Hardware-Based Randomized Encoding for Sensor Authentication in Power Grid SCADA Systems

*KEVIN HUTTO, *SANTIAGO GRIJALVA, *^VINCENT MOONEY III

*SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

^SCHOOL OF COMPUTER SCIENCE

GEORGIA INSTITUTE OF TECHNOLOGY

ATLANTA, GEORGIA

Georgia Tech School of Electrical and Computer Engineering

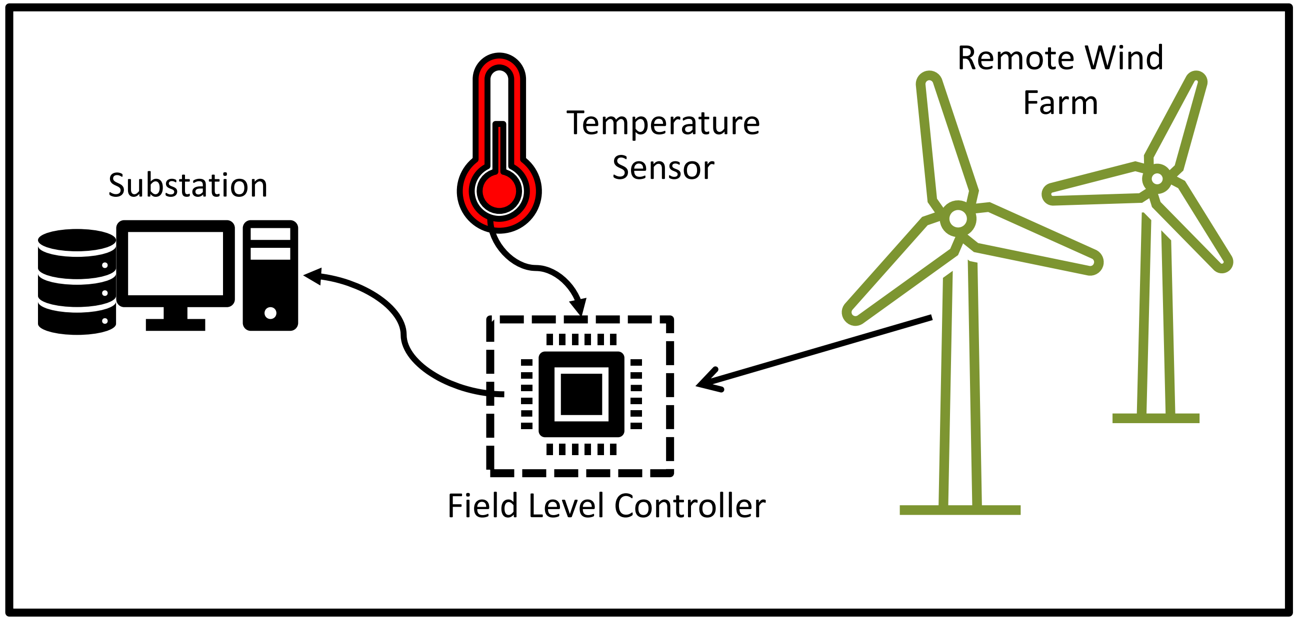
Acknowledgement

- This work has been partially supported by the U.S. Department of Energy's Office of Cybersecurity, Energy Security, and Emergency Response (CESER) under Cybersecurity for Energy Delivery Systems (CEDS) Agreement Number DE-CR0000004 to the Georgia Tech Research Corporation.

Contents

Problem Statement

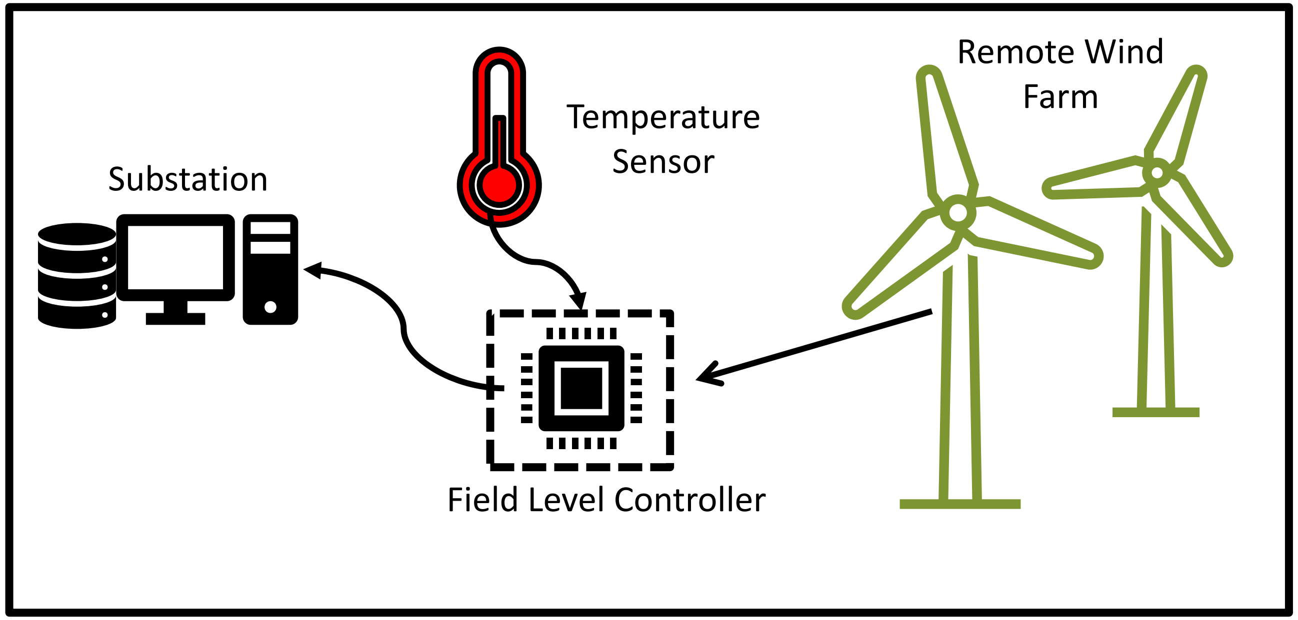
- A remote substation may be physically insecure
- An adversary may replace the sensor module with a fake device generating erroneous data
- Erroneous data in the supervisory control center could lead to power outages from unneeded protective actions through a false data injection attack [1]



Vulnerable Remote Substation

Problem Statement

- Goal: Design a lightweight sensor module which cannot be functionally replicated or replaced by an adversary without automatic notice from the control center
- Sensor module should protect against both physical replacement and tampering of the post-sensed data

