input encoded
  - This is only enough information to determine one unencoded value for the difference computation which requires two input values



# Vulnerabilities

- We have attempted to red team exhaustively and note that we do not protect against
  - An adversary who can determine  $H_j$  via a mechanism such as a Hardware Trojan
  - Determination of the Computation Tables when output binning is utilized

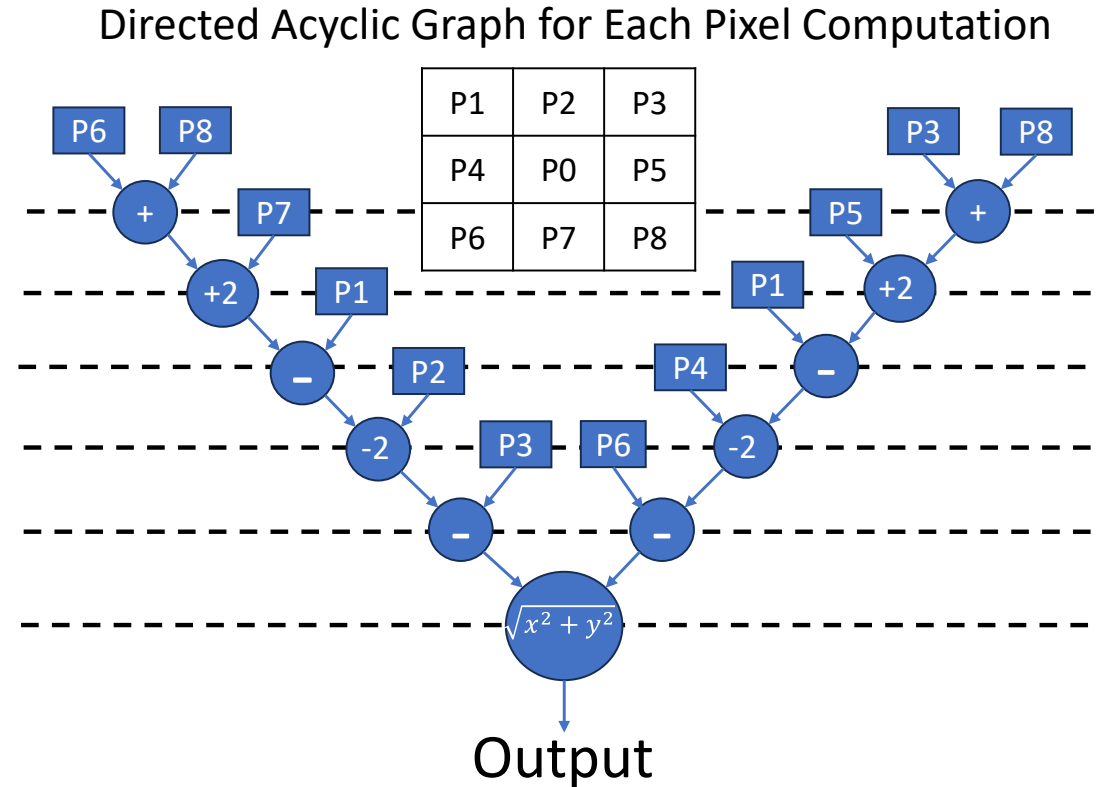


# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- **Implementing a Privacy Homomorphism With a Security-Enhanced ADC**
  - Privacy Homomorphism Scheme
  - Scheme Extensions
  - Architecture for Edge Detection
  - Security
  - **Comparison to Fully Homomorphic Encryption**
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- References

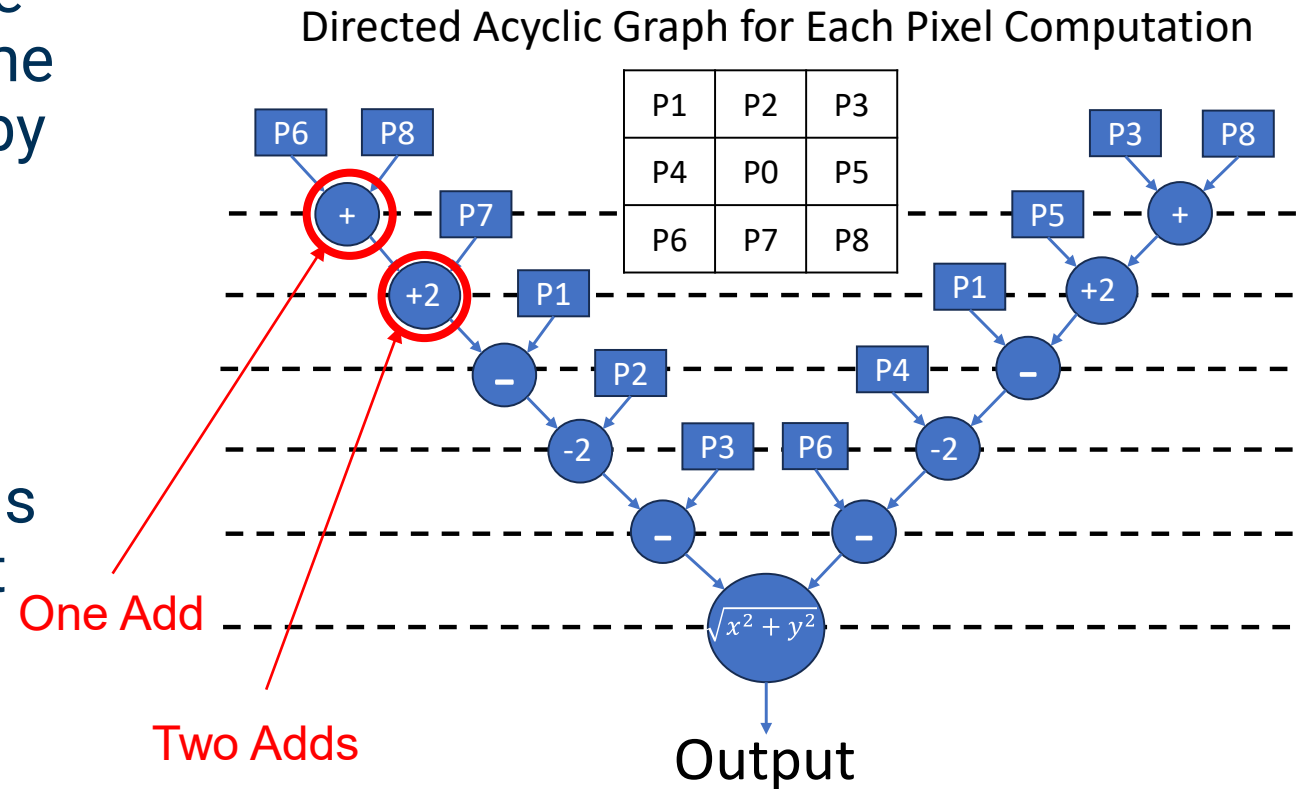
# Comparison to FHE

- For comparison, we map Canny Edge detection to the BGV FHE [11] scheme by utilizing the operations provided by HElib [23] and the timing realized by the hardware BGV implementation BASALISC [12]
- Complete computation per pixel consists of 15 additions/subtractions and two multiplies and a square root operation



# Comparison to FHE

- For comparison, we map Canny Edge detection to the BGV FHE [11] scheme by utilizing the operations provided by HElib [23] and the timing realized by the hardware BGV implementation BASALISC [12]
- Complete computation per pixel consists of 15 additions/subtractions and two multiplies and a square root operation
  - When, for example,  $X + 2Y$  is performed as  $X + Y + Y$  instead of  $X + 2 * Y$



# Comparison to FHE

- Square root can be implemented with algorithms such as Newton-Raphson [24]
  - Multiple iterations required for accuracy
  - Decent accuracy generally requires  $>4$  iterations
  - Each iteration requires a multiply
- Utilizing BGV with HElib shows a need to bootstrap 8-bit encrypted values after 3 multiplies
- Conservatively assume square root requires 3 multiplies, 3 add/sub and 1 bootstrapping



# Comparison to FHE

- Square root can be implemented with algorithms such as Newton-Raphson [24]
  - Multiple iterations required for accuracy
  - Decent accuracy generally requires >4 iterations
  - Each iteration requires a multiply
- Utilizing BGV with HElib shows a need to bootstrap 8-bit encrypted values after 3 multiplies
- Conservatively assume square root requires 3 multiplies, 3 add/sub and 1 bootstrapping

## BASALISC Operation times

Operation	Time
Add/Sub	8 $\mu$ s
Mult	20 $\mu$ s
Bootstrapping	40 ms

- 18 add/subs and 5 multiplies = 244  $\mu$ s
  - With bootstrapping, 40244  $\mu$ s per pixel
- Total computation time for 720p image = **~10 hours**
- Our implementation (achieving one 720p image per second) shows **~37,000X speed up**

# Comparison to FHE

- Square root can be implemented with algorithms such as Newton-Raphson [24]
  - Multiple iterations required for accuracy
  - Decent accuracy generally requires >4 iterations
  - Each iteration requires a multiply
- Utilizing BGV with HElib shows a need to bootstrap 8-bit encrypted values after 3 multiplies
- Conservatively assume square root requires 3 multiplies, 3 add/sub and 1 bootstrapping

## BASALISC Operation times

Operation	Time
Add/Sub	8 $\mu$ s
Mult	20 $\mu$ s
Bootstrapping	40 ms

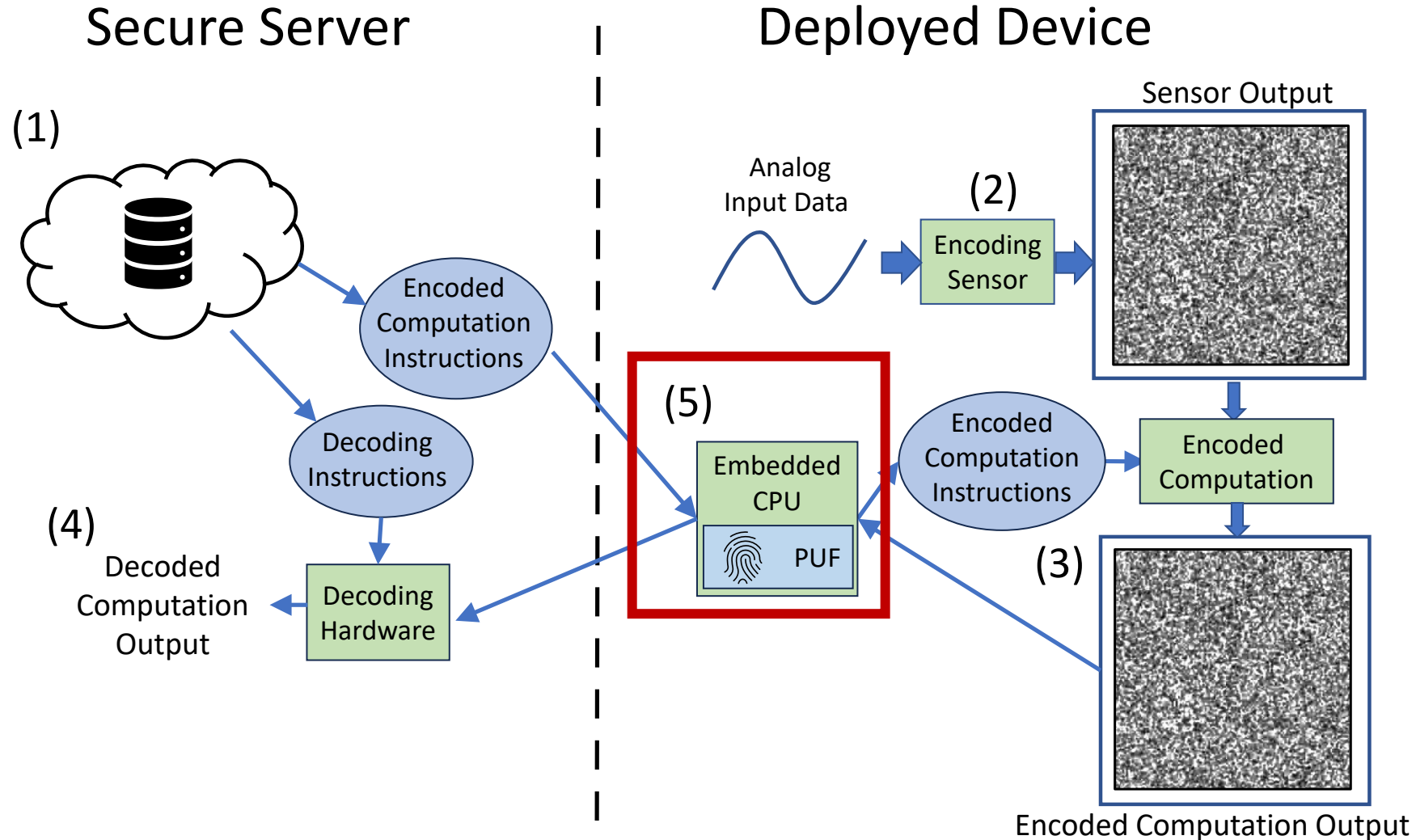
- 18 add/subs and 5 multiplies = 244  $\mu$ s
  - With bootstrapping, 40244  $\mu$ s per pixel
- Total computation time for 720p image = **~10 hours**
- Our implementation (achieving one 720p image per second) shows **~37,000X speed up**

The actual performance of Square Root requires division, which BGV and BASALISC do not enable