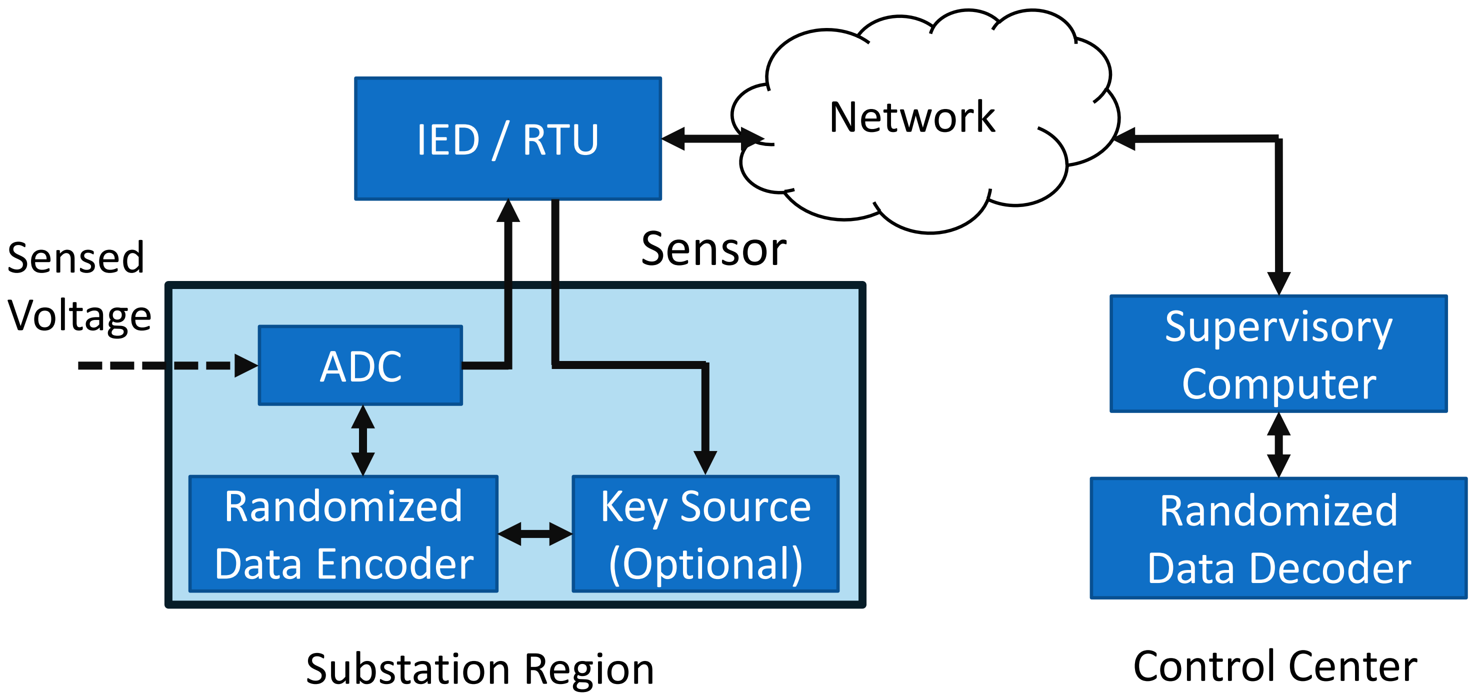


Vulnerable Remote Substation

Contents

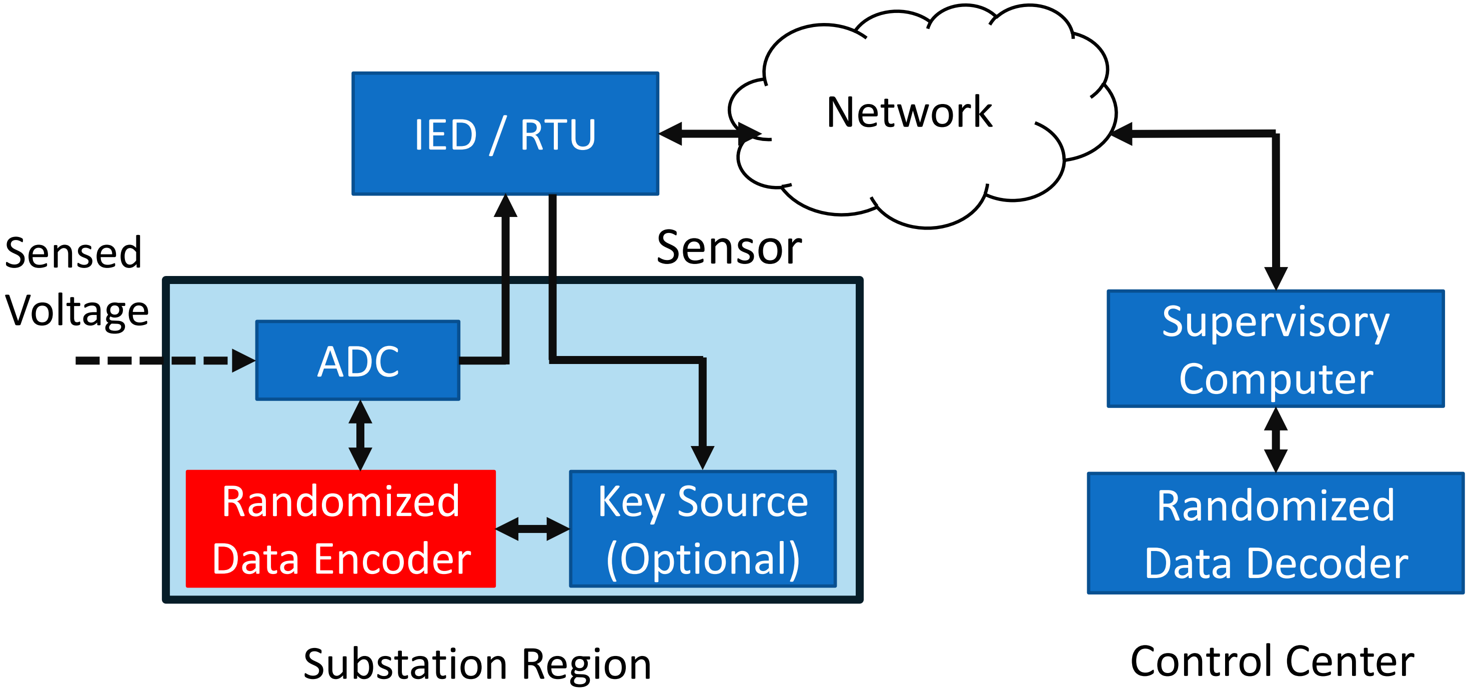
Target Architecture

- Analog to digital conversion and data encoding are conducted on the same chip
- Encoding is a function of a loadable key
- The ADC directly outputs encoded data based on a key known to a control center
  - No registered unencoded data exists in the sensor



Target Architecture

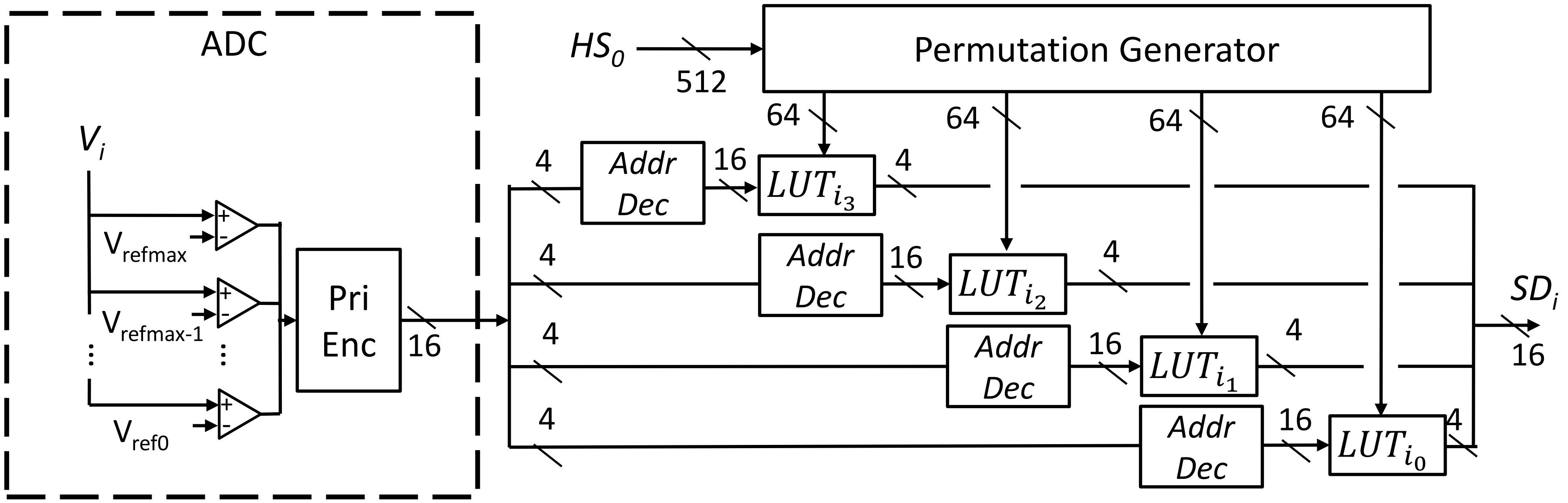
- Analog to digital conversion and data encoding are conducted on the same chip
- Encoding is a function of a loadable key
- The ADC directly outputs encoded data based on a key known to a control center
  - No registered unencoded data exists in the sensor



Contents

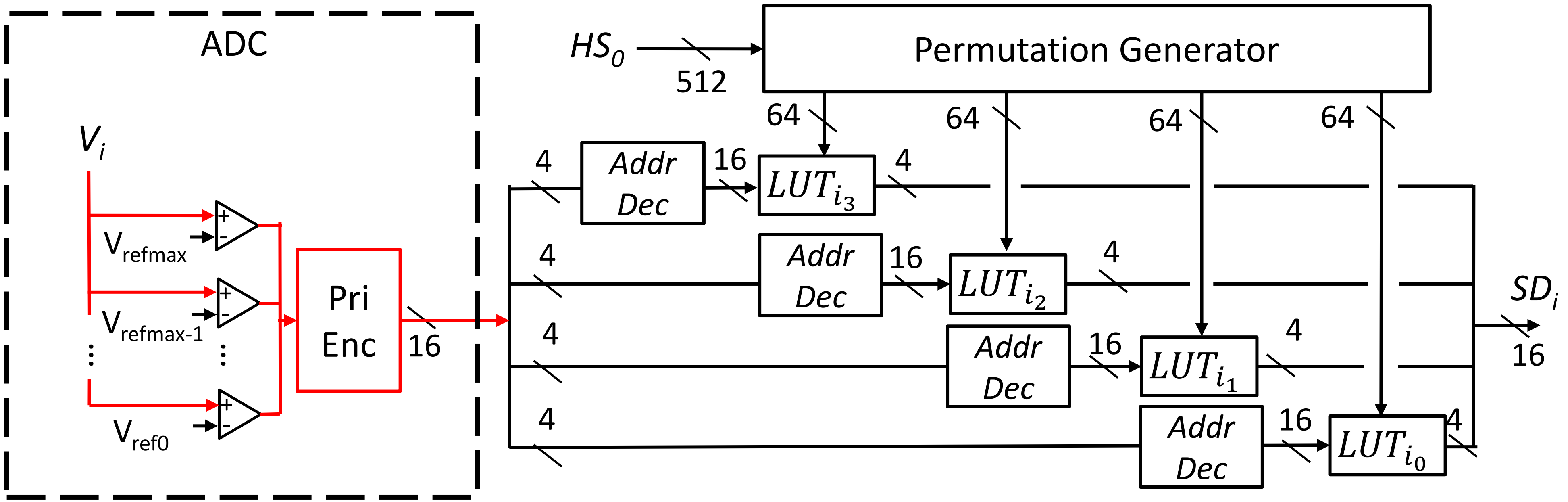
Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



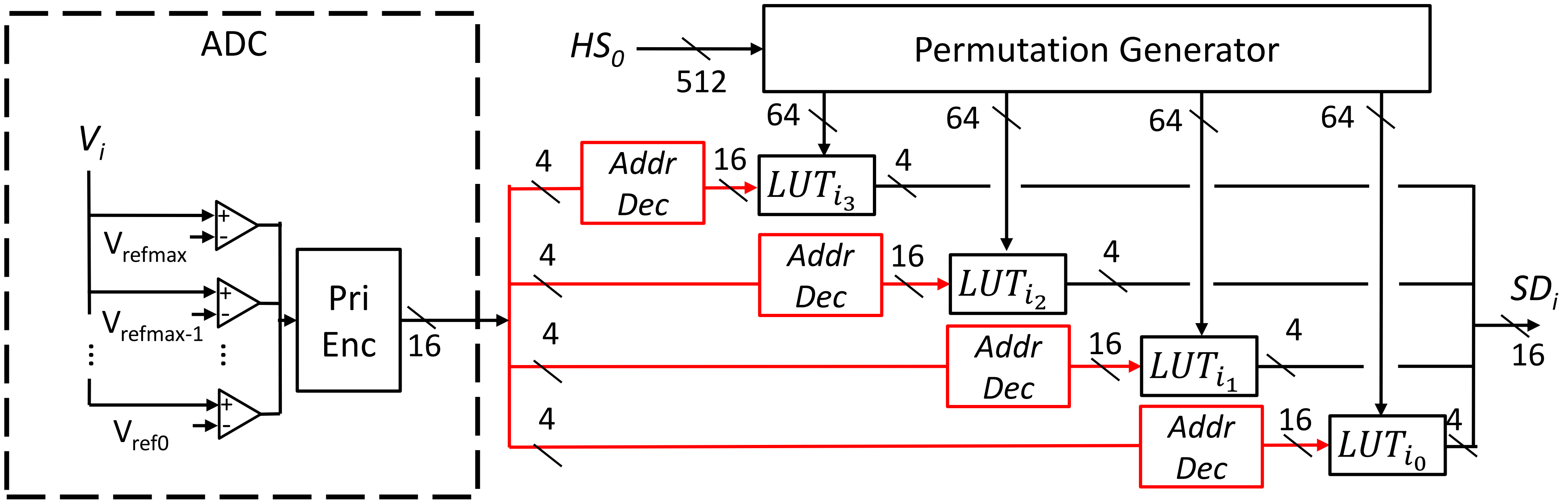
Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



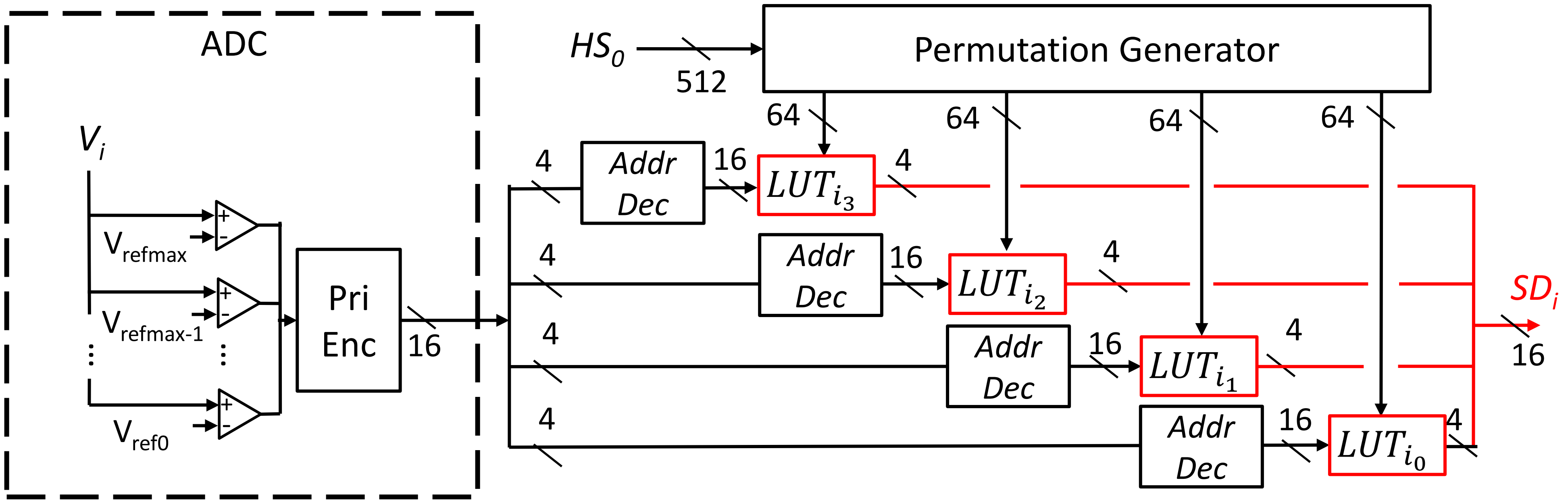
Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