# Outline

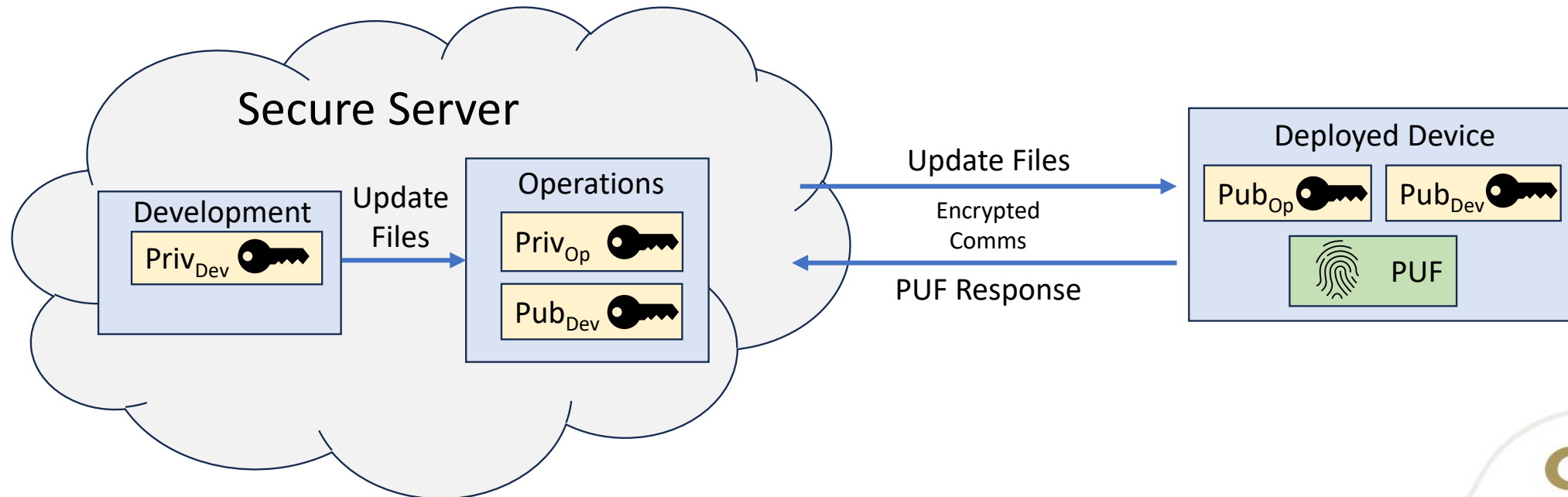
- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- **PUF-Based Authentication for Delivery of Software and Firmware Updates**
  - PUF Authentication
  - Two-Factor Authentication Protocol
  - Protocol Testing
- Conclusions
- List of Publications
- References

# PUF-Based Authentication for Delivery of Software and Firmware Updates



# PUF-Based Authentication for Delivery of Software and Firmware Updates

- Utilize a secure update mechanism named GridTrust [25][26][27][28][29][30]
  - The secure update mechanism is supplied to ensure unauthorized device updates and Computation Tables are never accepted and loaded by the remote sensor
  - The update protocol includes a PUF on the device which provides authentication to the server
- Protocol components include:
    - Two public-private key pairs
    - A PUF residing on the deployed device

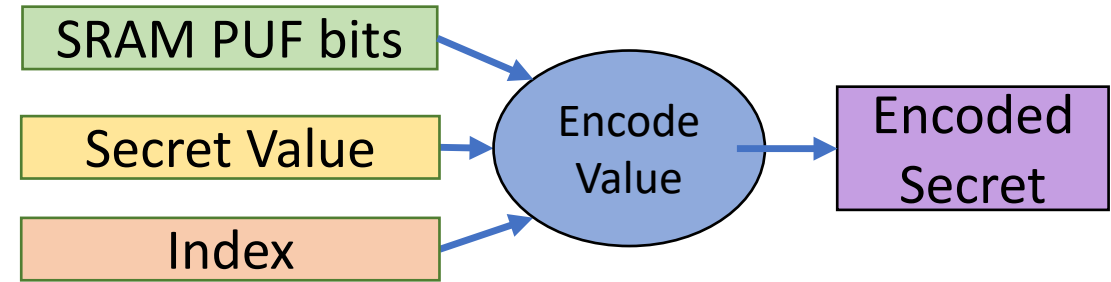


# Outline

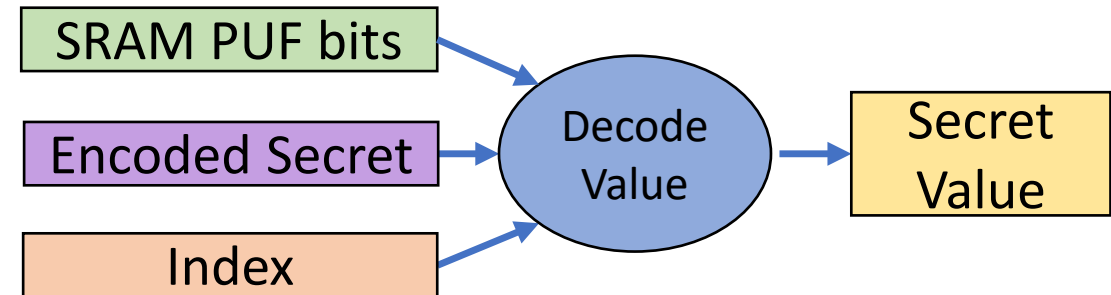
- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- **PUF-Based Authentication for Delivery of Software and Firmware Updates**
  - **PUF Authentication**
  - Two-Factor Authentication Protocol
  - Protocol Testing
- Conclusions
- List of Publications
- References

# SRAM PUF Utilization

- Utilize NXP LPC55S69 microprocessor, containing an onboard SRAM PUF
- The PUF has two main functions
  - Secret store – Use the SRAM PUF bits to encode a secret value which is stored into a protected flash memory
  - Secret retrieval – Use the SRAM PUF bits to decode a secret value from the protected flash memory



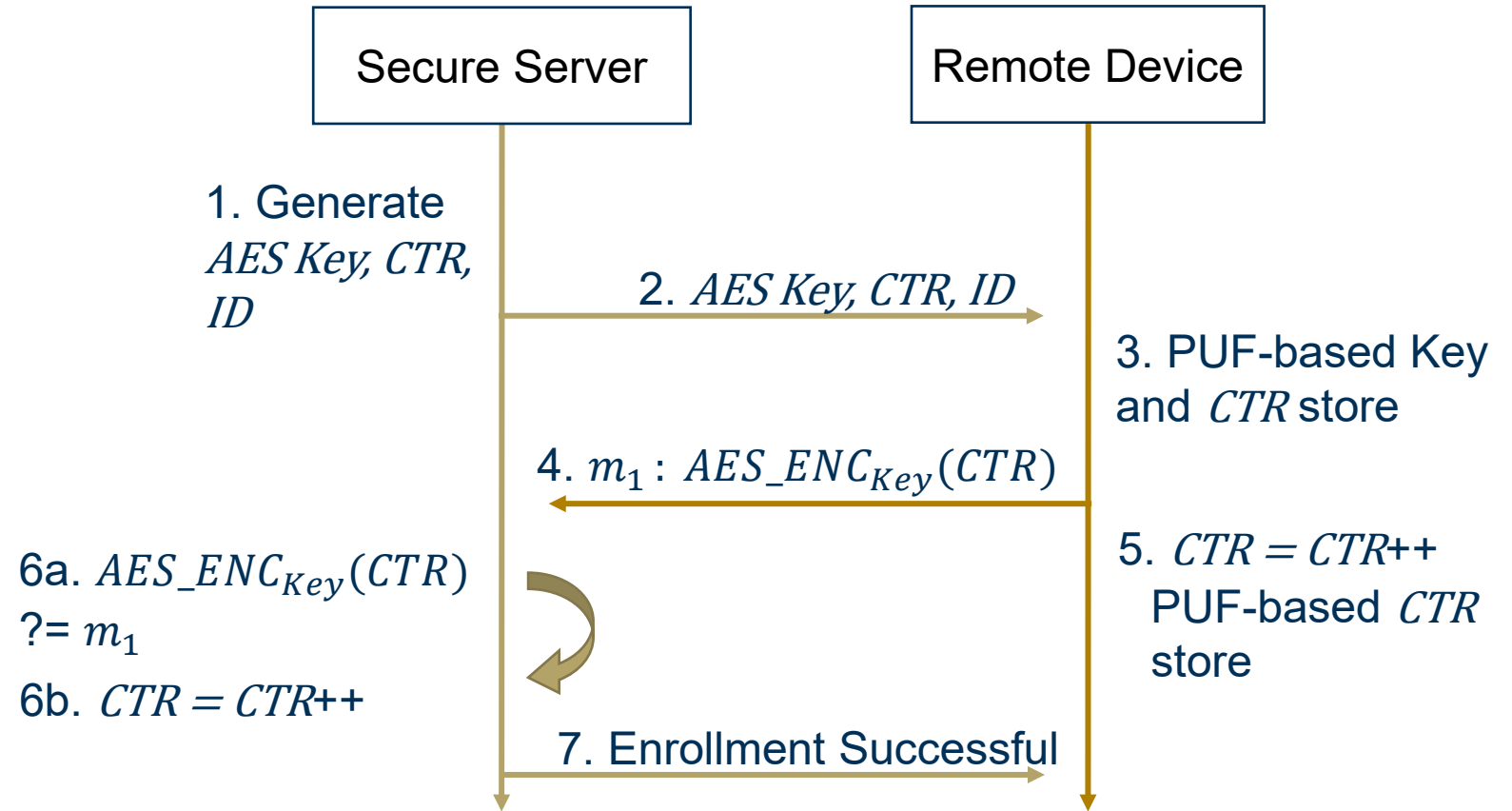
## Secret Store



## Secret Retrieval

# PUF-Based Authentication (Enrollment)

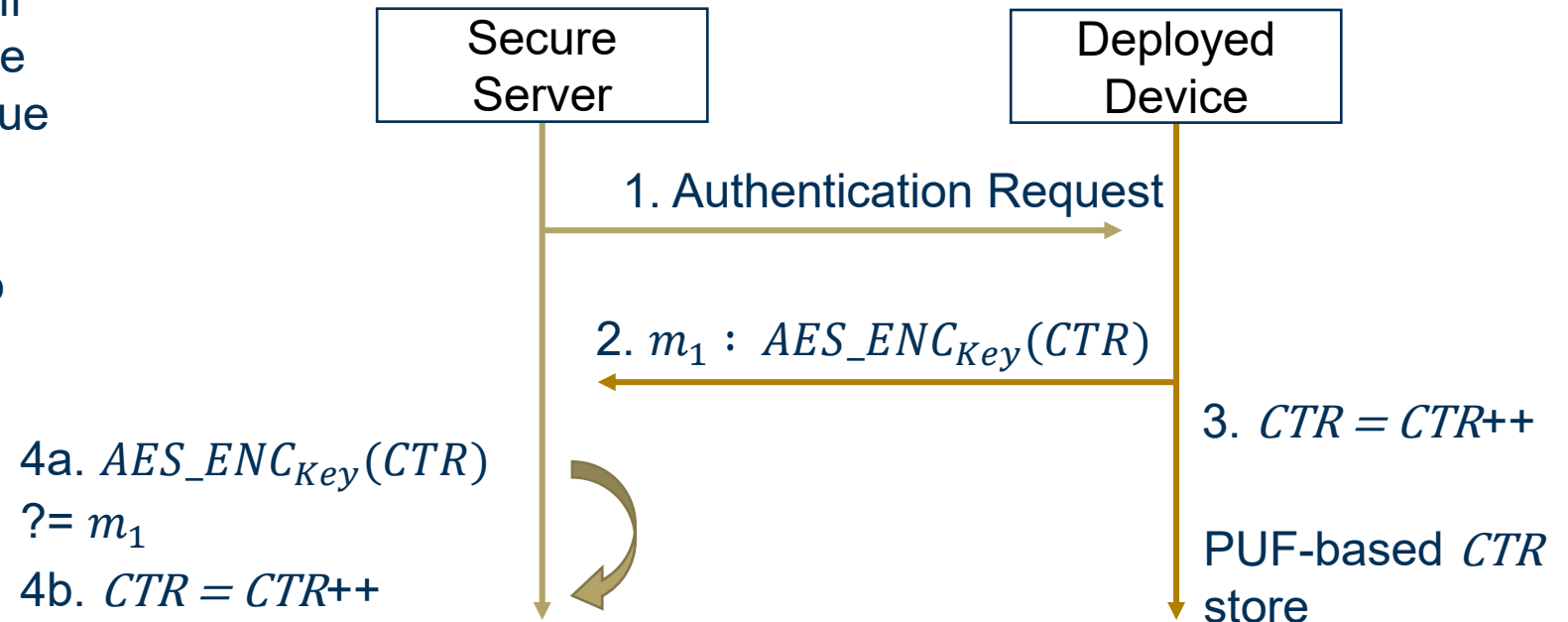
- Enrollment: Storing the PUF's unique signature in a secure database at the server
- For **SRAM PUFs** with our scheme this entails providing a key(s) to the PUF





# PUF-Based Authentication (Normal Operation)

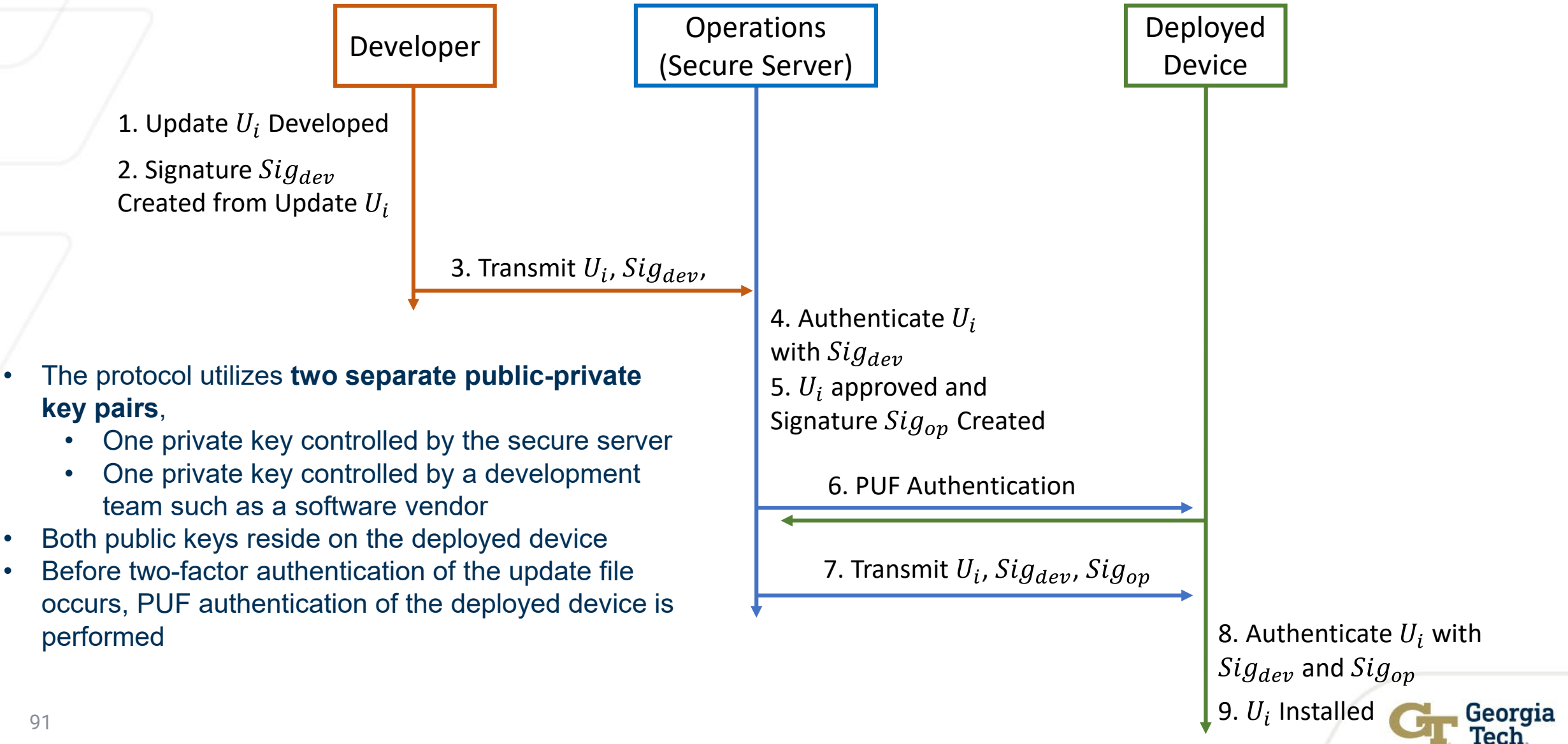
- The device authenticates itself to the server by presenting the correct encrypted counter value
- Both the device and server increment the stored CTRs to prevent replay attacks



# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- **PUF-Based Authentication for Delivery of Software and Firmware Updates**
  - PUF Authentication
  - **Two-Factor Authentication Protocol**
  - Protocol Testing
- Conclusions
- List of Publications
- References

# Two-Factor Authentication

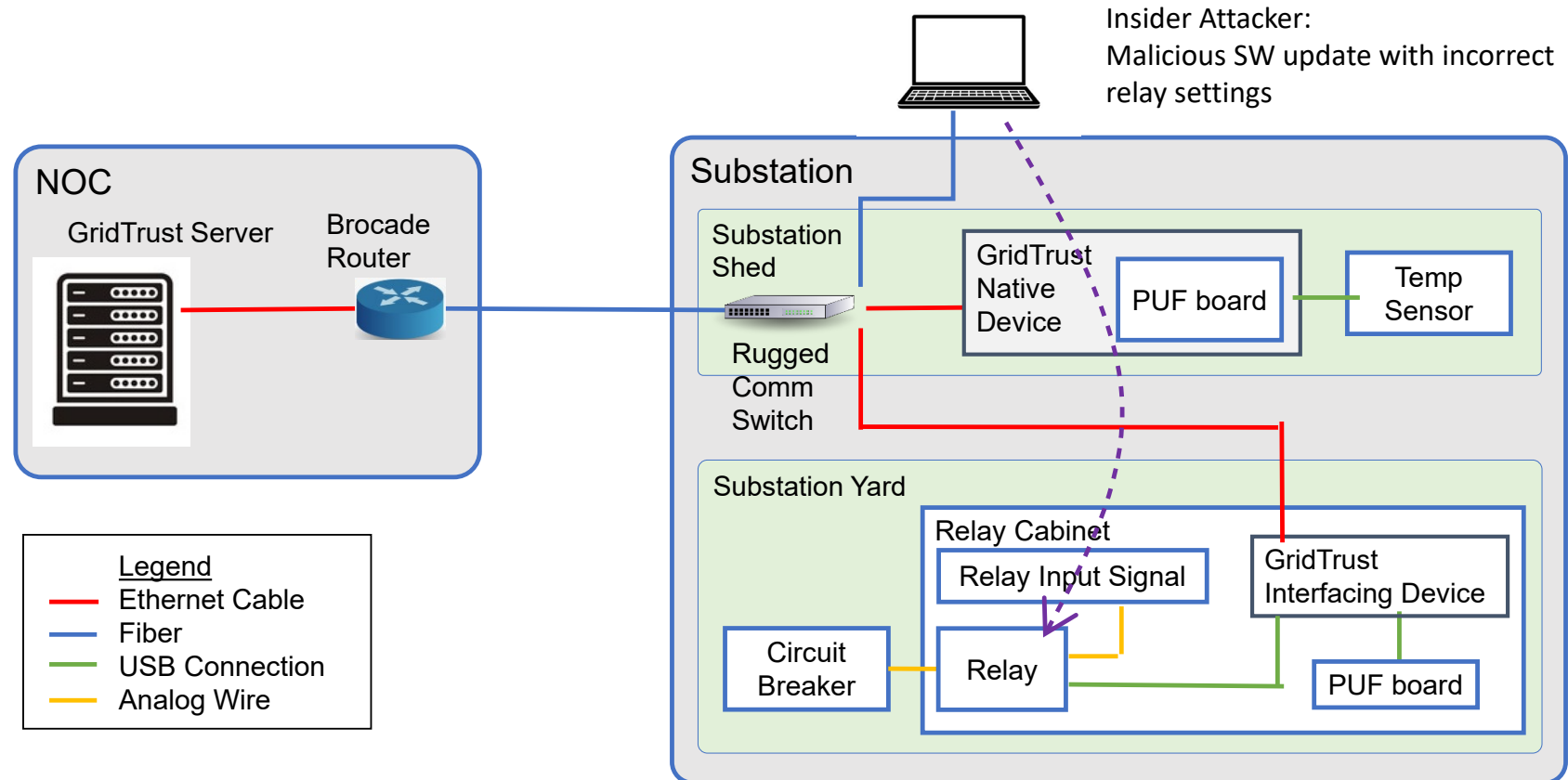


# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- **PUF-Based Authentication for Delivery of Software and Firmware Updates**
  - PUF Authentication
  - Two-Factor Authentication Protocol
  - **Protocol Testing**
- Conclusions
- List of Publications
- References

# Protocol Testing

- Protocol (called GridTrust) was prototyped in an electrical substation with cooperation from Marietta Power & Water
- Testing involved two remote devices connected to the server
- A red team audit was performed by the Georgia Tech Research Institute CIPHER Lab



# Red Team Testing

- Red team assessment performed with TLS on and off
- The assessment team was unable to forge any signature or replace any legitimate update files with invalid update files

Attack Mechanism	With PUF Protocol?	Update Secure	
		TLS On	TLS Off
No Attack (Baseline)	NO	✓	✓
	YES	✓	✓
Missing/Tampered Signature	NO	✗	✗
	YES	✓	✓
Forged Certificate	NO	✗	NA
	YES	✓	NA
Man-in-the-middle (ARP*) Protocol will not prevent ARP cache poisoning, but will still block the malicious update	NO	✗	✗
	YES	✓	✓
Man-in-the-middle (HTTP proxy)	NO	✗	✗
	YES	✓	✓

✓ means the update is secure  
✗ means the update is insecure

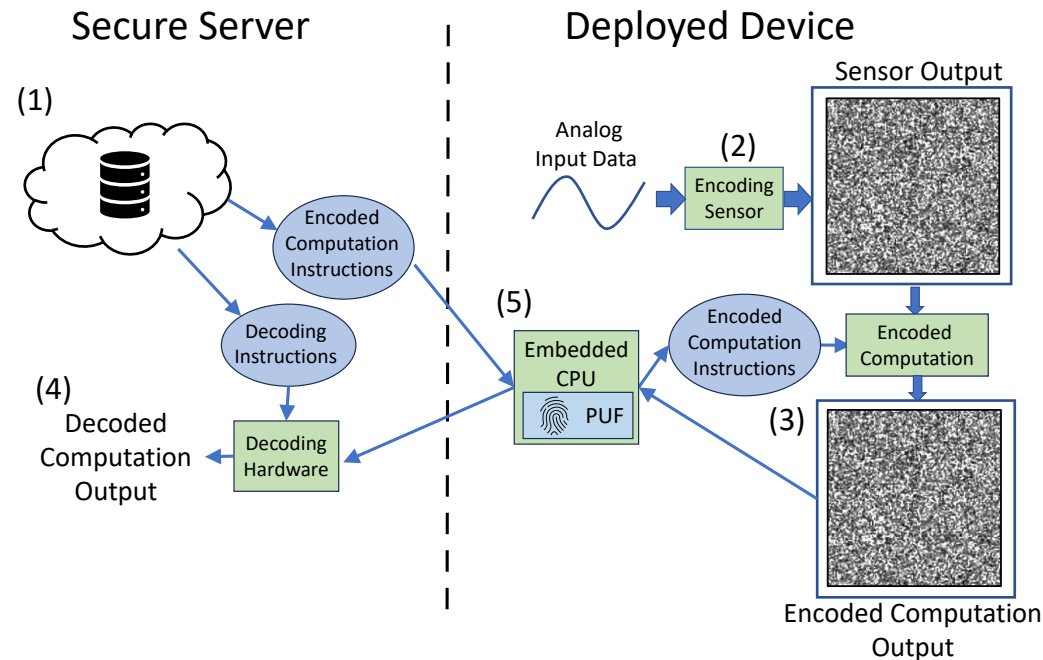
■ test was completed  
□ test not applicable

# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- **Conclusions**
- List of Publications
- References

# Conclusion

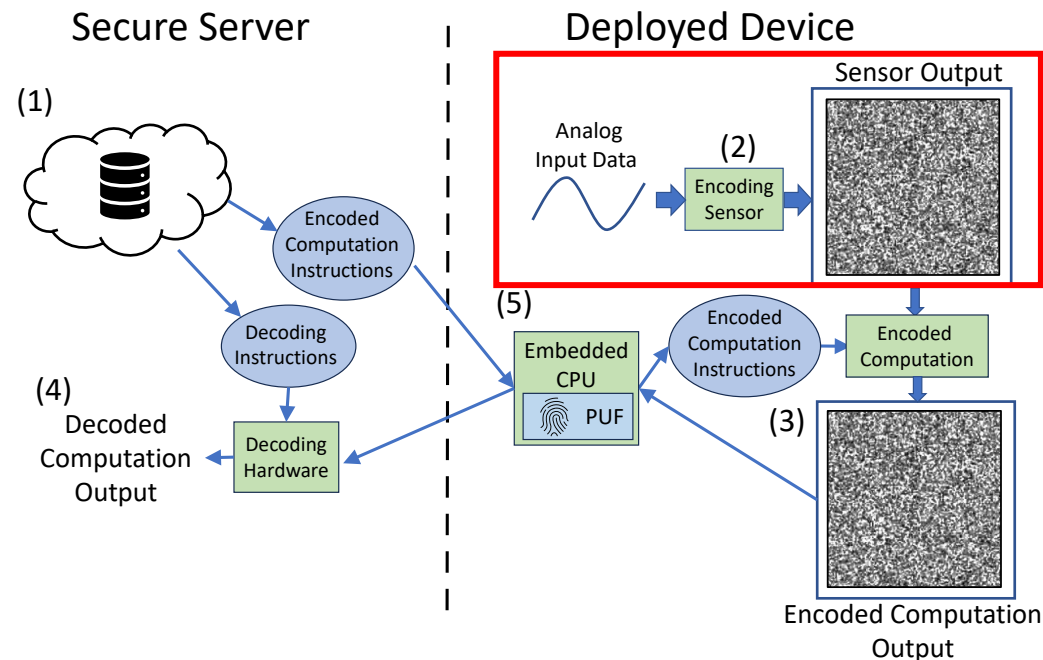
- Enhanced security provided to remotely deployed devices via three mechanisms
  - RanCode, a security enhanced ADC concept
  - RanCompute, a mechanism to perform a privacy homomorphism using RanCode
  - PUF-Based authentication for the purpose of authenticating RanCompute bitstreams and results
- RanCompute privacy homomorphism shows the ability to perform some classes of computations orders of magnitude faster than FHE implementations





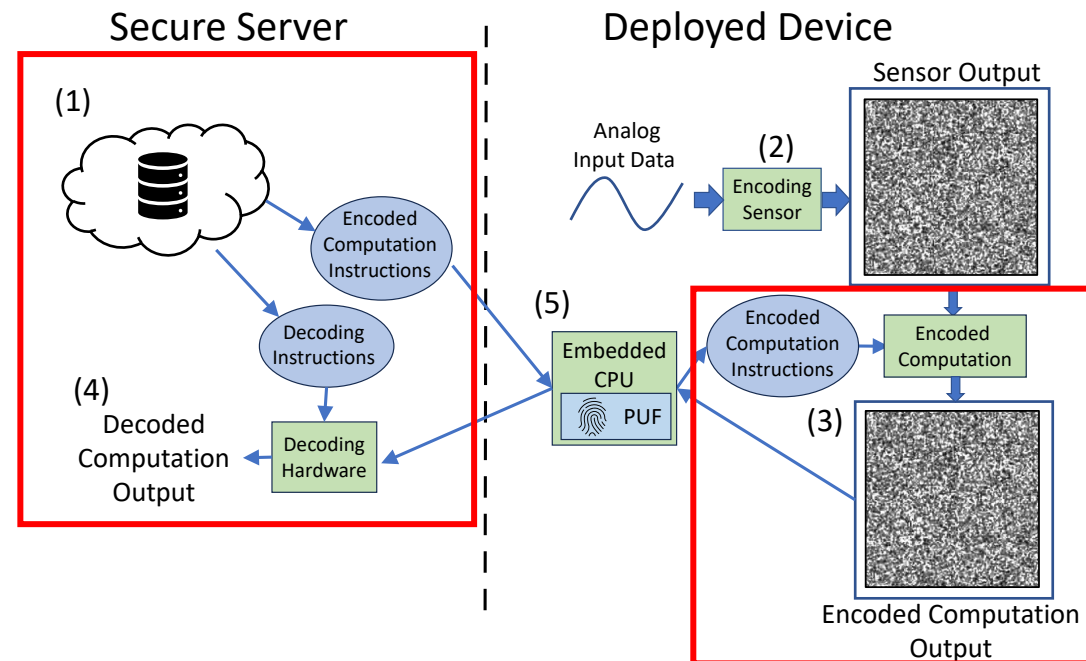
# Conclusion

- Enhanced security provided to remotely deployed devices via three mechanisms
  - **RanCode, a security enhanced ADC concept**
  - RanCompute, a mechanism to perform a privacy homomorphism using RanCode
  - PUF-Based authentication for the purpose of authenticating RanCompute bitstreams and results
- RanCompute privacy homomorphism shows the ability to perform some classes of computations orders of magnitude faster than FHE implementations