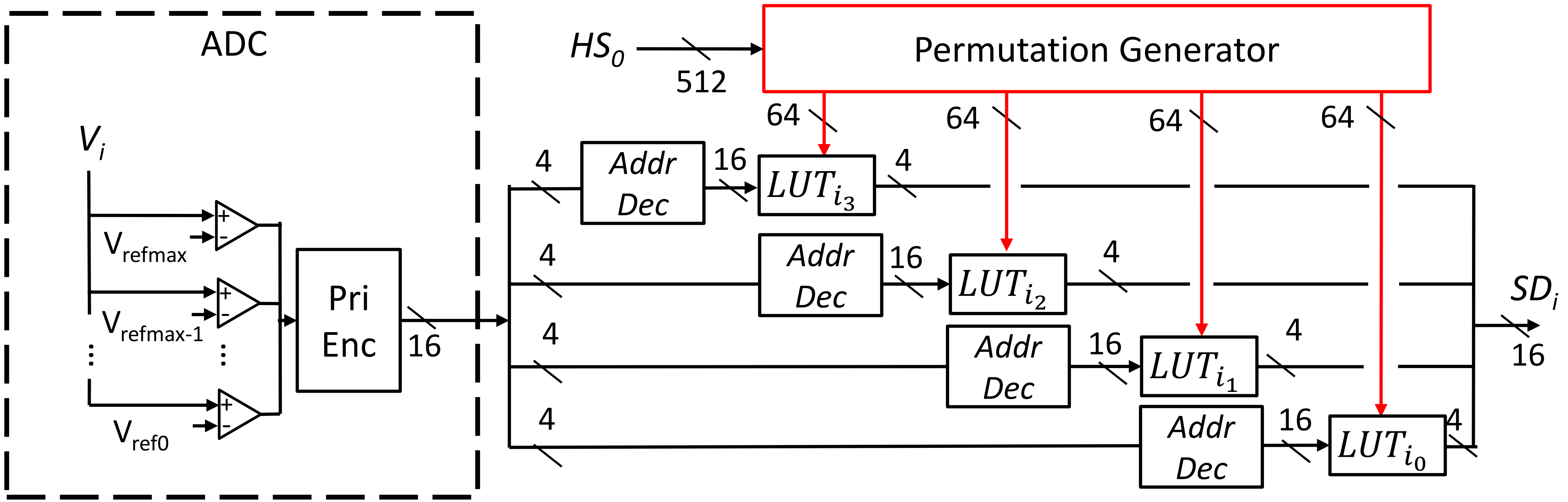
Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



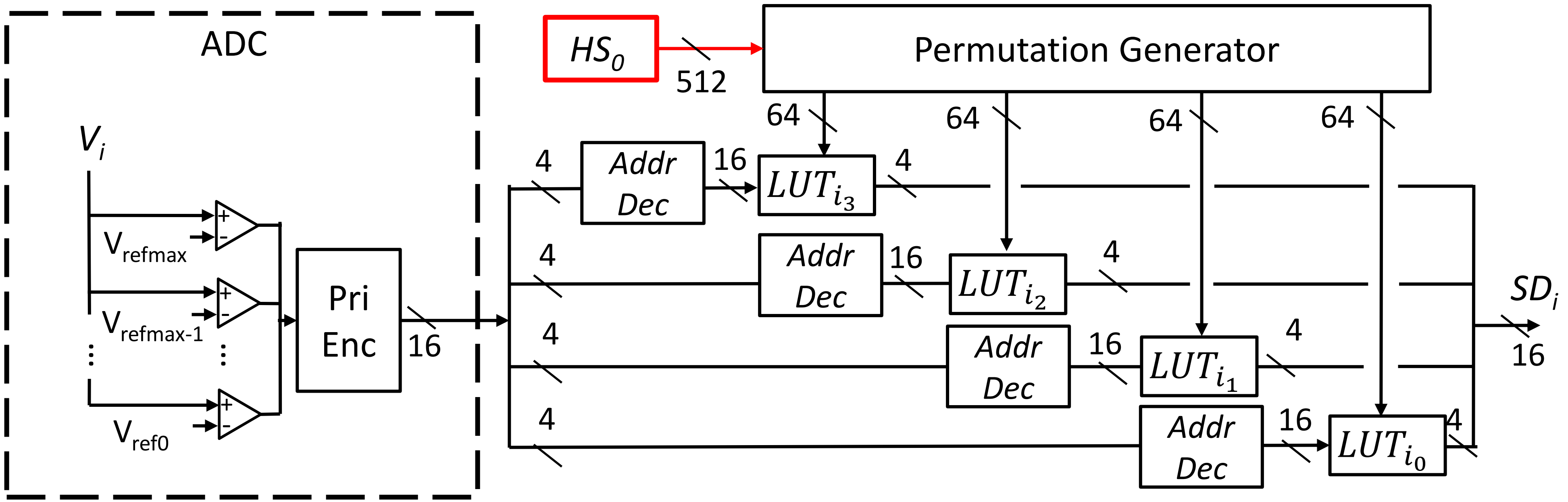
Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



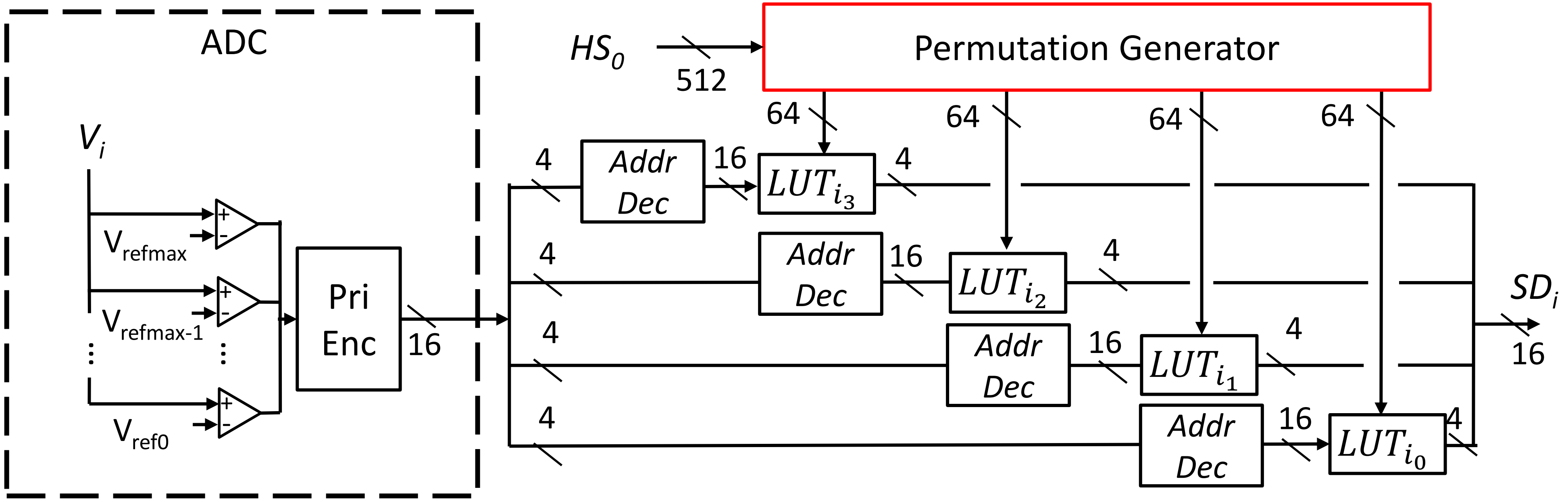
Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