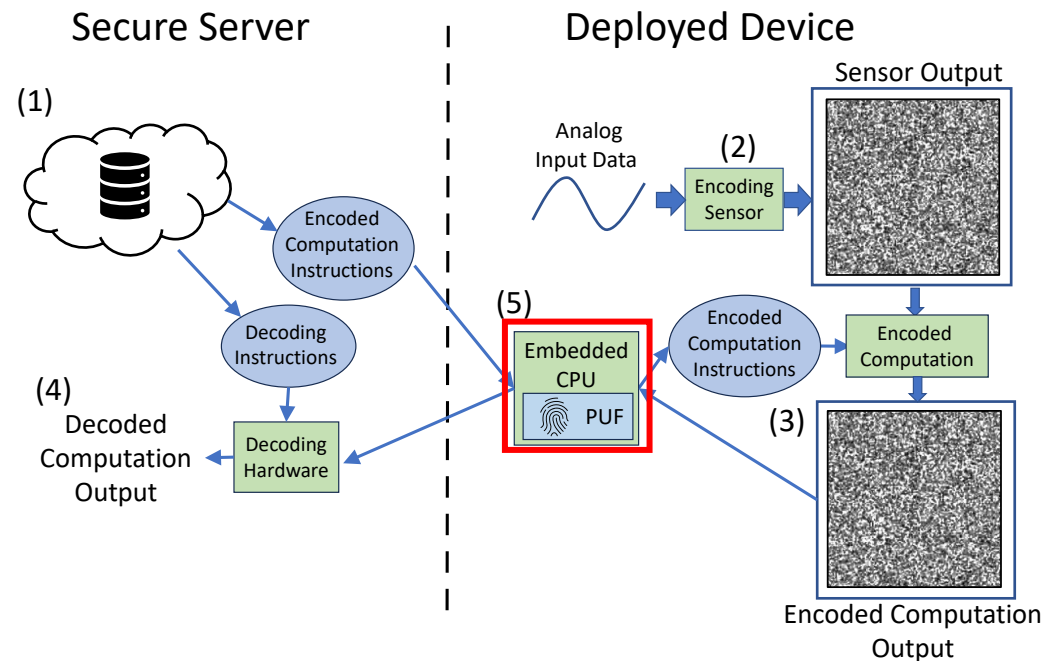
# Conclusion

- Enhanced security provided to remotely deployed devices via three mechanisms
  - RanCode, a security enhanced ADC concept
  - **RanCompute, a mechanism to perform a privacy homomorphism using RanCode**
  - PUF-Based authentication for the purpose of authenticating RanCompute bitstreams and results
- RanCompute privacy homomorphism shows the ability to perform some classes of computations orders of magnitude faster than FHE implementations



# Conclusion

- Enhanced security provided to remotely deployed devices via three mechanisms
  - RanCode, a security enhanced ADC concept
  - RanCompute, a mechanism to perform a privacy homomorphism using RanCode
  - **PUF-Based authentication for the purpose of authenticating RanCompute bitstreams and results**
- RanCompute privacy homomorphism shows the ability to perform some classes of computations orders of magnitude faster than FHE implementations



# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- **List of Publications**
- References

# Publications

- Journal publications

- K. Hutto, V. Mooney, “Implementing Privacy Homomorphism with Random Encoding and Computation Controlled by a Remote Secure Server,” ACM Transactions on Embedded Computing Systems (TECS).
- Under Review: K. Hutto, K. Vetter, V. Mooney, “Canny Edge Detection as a Privacy Homomorphism on a Remote Device,” IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems (TCAD).

- Conference publications

- K. Hutto, V. Mooney, “Late Breaking Results: COPPER: Computation Obfuscation by Producing Permutations for Encoding Randomly,” 2023 60th Design Automation Conference (DAC 60), July 2023.
- K. Hutto, S. Grijalva, V. Mooney, “RanCompute: Computational Security in Embedded Devices via Random Input and Output Encodings,” 2022 11th Mediterranean Conference on Embedded Computing (MECO’22), June 2022, **Best Paper Award (1 out of 152)**.
- K. Hutto, S. Paul, B. Newberg, V. Boyapati, Y. Vunnam, S. Grijalva, V. Mooney, “PUF-Based Two-Factor Authentication Protocol for Securing the Power Grid Against Insider Threat,” Kansas Power and Energy Conference (KPEC’22), April 2022.
- K. Hutto, S. Grijalva, V. Mooney, “Hardware-Based Randomized Encoding for Sensor Authentication in Power Grid SCADA Systems,” 2022 Texas Power and Energy Conference, February 2022.
- K. Hutto, V. Mooney, “Sensing with Random Encoding for Enhanced Security in Embedded Systems,” 2021 10th Mediterranean Conference on Embedded Computing (MECO’21), pp. 809-814, June 2021.

- Non-thesis Publications

- J. Keller, S. Paul, K. Hutto, S. Grijalva, V. Mooney, “Developing Simulation Capabilities for Supply Chain Cybersecurity of the Electricity Grid,” 2023 IEEE PES Innovative Smart Grid Technologies Latin America, November 2023.
- J. Keller, K. Hutto; S. Grijalva; V. Mooney; T. Lewis; R. Barrett; B. Holland; E. Patten, “Experimental System for Supply Chain Cyber-Security of Distribution Switch Controls,” 2024 Texas Power and Energy Conference, March 2024.

# Outline

- Introduction
- Research Overview
- Background and Prior Work
- Threat Model
- Security-Enhanced Analog-To-Digital Converter
- Implementing a Privacy Homomorphism With a Security-Enhanced ADC
- PUF-Based Authentication for Delivery of Software and Firmware Updates
- Conclusions
- List of Publications
- **References**

# References

- [1] “Forbes, Criminals Hacked A Fish Tank To Steal Data From A Casino, Accessed:2024-03-01. [Online]. Available: <https://www.forbes.com/sites/leemathews/2017/07/27/criminals-hacked-a-fish-tank-to-steal-data-from-a-casino>.
- [2] “Researchers uncover over a dozen security flaws in akuvox e11 smart intercom,” Ravie Lakshmanan, 2023. [Online]. Available: <https://thehackernews.com/2023/03/researchers-uncover-over-dozen-security.html>.
- [3] “Security startup verkada hack exposes 150,000 security cameras in tesla factories, jails, and more,” Chaim Gartenberg, 2021. [Online]. Available: <https://www.theverge.com/2021/3/9/22322122/verkada-hack-150000-security-cameras-tesla-factory-cloudflare-jails-hospitals>
- [4] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” Foundations of Secure Computation, 1978.
- [5] C. Gentry, “A fully homomorphic encryption scheme,” crypto.stanford.edu/craig, Ph.D. dissertation, Stanford University, 2009.
- [6] M. Albrecht et al., Homomorphic encryption standard, Cryptology ePrint Archive, Paper 2019/939, <https://eprint.iacr.org/2019/939>, 2019. [Online]. Available: <https://eprint.iacr.org/2019/939>.
- [7] C. Bonte, I. Iliashenko, J. Park, H. V. L. Pereira, and N. P. Smart, “Final: Faster fhe instantiated with ntru and lwe,” in Advances in Cryptology – ASIACRYPT 2022: 28<sup>th</sup> International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II, Taipei, Taiwan: Springer-Verlag, 2023, pp. 188–215, ISBN: 978-3-031-22965-7. [Online]. Available: <https://doi.org/10.1007/978-3-031-22966-4%5C%7>.
- [8] I. Chillotti, N. Gama, M. Georgieva, and M. Izabach`ene, “Tfhe: Fast fully homomorphic encryption over the torus,” Journal of Cryptology, vol. 33, no. 1, pp. 34–91, Jan. 2020. [Online]. Available: <https://doi.org/10.1007/s00145-019-09319-x>.
- [9] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, “A survey on homomorphic encryption schemes: Theory and implementation,” ACM Comput. Surv., vol. 51, no. 4, Jul. 2018. [Online]. Available: <https://doi.org/10.1145/3214303>.
- [10] Z. Brakerski and V. Vaikuntanathan, “Fully homomorphic encryption from ring-lwe and security for key dependent messages,” in Advances in Cryptology – CRYPTO 2011, P. Rogaway, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 505–524, ISBN: 978-3-642-22792-9.
- [11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” in Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ser. ITCS ’12, Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 309–325, ISBN: 9781450311151.
- [12] R. Geelen et al., “Basalisc: Programmable hardware accelerator for bgv fully homomorphic encryption,” IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2023, no. 4, pp. 32–57, Aug. 2023. [Online]. Available: <https://tches.iacr.org/index.php/TCHES/article/view/11157>.
- [13] N. Samardzic et al., “Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data,” in Proceedings of the 49th Annual International Symposium on Computer Architecture, ser. ISCA ’22, New York, New York: Association for Computing Machinery, 2022, pp. 173–187, ISBN: 9781450386104. [Online]. Available: <https://doi.org/10.1145/3470496.3527393>.
- [14] P. Socha, V. Miřkovsk´y, and M. Novotn´y, “A comprehensive survey on the noninvasive passive side-channel analysis,” Sensors, vol. 22, no. 21, pp. 80–96, 2022. [Online].
- [15] K. Hutto and V. Mooney, “Sensing with random encoding for enhanced security in embedded systems,” in 2021 10th Mediterranean Conference on Embedded Computing (MECO), 2021, pp. 1–6.

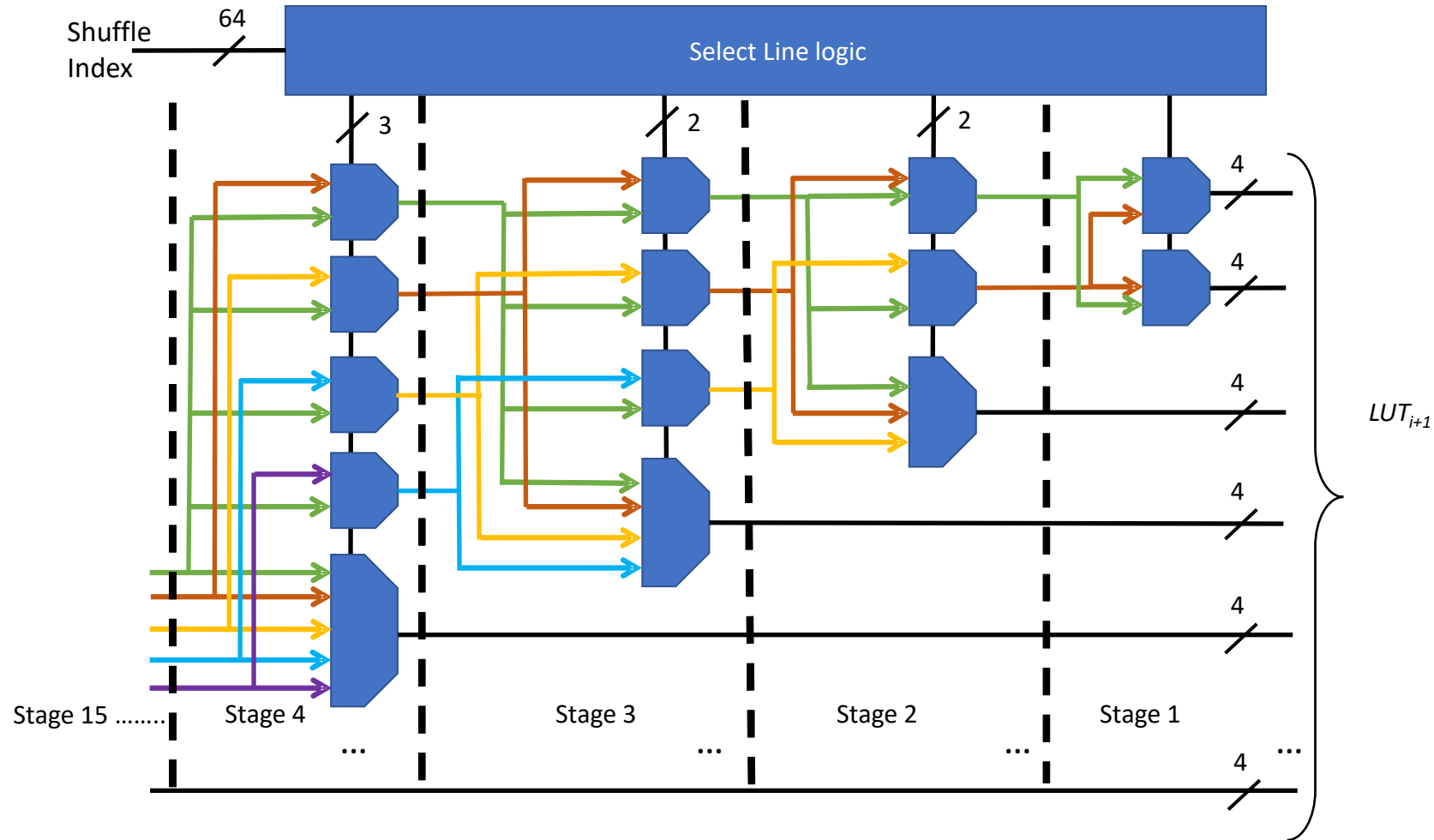
# References Cont.

- [16] K. Hutto, S. Grijalva, and V. Mooney, "Hardware-based randomized encoding for sensor authentication in power grid scada systems," in 2022 IEEE Texas Power and Energy Conference (TPEC), 2022, pp. 1–6.
- [17] D. E. Knuth, "The art of computer programming. volume 2, seminumerical algorithms," p. 192, 1997.
- [18] K. Hutto, S. Grijalva, and V. Mooney, "Rancompute: Computational security in embedded devices via random input and output encodings," in 2022 11th Mediterranean Conference on Embedded Computing (MECO), 2022, pp. 1–8.
- [19] K. Hutto and V. Mooney, "Late breaking results: Copper: Computation obfuscation by producing permutations for encoding randomly," in 2023 60th ACM/IEEE Design Automation Conference (DAC), 2023, pp. 1–2.
- [20] K. Hutto and V. Mooney, "Implementing privacy homomorphism with random encoding and computation controlled by a remote secure server," ACM Trans. Embed. Comput. Syst., Mar. 2024, Just Accepted. [Online]. Available: <https://doi.org/10.1145/3651617>.
- [21] J. Canny, "A computational approach to edge detection," IEEE transactions on pattern analysis and machine intelligence, vol. PAMI-8, no. 6, pp. 679–698, 1986.
- [22] L. E. Bassham III et al., Sp 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. National Institute of Standards and Technology, Gaithersburg, MD, 2010.
- [23] S. Halevi and V. Shoup, Design and implementation of helib: A homomorphic encryption library, Cryptology ePrint Archive, Paper 2020/1481, 2020. [Online]. Available: <https://eprint.iacr.org/2020/1481>.
- [24] R. G. Lyons, Streamlining Digital Signal Processing: A Tricks of the Trade Guidebook. Wiley-IEEE Press, 2012, ch. 25, pp. 243–250, ISBN: 9781118316948.
- [25] S. Paul, Y.-C. Chen, S. Grijalva, and V. J. Mooney, "A cryptographic method for defense against mitm cyber attack in the electricity grid supply chain," in 2022 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), 2022, pp. 1–5.
- [26] J. Keller, S. Paul, S. Grijalva, and V. J. Mooney, "Experimental setup for grid control device software updates in supply chain cyber-security," in 2022 North American Power Symposium (NAPS), 2022, pp. 1–6.
- [27] B. Newberg, S. Grijalva, and V. Mooney, "Open-source architecture for multi-party update verification for data acquisition devices," in 2022 IEEE Power and Energy Conference at Illinois (PECI), 2022, pp. 1–7.
- [28] K. Hutto et al., "Puf-based two-factor authentication protocol for securing the power grid against insider threat," in 2022 IEEE Kansas Power and Energy Conference (KPEC), 2022, pp. 1–6.
- [29] J. Keller, S. Paul, K. Hutto, S. Grijalva, and V. J. Mooney, "Developing simulation capabilities for supply chain cybersecurity of the electricity grid," in 2023 IEEE PES Innovative Smart Grid Technologies Latin America (ISGT-LA), 2023, pp. 205–209.
- [30] J. Keller et al., "Experimental system for supply chain cyber-security of distribution switch controls," in 2024 IEEE Texas Power and Energy Conference (TPEC), 2024, pp. 1–6.



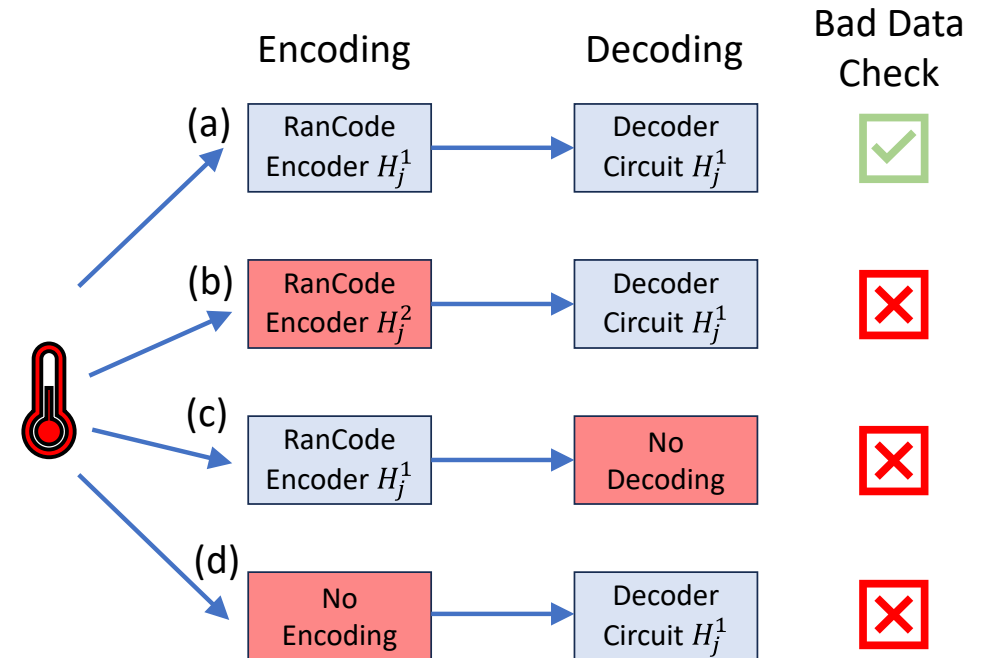
# Extra Slides

# RanCode Shuffle Unit



# Bad data detection via Chi-Square

- Bad data detection via Chi-Square
- To simulate a software or hardware replacement of the sensor data such as in a false-data injection attack, we performed four configurations of the RanCode encoding and decoding circuit

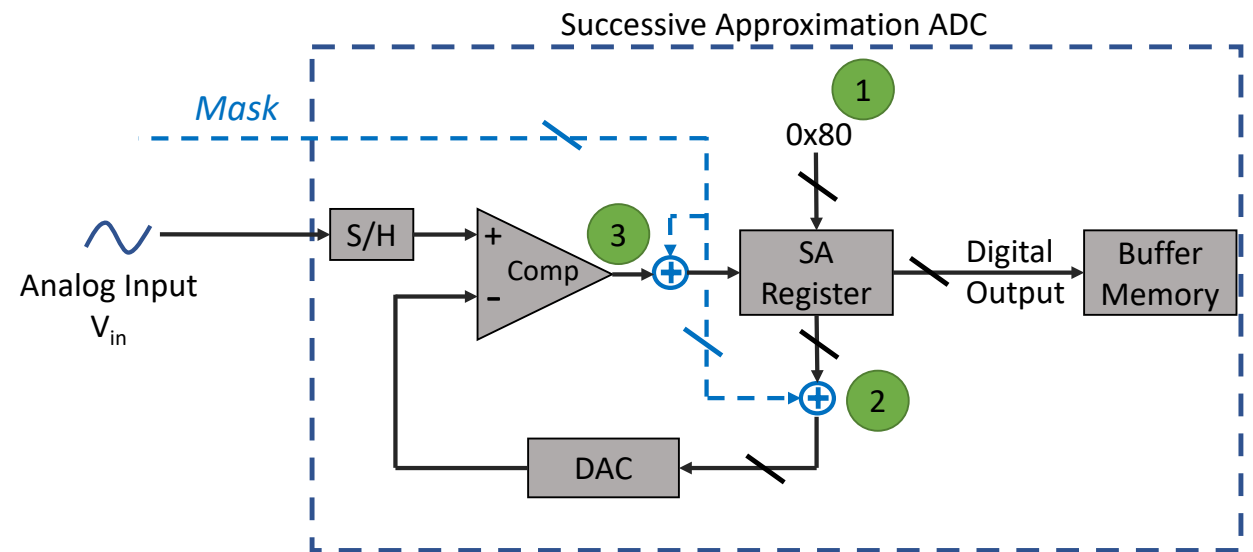
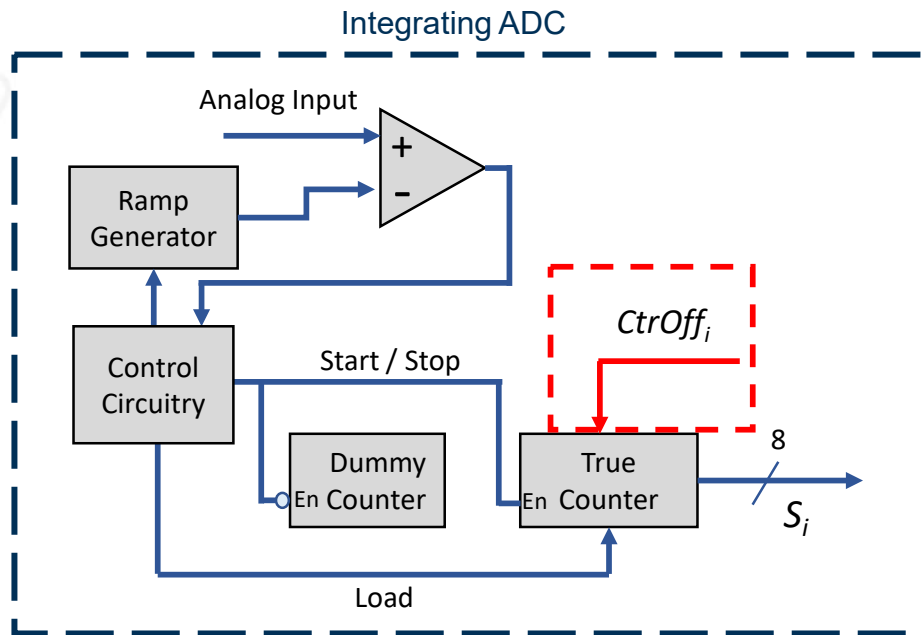


$\chi^2$  Goodness of Fit Testing

	Correct Configuration	No Encoder	No Decoder	Mismatch Key
$\chi^2$ statistic	2	1843628	6193470	5498892

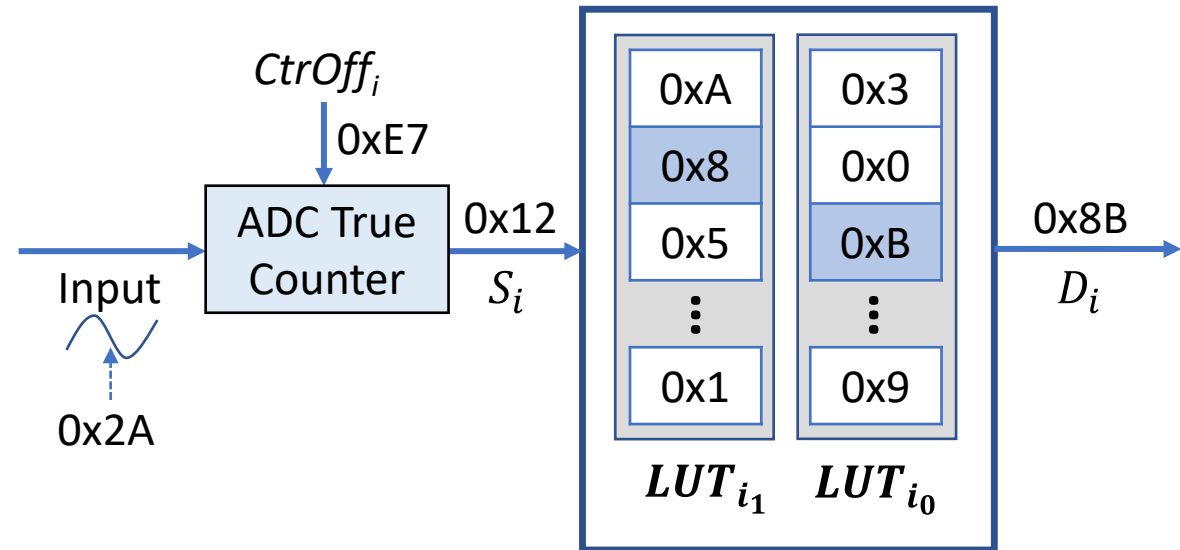
# Alternate RanCode ADC Architectures

Integrating (Dual-slope) and Successive Approximation ADCs are commonly used ADC architectures. Investigation into employing this architectures with RanCode is being performed.

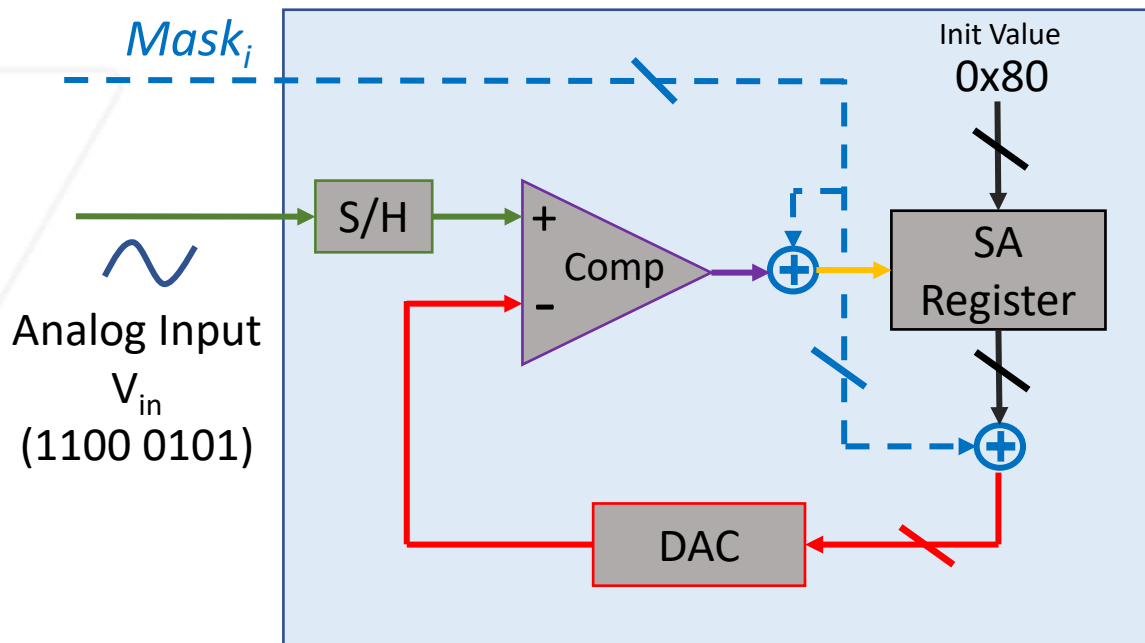


# Integrating ADC Example Operation

- An analog value equating to 0x2A is applied as input
- The input is added with the CtrOff value of 0xE7, creating 0x12
- 0x1 selects 0x8, 0x2 selects 0xB
- Final sensor output is 0x8B