Encoder Implementation

- Takes in an analog value and produces a 16-bit encoded output, without storing an unencoded version in any buffer memory
- Derived from prior work in [3]



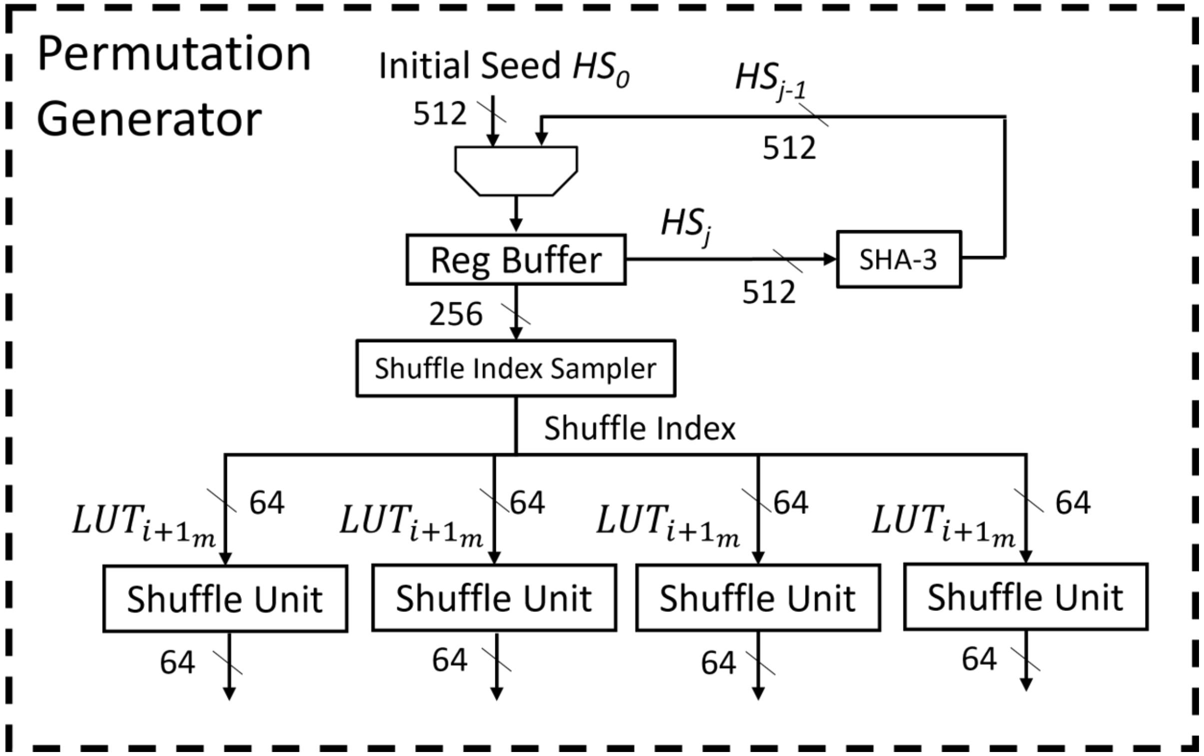
Permutation Generator

Consists of four “Shuffle Units” generating independent permutations on the set of 4-bit values

- Permutation generated via Knuth shuffle algorithm realized in hardware[4]

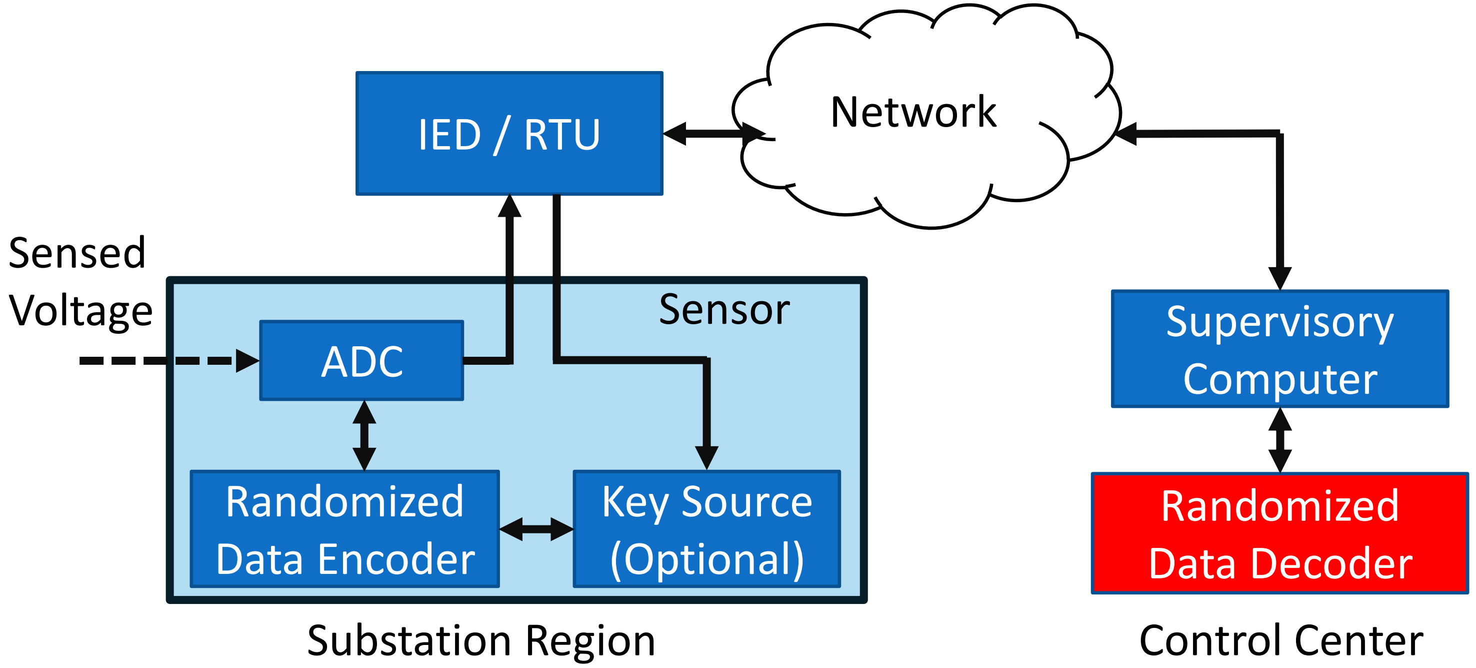
Each permutation is derived from an index provided by the output of a hardware SHA-3 module