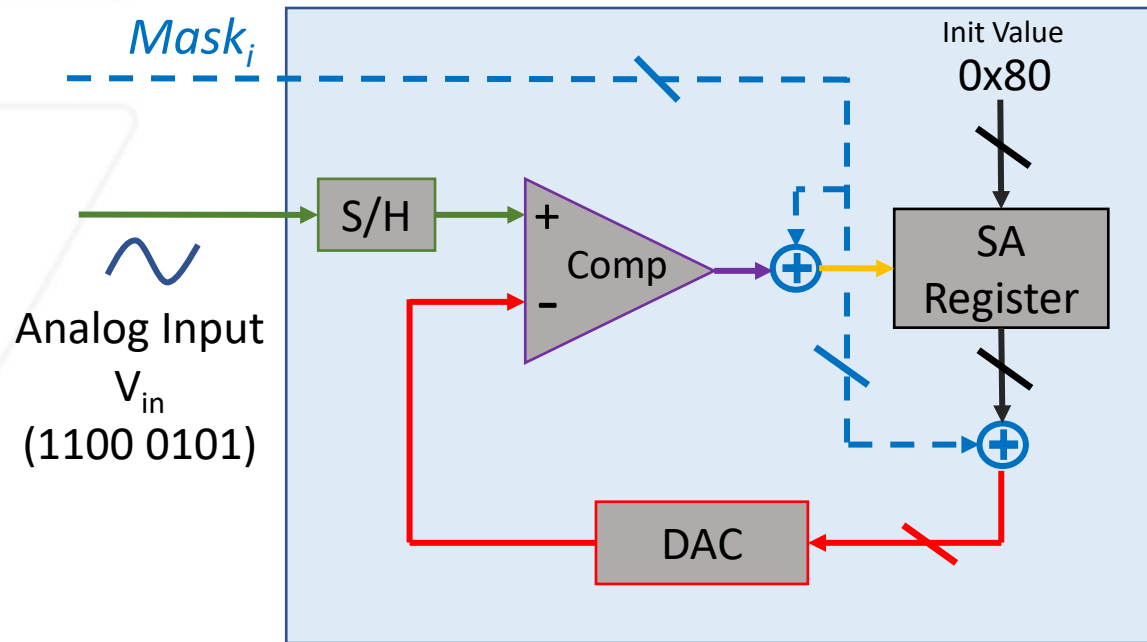


# SAR Example Operation



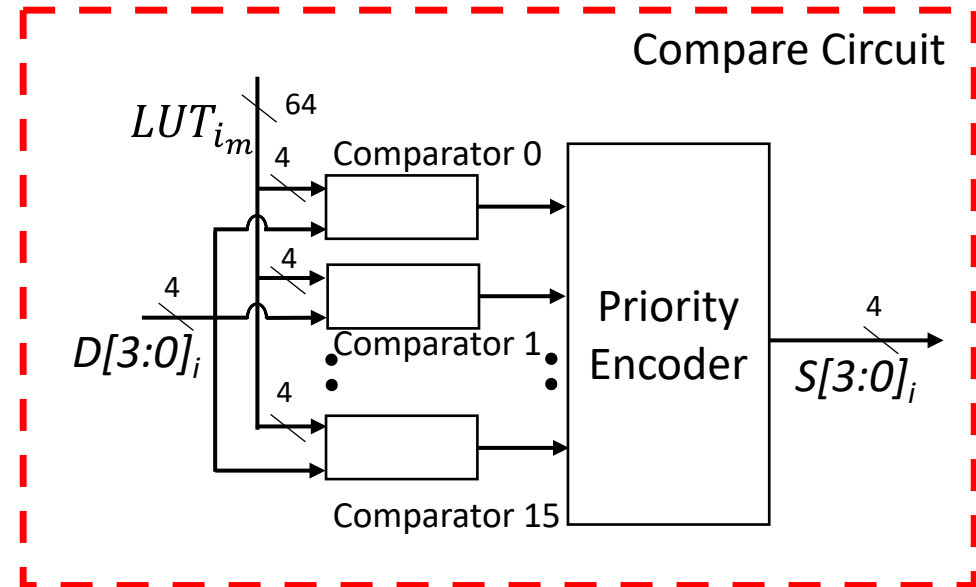
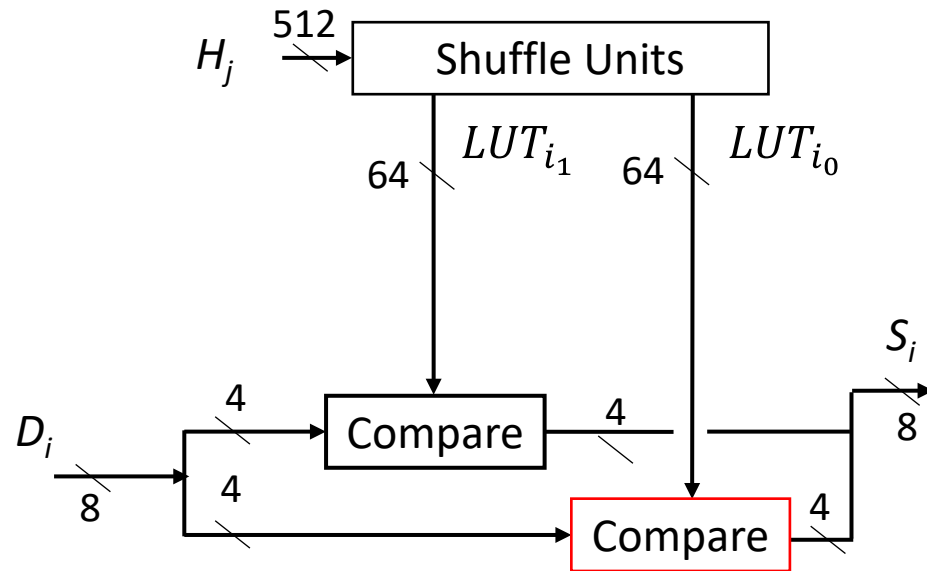
Round	SA Reg	Mask	Unmasked Reg	Input	Comp Out	Masked Comp
0	1000 0000	0000 0000	1000 0000	1100 0101	1	1
1	1100 0000	0000 0000	1100 0000	1100 0101	1	1
2	1110 0000	0000 0000	1110 0000	1100 0101	0	0
3	1101 0000	0000 0000	1101 0000	1100 0101	0	0
4	1100 1000	0000 0000	1100 1000	1100 0101	0	0
5	1100 0100	0000 0000	1100 0100	1100 0101	1	1
6	1100 0110	0000 0000	1100 0110	1100 0101	0	0
7	1100 0101	0000 0000	1100 0101	1100 0101	1	1
Result	1100 0101					

# SAR Example Operation



Round	SA Reg	Mask	Unmasked Reg	Input	Comp Out	Masked Comp
0	1000 0000	1010 0101	1000 0000	1100 0101	1	0
1	0100 0000	1010 0101	1100 0000	1100 0101	1	1
2	0110 0000	1010 0101	1110 0000	1100 0101	0	1
3	0111 0000	1010 0101	1101 0000	1100 0101	0	0
4	0110 1000	1010 0101	1100 1000	1100 0101	0	0
5	0110 0100	1010 0101	1100 0100	1100 0101	1	0
6	0110 0010	1010 0101	1100 0110	1100 0101	0	0
7	0110 0001	1010 0101	1100 0101	1100 0101	1	0
Result	0110 0000					

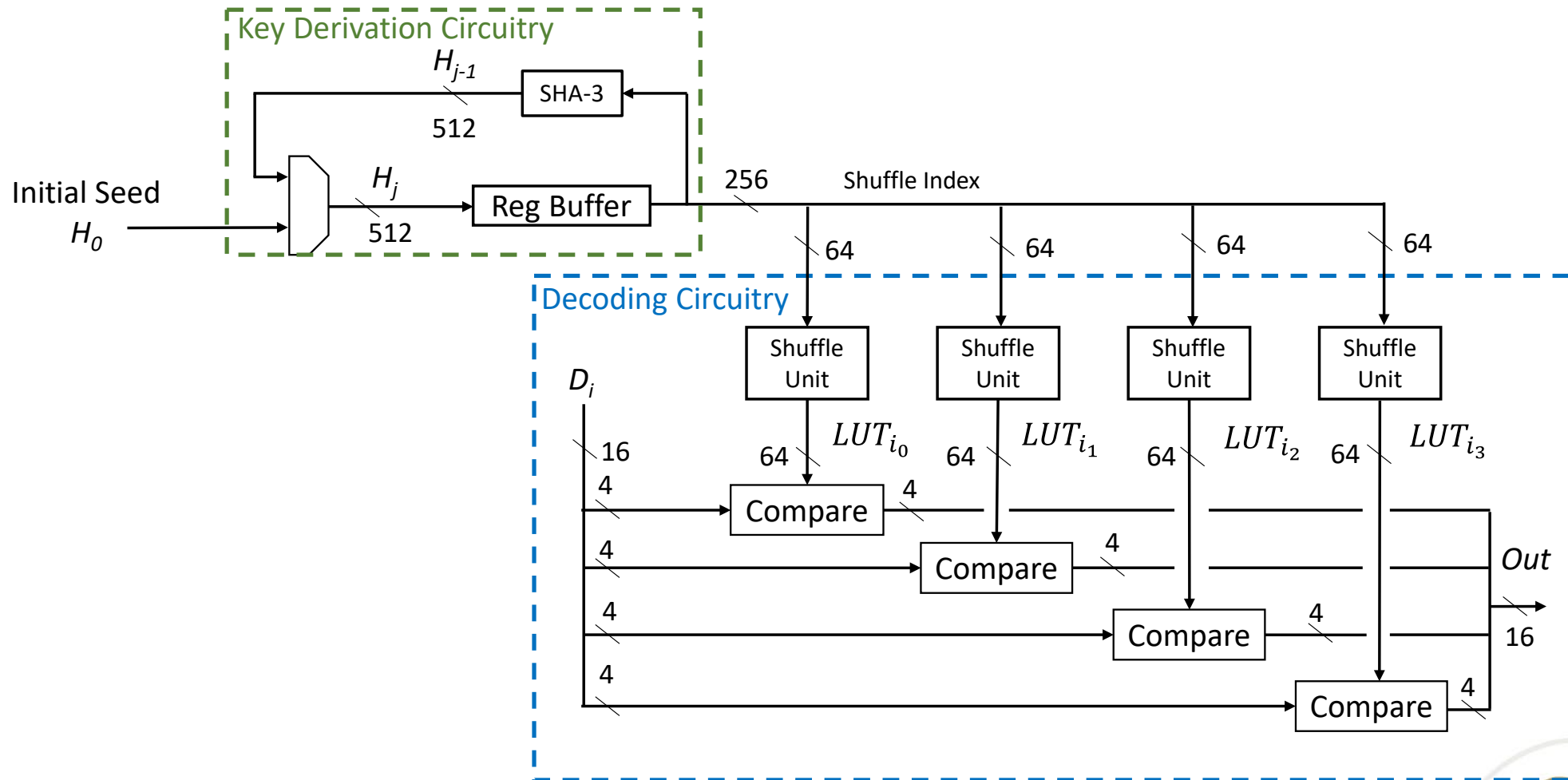
# RanCode Decoding Circuit



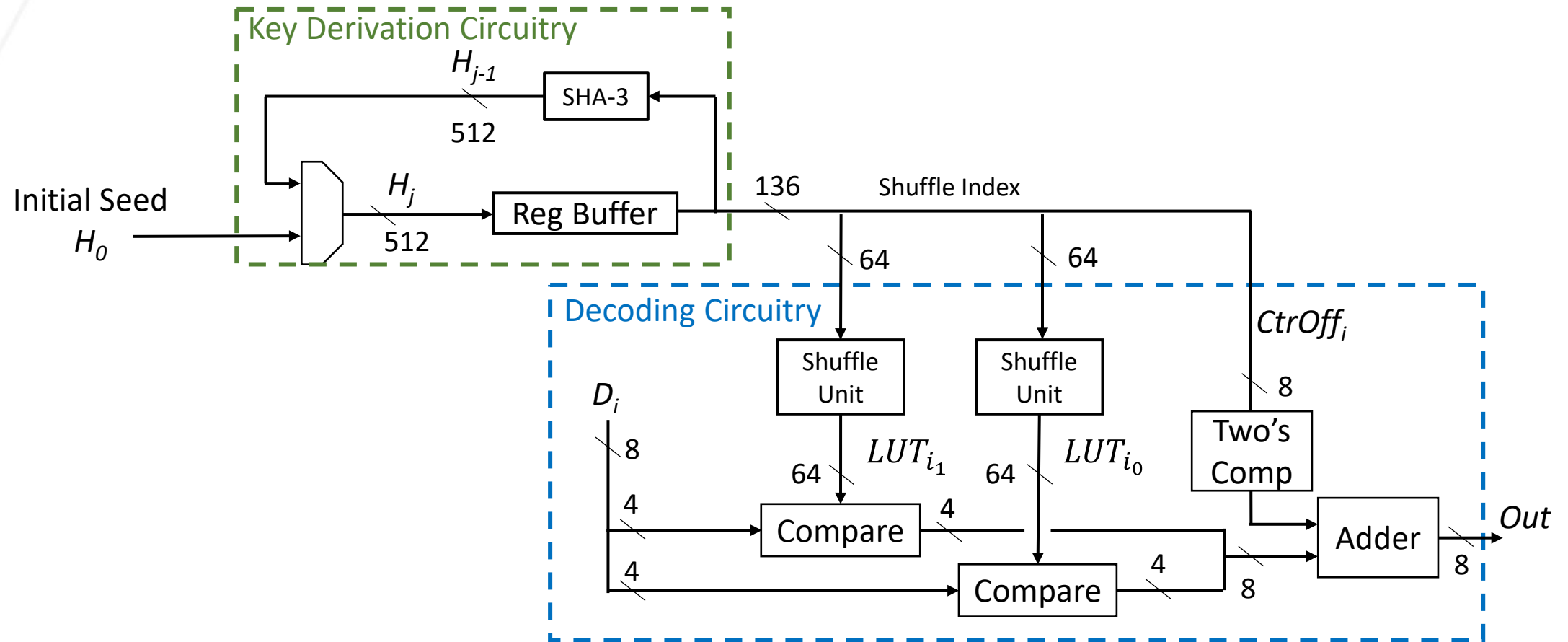
With the same starting value for  $H_0$ , decoding is accomplished by reordering the encoding components



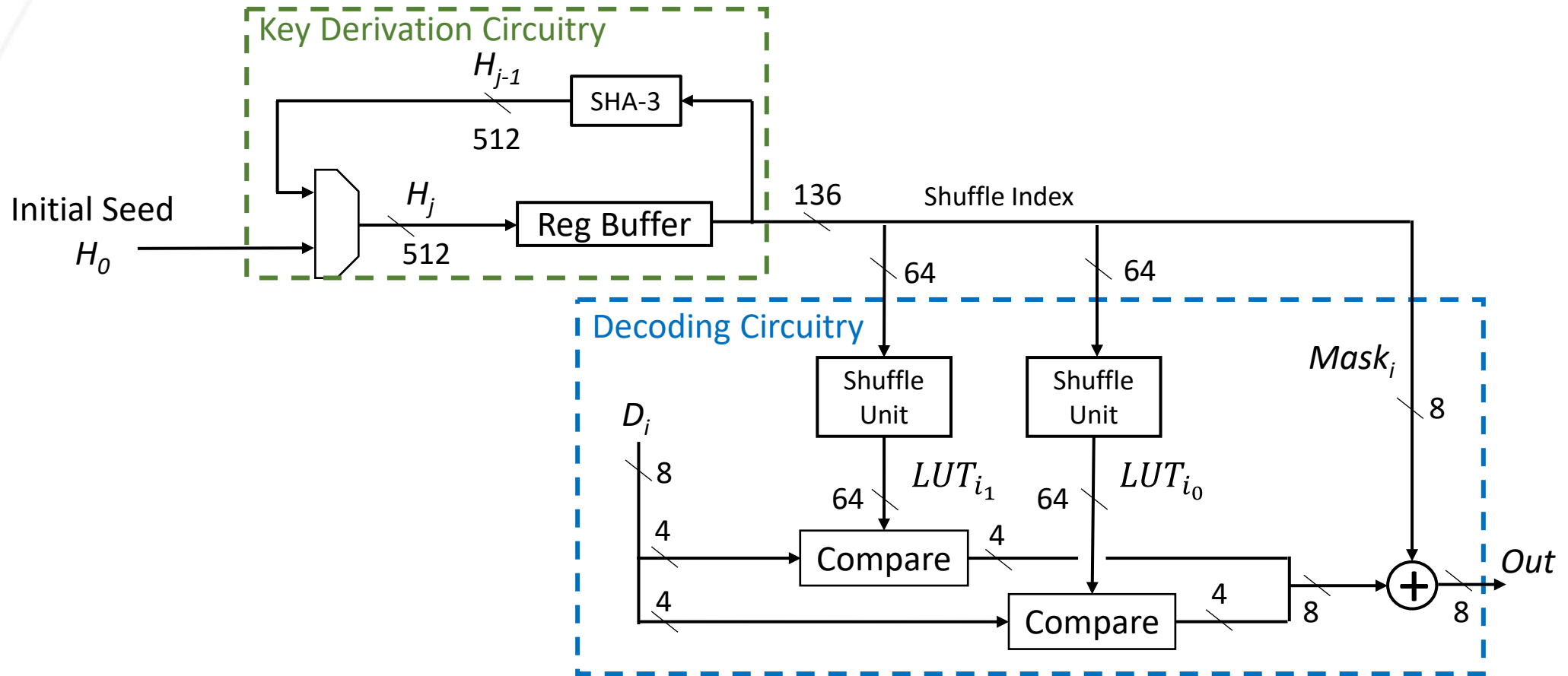
# RanCode Flash Decoder



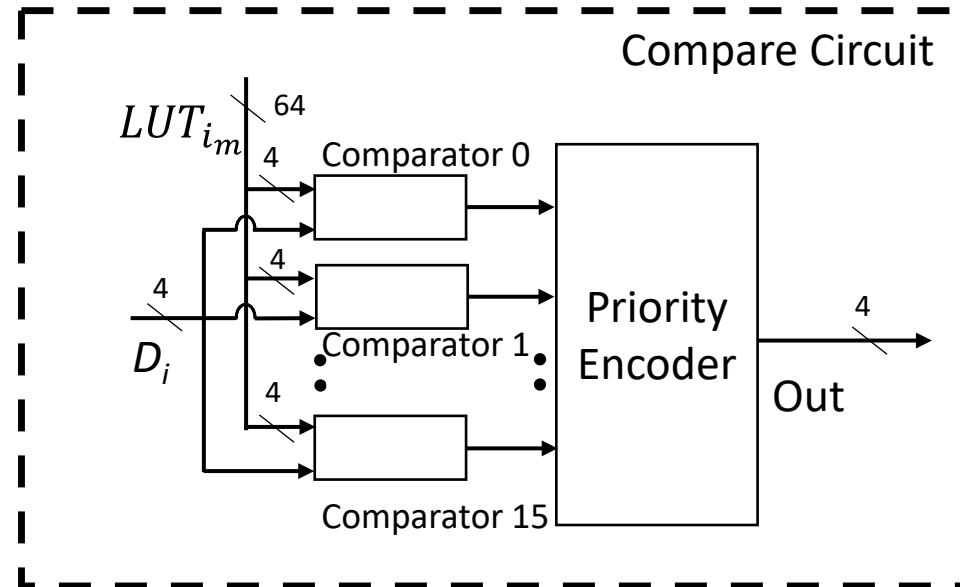
# RanCode Integrating Decoder



# RanCode SAR Decoder



# RanCode Decode Compare Circuit



# Security Analysis

- Pseudorandom Permutation (Knuth Shuffle Algorithm [17]) is reversible
  - Given a known arrangement of set elements and a known output, it can be easily determined what index was used in the algorithm
- With only a partial knowledge of the shuffle output, a subset of possible indices can be disregarded
- For a given set with  $k$  elements, there are  $k!$  permutations. With knowledge of the address of one element there are  $k - 1$  unknown element locations and  $(k - 1)!$  possible permutations for the remaining unknown element locations

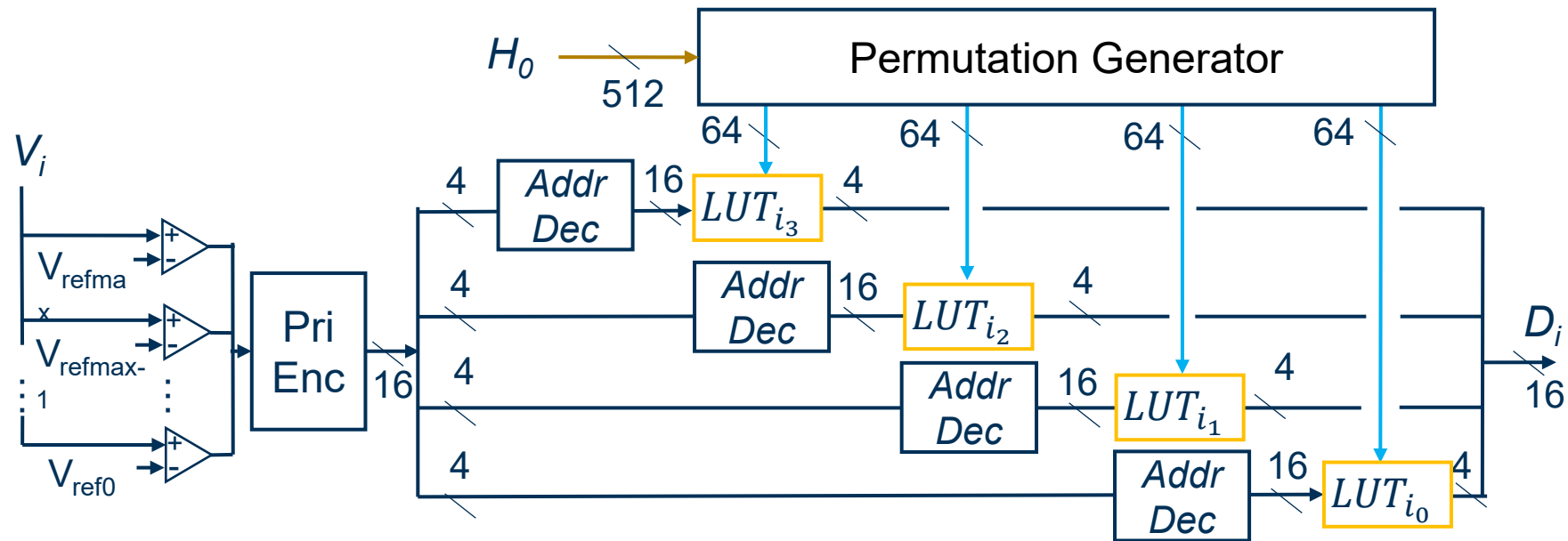
$$P(k, m) = ((k - 1)!)^m$$

# Security Analysis

- 64 bits of the  $H_j$  are used to select from the  $16!$  permutations. Each permutation held in  $LUT_{i_m}$  has a possible  $2^{64} / 16! = \sim 2^{16}$  corresponding indices
  - $I(k, m, l_s) = \frac{2^{\frac{l_s}{m}}}{k!}$
- The shuffle unit only uses half of the bits of  $H_j$ , unused bits must be accounted for as they affect the follow-on values  $H_{j+1}$
- After pruning, total number of possible indices to test is:
  - $P(k, m, l_s, l_h) = \frac{2^{\frac{l_s}{m}}}{k!} * (k - 1)!^m * 2^{l_h - l_s}$
  - $P(16, 4, 256, 512) = 2^{496}$  possible  $H_j$  after pruning

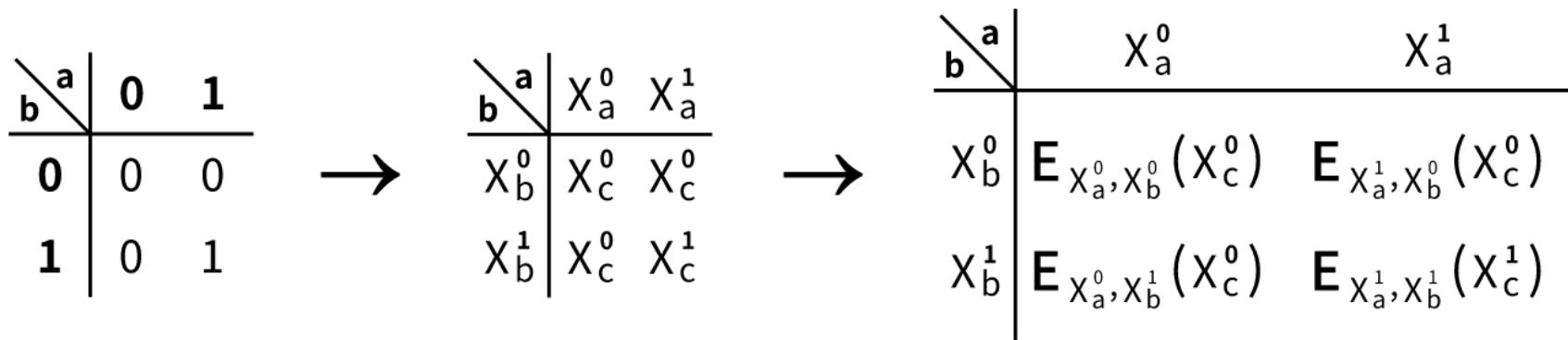
# Security Analysis

- $l_h$  = length of hash input, 512 bits
- $l_s$  = length of hash subset used for permutation, 256 bits
- $m$  = number of LUT modules, 4
- $k$  = elements permuted, 16



# Garbled Circuits

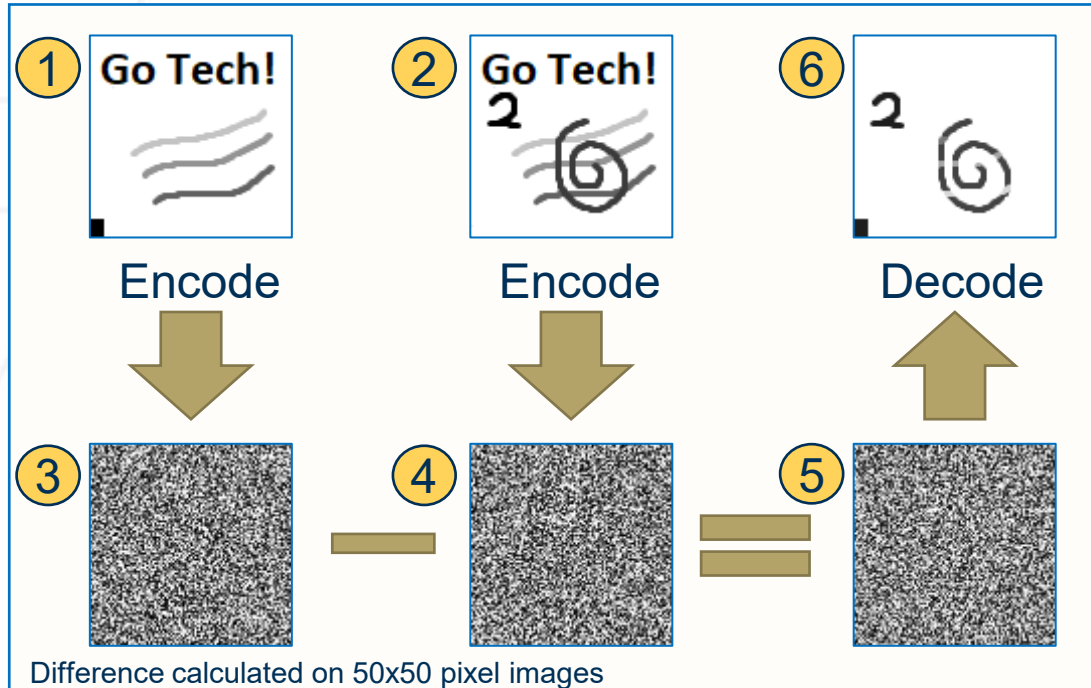
- The goal of secure multi-party computation (MPC) is to enable a group of independent data owners who do not trust each other or any common third party to jointly compute a function that depends on all of their private inputs
- Garbled Circuits: A form of MPC to allow secure computation between mistrusting parties with private inputs without a trusted third party. Involves encrypting tables with multiple keys





# RanCompute– Difference Calculation

## Lossy Precision



## Full Precision

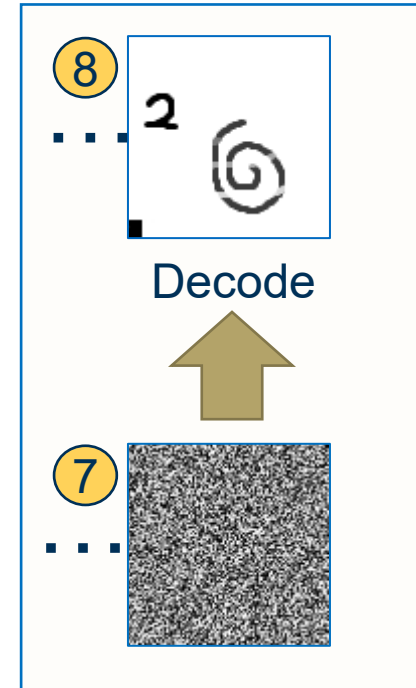


Image Difference: measure of the difference of pixel values between two images:

$$\text{Full Precision: } \forall i \rightarrow \text{Diff}_i(t, T) = |I_{t-T}(i) - I_t(i)|$$

$$\text{Lossy Precision: } \forall i \rightarrow \text{Diff}_i(t, T) = |I_{t-T}(i)[7:4] - I_t(i)[7:4]| \& \\ |I_{t-T}(i)[3:0] - I_t(i)[3:0]|$$