The initial input to SHA-3, HS_0, acts as an ephemeral symmetric key



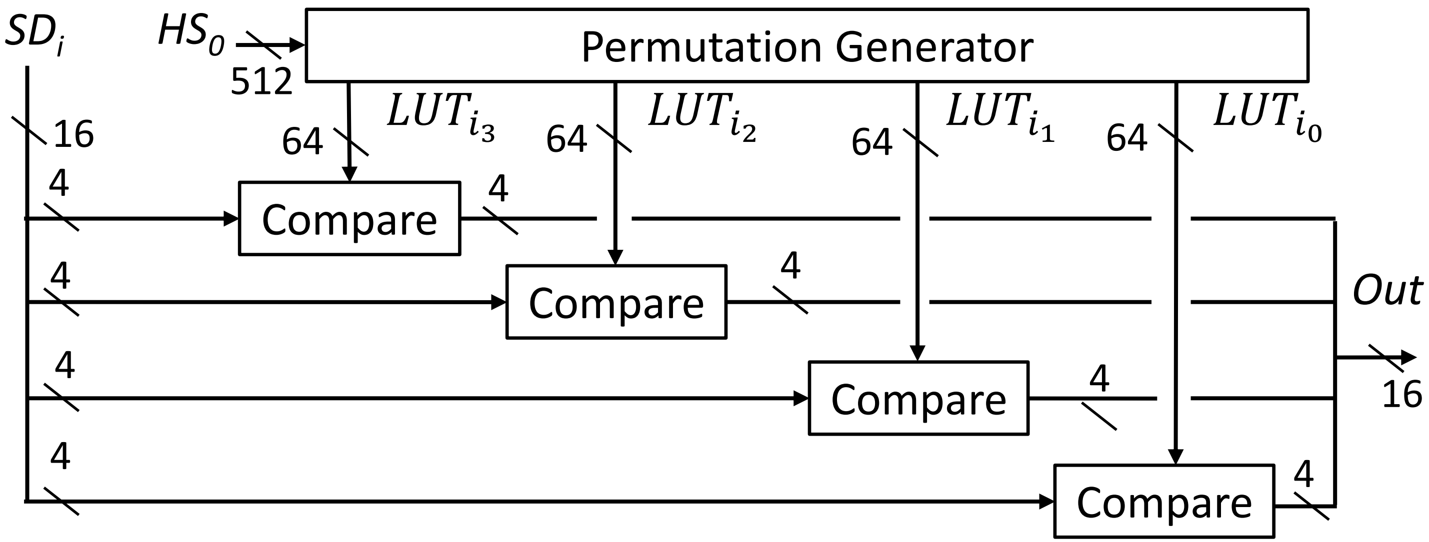
Target Architecture

- The control center contains a data decoder capable of interpreting the encoded data produced at the substation
- Control center must have the matching corresponding key, HS_0, used in the substation



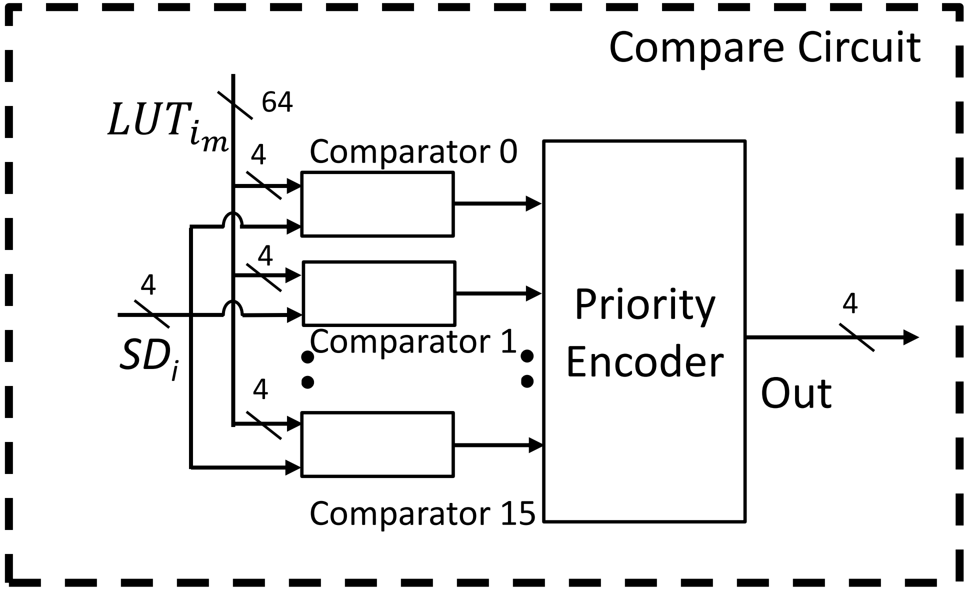
Decode Circuit

- Unencodes the randomized data
- Utilizes the same Permutation Generator as the encoding circuit
- Unencoded value retrieved via a comparison network



Decode Compare Circuit

- Four compare circuits (one for each LUT_{i_m}) inverse the encoding performed by the encoding circuit
- Each compare circuit operates in parallel



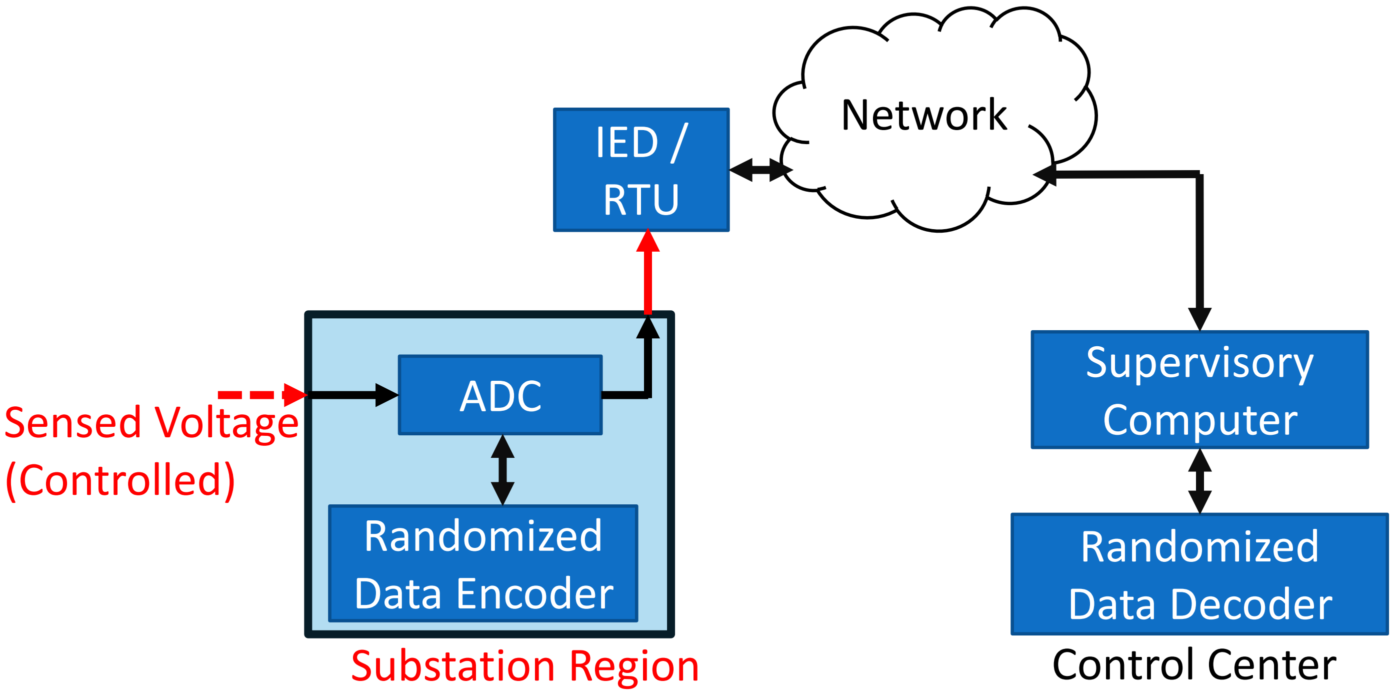
Contents

Security Analysis – Attack Model

- Adversary wishes to perform a false data injection attack [ref]
  - Requires believable (i.e., not random appearing) data after overwriting the original data, or the control center will reject the data as bad
- Adversary has one chance to inject false data before drawing attention
  - Assume continued erroneous data packets are investigated and the source discovered
- To guarantee intended malicious outputs, the adversary must know a stream of *future* encodings (the LUT_{i_m} mappings)
  - The adversary could alternatively learn the current internal hash value HS_j