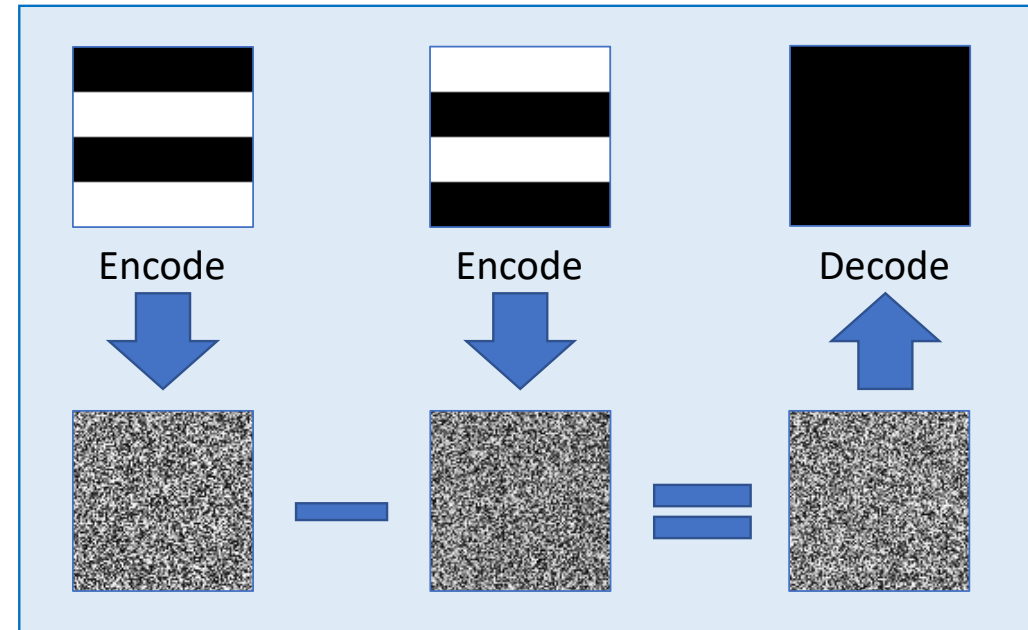
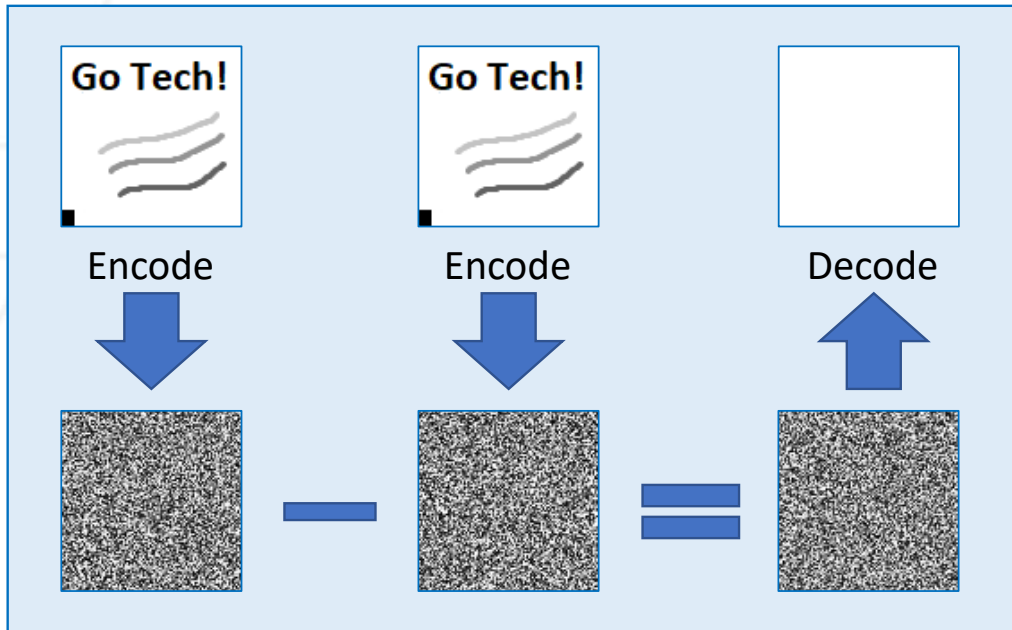
Circuit	LEs	Registers	DSPs	MHz
<b>Lossy Computation Circuitry (Equation 5.3)</b>				
Single Comp. Table	144	8	0	>500
All Comp. Tables for 50 × 50 images	2,533	8	0	>500
<b>Other Circuitry</b>				
RanCode	2,499	2,486	22	157

- Synthesis conducted targeting the Cyclone V 5CSXFC6D6F31C6
- With a high-performance SHA-3 core and proven dynamic FPGA technology, image difference can be calculated for 720p images at 20fps.

Table 1.3: Computation Table Data Usage

	Per Pixel		Per 720p Image	
	Lossy	Full	Lossy	Full
Comp Tables	1	1	921,600	921,600
Total Comp Table Rows	512	65,536	4.7e8	6.0e10
Total Comp Table Bytes	720,896	131,072	471 MB	120 GB

# Image Difference Calculations




Full precision computations, 50x50 pixels

# RanCompute Matching Outputs

- One aspect which helps to hide the identity of a digital computation is to have different computation tables with identical output frequencies
  - Output frequency – the number of times (multiplicity) a specific output appears in all possible outputs (including repeat values) resulting from a function  $F_m()$  given a finite input set
- We add a minimum number of encodings to ensure matching output frequencies of two target computations

A	B	$F_1(A, B)$ $= A + B$	A	B	$F_2(A, B)$ $= A * B$
0	0	0 ( $S_0^1$ )	0	0	0 ( $S_0^2$ )
0	1	1 ( $S_1^1$ )	0	1	0 ( $S_0^2$ )
1	0	1 ( $S_1^1$ )	1	0	0 ( $S_0^2$ )
1	1	2 ( $S_2^1$ )	1	1	1 ( $S_1^2$ )




A	B	$F_1(A, B)$ $= A + B$	A	B	$F_2(A, B)$ $= A * B$
0	0	00 ( $S_0^1$ )	0	0	00 ( $S_{0a}^2$ )
0	1	01 ( $S_1^1$ )	0	1	01 ( $S_{0b}^2$ )
1	0	01 ( $S_1^1$ )	1	0	01 ( $S_{0b}^2$ )
1	1	10 ( $S_2^1$ )	1	1	10 ( $S_1^2$ )

# RanCompute Matching Outputs

- One aspect which helps to hide the identity of a digital computation is to have different computation tables with identical output frequencies
  - Output frequency – the number of times (multiplicity) a specific output appears in all possible outputs (including repeat values) resulting from a function  $F_m()$  given a finite input set
- We add a minimum number of encodings to ensure matching output frequencies of two target computations

A	B	$F_1(A, B)$ $= A + B$	A	B	$F_2(A, B)$ $= A * B$
0	0	0 ( $S_0^1$ )	0	0	0 ( $S_0^2$ )
0	1	1 ( $S_1^1$ )	0	1	0 ( $S_0^2$ )
1	0	1 ( $S_1^1$ )	1	0	0 ( $S_0^2$ )
1	1	2 ( $S_2^1$ )	1	1	1 ( $S_1^2$ )




A	B	$F_1(A, B)$ $= A + B$	A	B	$F_2(A, B)$ $= A * B$
0	0	00 ( $S_0^1$ )	0	0	00 ( $S_{0a}^2$ )
0	1	01 ( $S_1^1$ )	0	1	01 ( $S_{0b}^2$ )
1	0	01 ( $S_1^1$ )	1	0	01 ( $S_{0b}^2$ )
1	1	10 ( $S_2^1$ )	1	1	10 ( $S_1^2$ )

# RanCompute Matching Outputs

- One aspect which helps to hide the identity of a digital computation is to have different computation tables with identical output frequencies
  - Output frequency – the number of times (multiplicity) a specific output appears in all possible outputs (including repeat values) resulting from a function  $F_m()$  given a finite input set
- We add a minimum number of encodings to ensure matching output frequencies of two target computations

A	B	$F_1(A, B)$ $= A + B$	A	B	$F_2(A, B)$ $= A * B$
0	0	0 ( $S_0^1$ )	0	0	0 ( $S_0^2$ )
0	1	1 ( $S_1^1$ )	0	1	0 ( $S_0^2$ )
1	0	1 ( $S_1^1$ )	1	0	0 ( $S_0^2$ )
1	1	2 ( $S_2^1$ )	1	1	1 ( $S_1^2$ )



A	B	$F_1(A, B)$ $= A + B$	A	B	$F_2(A, B)$ $= A * B$
0	0	00 ( $S_0^1$ )	0	0	00 ( $S_{0a}^2$ )
0	1	01 ( $S_1^1$ )	0	1	01 ( $S_{0b}^2$ )
1	0	01 ( $S_1^1$ )	1	0	01 ( $S_{0b}^2$ )
1	1	10 ( $S_2^1$ )	1	1	10 ( $S_1^2$ )

# RanCompute Matching Outputs

A	B	$F_1(A, B)$ $= A + B$
0	0	0 ( $S_0^1$ )
0	1	1 ( $S_1^1$ )
1	0	1 ( $S_1^1$ )
1	1	2 ( $S_2^1$ )

A	B	$F_2(A, B)$ $= A * B$
0	0	0 ( $S_0^2$ )
0	1	0 ( $S_0^2$ )
1	0	0 ( $S_0^2$ )
1	1	1 ( $S_1^2$ )

$S_0^1$  = Symbol representing zero for function  $F_1$

$S_{0a}^2$  = Symbol representing zero for function  $F_2$ , version  $a$

$S_{0b}^2$  = Symbol representing zero for function  $F_2$ , version  $b$



A	B	$F_1(A, B)$ $= A + B$
0	0	00 ( $S_0^1$ )
0	1	01 ( $S_1^1$ )
1	0	01 ( $S_1^1$ )
1	1	10 ( $S_2^1$ )

A	B	$F_2(A, B)$ $= A * B$
0	0	00 ( $S_{0a}^2$ )
0	1	01 ( $S_{0b}^2$ )
1	0	01 ( $S_{0b}^2$ )
1	1	10 ( $S_1^2$ )

# RanCompute Matching Outputs

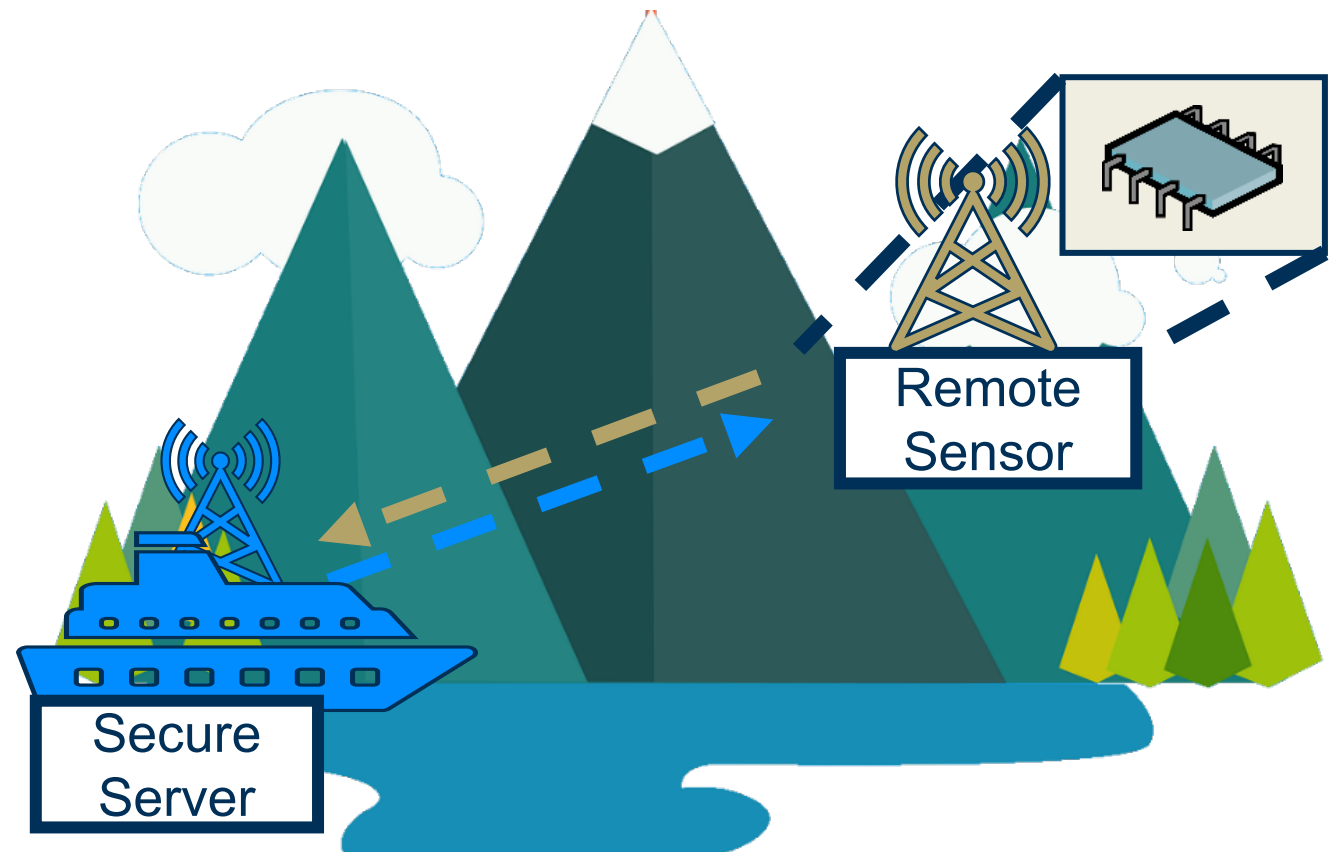
- Look-Up Table (LUT) result for each of the operations with randomized inputs and randomized outputs equalized for frequency

<b>A</b>	<b>B</b>	<b><math>F_1(A, B)</math> <math>= A + B</math></b>
0	0	11 ( $S_1^1$ )
0	1	00 ( $S_2^1$ )
1	0	10 ( $S_0^1$ )
1	1	11 ( $S_1^1$ )

<b>A</b>	<b>B</b>	<b><math>F_2(A, B)</math> <math>= A * B</math></b>
0	0	11 ( $S_{0b}^2$ )
0	1	00 ( $S_1^2$ )
1	0	10 ( $S_{0a}^2$ )
1	1	11 ( $S_{0b}^2$ )

# Research Goals

- Configure the sensor to create data directly in an encoded format with no mechanism to decode the sensor data on-chip.
- Maintain a way to perform processing on the sensor encoded data through usage of a privacy homomorphism.
- Only decode the processed data once offloaded to a secure server which possesses the capability to perform decoding.
- Now an adversary with all data on the device gains no information





# Security Analysis

- Each input pixel has a unique permutation
- Each output pixel has a unique permutation
- Each computation has a unique LUT
- Without a mapping for any of the pixels (input or output), we are not sure how an adversary can convert the images into actual values, even in the case where the single level image difference calculation is known
- Note that we are assuming that the adversary does not have a sensor capturing the exact same (or even approximately the same) images and associated pixels