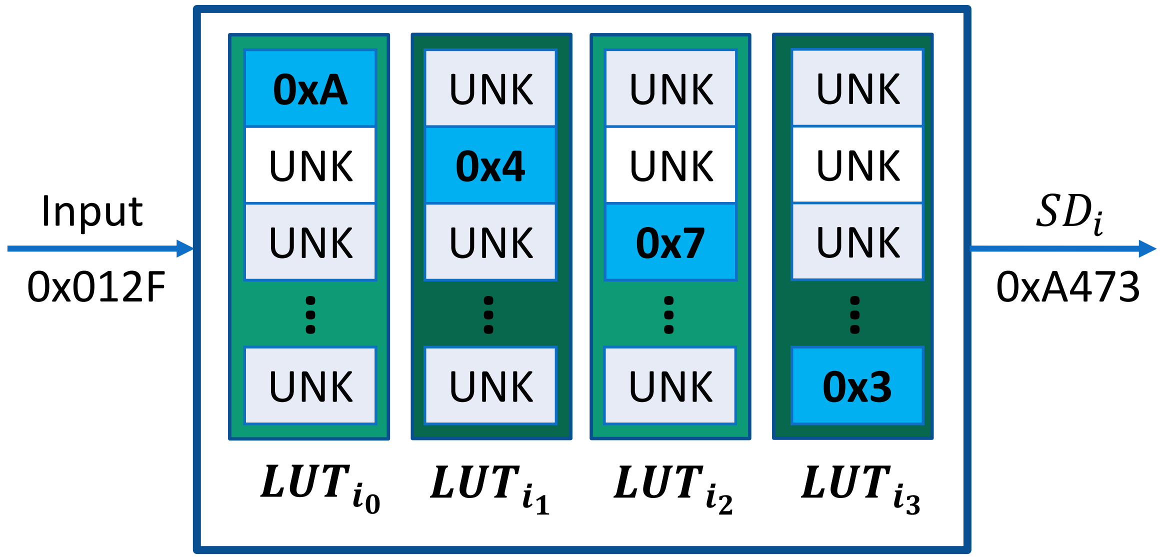
Security Analysis – Attack Model

- Adversary (i.e., a lone wolf insider) has access to stream of encoded outputs
- Adversary can precisely know a number of unencoded inputs (i.e., bulk power transformer temperature)
  - Possibly they monitor the target with an unencoded sensor
- Thus, adversary has a number of unencoded inputs to encoded outputs
- With the known input to output mapping subsets, the adversary attempts to determine future complete mappings to perform the false data injection attack



Attack Model Cont.

- With the discovered mapping 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?



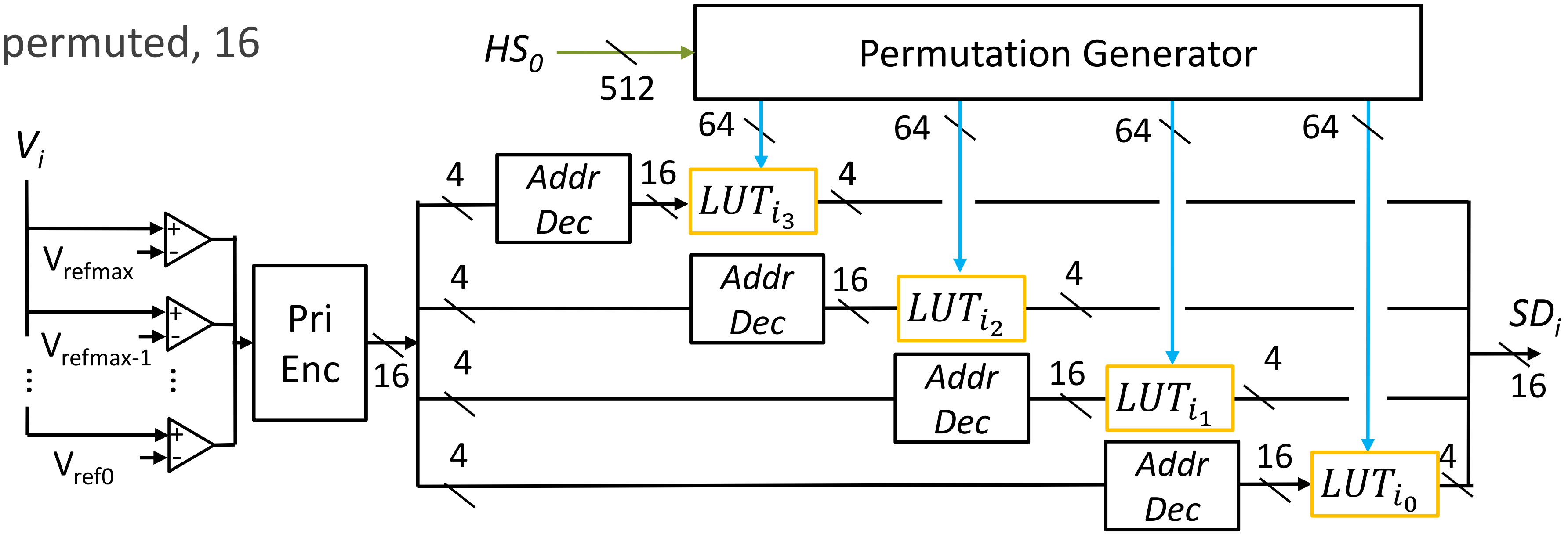
Security Analysis

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



Security Analysis

- Knuth Shuffle Algorithm is reversible
  - Given a known arrangement of set elements and a know 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

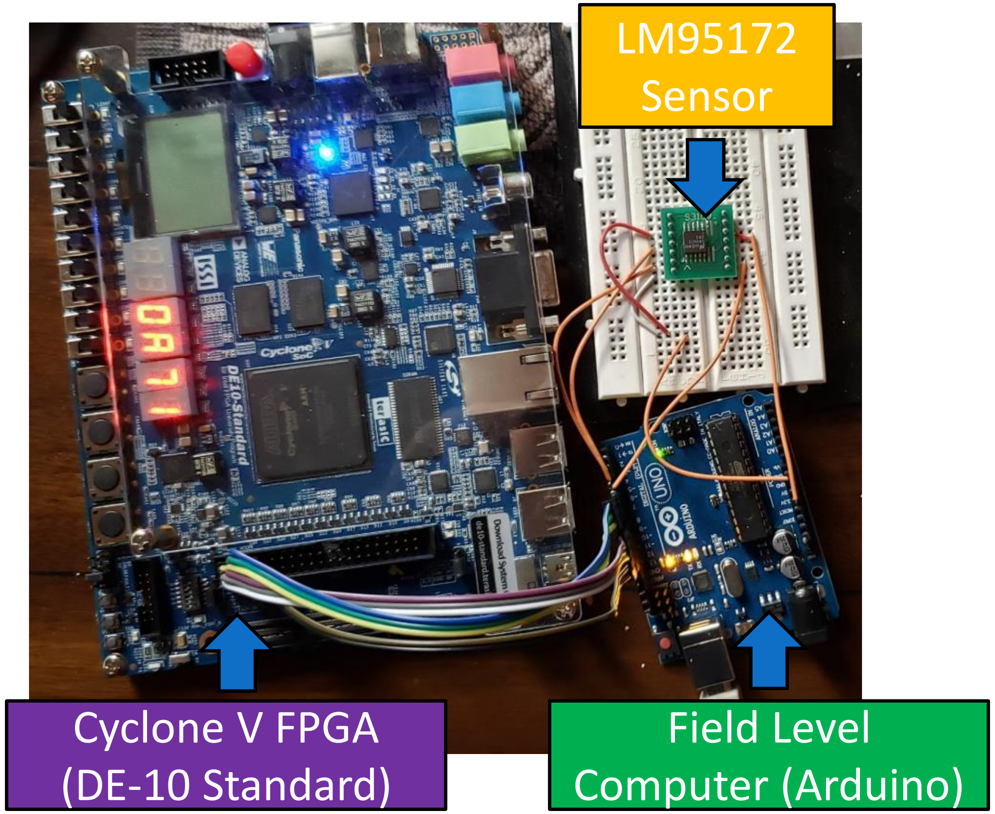
- 64 bits of the HS_j are used to select from the 16! permutations. Each permutation held in LUT_{i_m} has a possible 2^64 / 16! = ~2^16 corresponding indices
  - $I(k, m, l_s) = \frac{2^{\frac{l_s}{m}}}{k!}$
- The shuffle unit only uses half of the bits of HS_j, unused bits must be accounted for as they affect the follow-on values HS_{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 HS_j after pruning

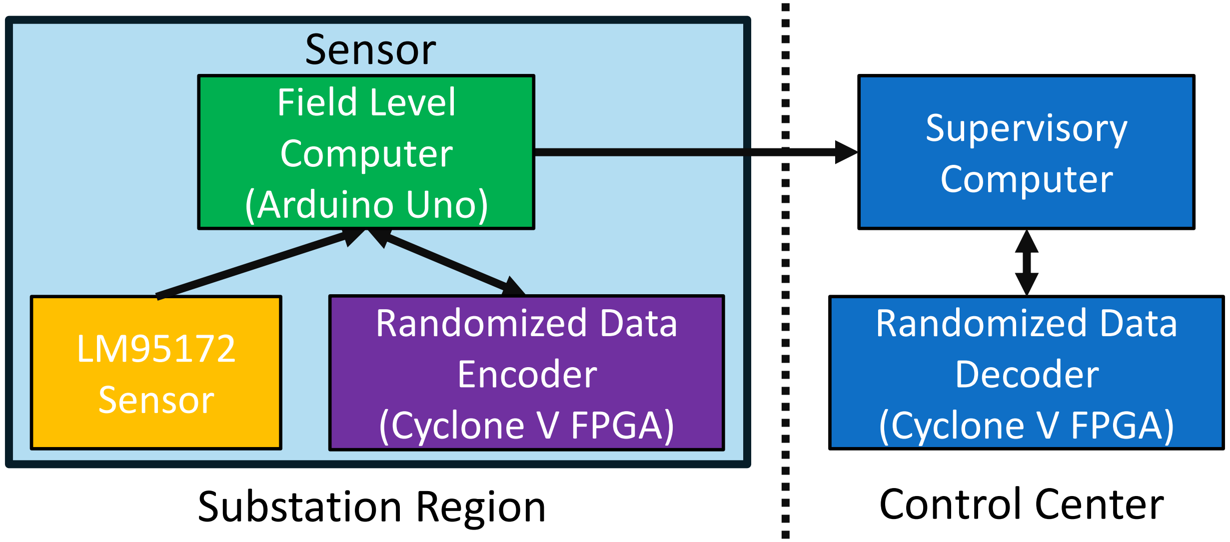
Security Analysis

- Each of the 2^{496} possible HS_j values will provide a permutation mapping to the known 4 locations in the 4 LUTs
- Only one of these possible HS_j values matches the actual internal value
- If the wrong HS_j value is used, the follow on values LUT_{i+1_m} will not match the actual Randomized Data Encoder mappings, and the central server will see bad data when decoding

Contents

Experimental Setup





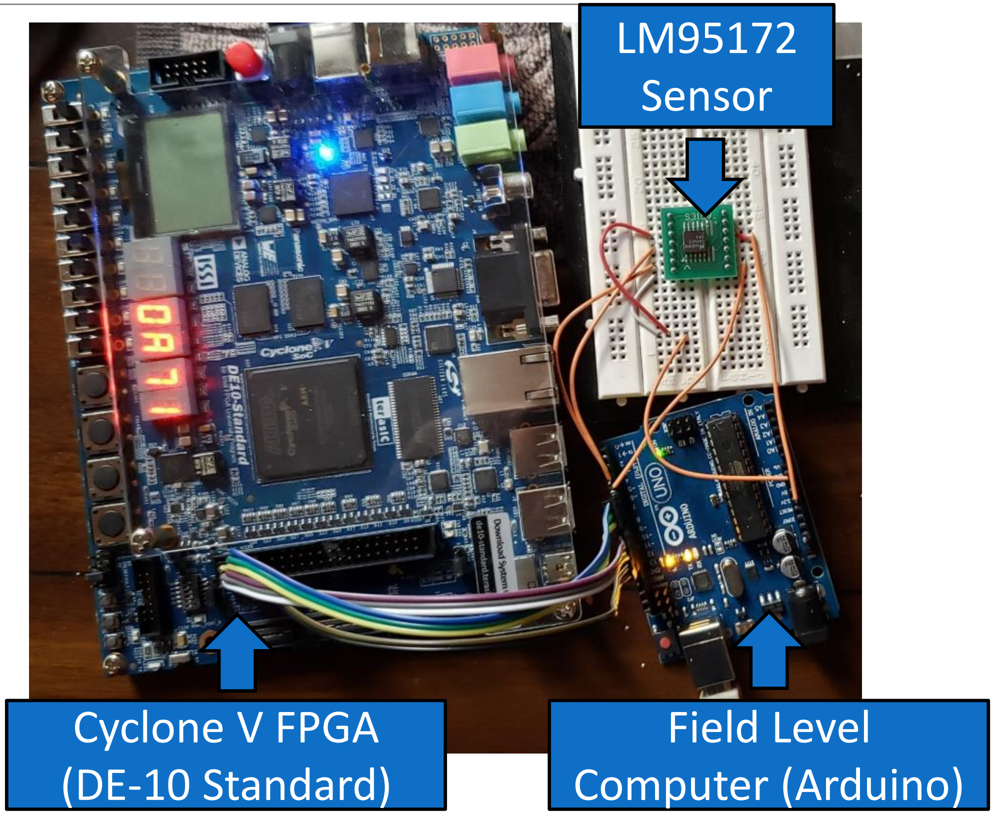
Experiments Conducted

Generated multiple data streams:

- 1. Temperature Sensor -> Encoder -> Decoder
- 2. Temperature Sensor -> Encoder
- 3. Temperature Sensor -> Decoder
- 4. Temperature Sensor -> Encoder -> Decoder (mismatched HS_0)

Performed χ^2 test on data streams

- Correct data indicated for data stream 1
- Significant bad data for data streams 2, 3, and 4



Synthesis Results

- Synthesis conducted targeting the Cyclone V 5CSXFC6D6F31C6 on the TerAsic DE-10 Standard Development Kit
- Utilized Quartus Prime 20.1.1
- Practical bottleneck in sampling rate was due to temperature sensor limitations

FPGA Utilization	Logic Blocks	Registers	DSP Blocks	Max Frequency (MHz)
Encoding Circuit	4044	3694	44	>180
Decoding Circuit	4088	3461	44	>200
Permutation Generator	1346	0	44	>200
Shuffle Unit	418	0	11	>200

References

[1] R. Deng, G. Xiao, R. Lu, H. Liang, and A. V. Vasilakos, “False data injection on state estimation in power systems—attacks, impacts, and defense: A survey,” IEEE transactions on industrial informatics, vol. 13, no. 2, pp. 411–423, 2017.

[2] R. Maes, Physically Unclonable Functions Constructions, Properties and Applications, 1st ed., 2013

[3] K. Hutto and V. Mooney III, “Sensing with random encoding for enhanced security in embedded systems,” Mediterranean Conference on Embedded Computing (MECO), vol. 10, pp. 809–814, 2021

[4] D. E. Knuth, The Art of Computer Programming. Volume 2, Seminumerical Algorithms, 3rd ed., 1997.

Thank You

Kevin Hutto

khutto30@gatech.edu

Santiago Grijalva

grijalva@ece.gatech.edu

Vincent Mooney

mooney@ece.gatech.edu

Georgia Tech School of Electrical and Computer Engineering