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

Kevin Hutto and Santiago Grijalva
*School of Electrical and Computer Engineering*
*Georgia Institute of Technology*
Atlanta, Georgia
khutto30@gatech.edu, sgrijalva@ece.gatech.edu

Vincent Mooney
*School of Electrical and Computer Engineering*
*School of Computer Science*
*Georgia Institute of Technology*
Atlanta, Georgia
mooney@ece.gatech.edu

***Abstract*—Supervisory Control and Data Acquisition (SCADA) systems are utilized extensively in critical power grid infrastructures. Modern SCADA systems have been proven to be susceptible to cyber-security attacks and require improved security primitives in order to prevent unwanted influence from an adversarial party. One section of weakness in the SCADA system is the integrity of field level sensors providing essential data for control decisions at a master station. In this paper we propose a lightweight hardware scheme providing inferred authentication for SCADA sensors by combining an analog to digital converter and a permutation generator as a single integrated circuit. Through this method we encode critical sensor data at the time of sensing, so that unencoded data is never stored in memory, increasing the difficulty of software attacks. We show through experimentation how our design stops both software and hardware false data injection attacks occurring at the field level of SCADA systems.**

***Index Terms*—Hardware Security, Power Grid, False Data Injection**

## I. Introduction

The power grid and supporting infrastructure contain a collection of complex distributed control systems. These systems consist mainly of supervisory control and data acquisition (SCADA) architectures [1]. In recent years numerous high profile cyber-attacks have occurred in various areas of the SCADA hierarchy, with some attacks causing blackouts. Many of these attacks were highly sophisticated, relying on vulnerabilities in multiple regions of the SCADA systems, such as the Ukraine power grid hack in 2015 [2].

Certain sections of the power grid may be more vulnerable than others. For instance, consider a simplified substation such as shown in Fig. 1. The substation is connected to a remote wind farm. The wind farm has numerous Intelligent Electronic Devices (IEDs) and Remote Terminal Units (RTUs) that receive critical sensor data from the power grid, including voltage, current, powers, temperatures, etc. However, the IEDs are less physically secure than traditional units located in a guarded nuclear power plant, for instance, presenting additional security challenges. By gaining physical access to the unmanned substation room, a capable adversary may be able to replace sensors or install malware to spoof sensor data vital to the overall supervisory control. The false data injection attack has been used in attacks such as the Stuxnet attack, which reprogrammed programmable logic controllers (PLCs) in order to mask centrifuge frequencies which exceeded their operating limits [3]. More generally, a false data injection attack can be used to influence real-time control power network applications such as state estimation or automatic generation control [4][5].

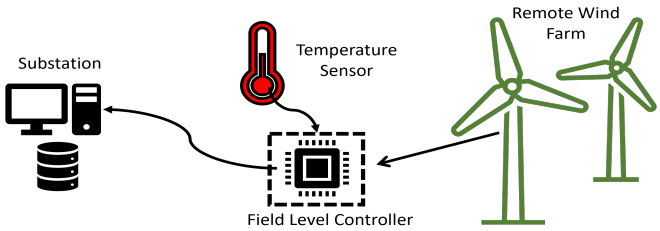


Fig. 1. Remote Substation Serviced from an Off-Shore Wind Farm

If an adversary wants to disrupt service from a sensor interfaced through an IED, an adversary could attack either through software or hardware. In a software attack, an adversary who has gained control of the software stack operating on an RTU or IED may be able to overwrite the sensor data in a way that enables a bad command or allows unwanted influence on power system operation. For a hardware attack, the adversary can instead replace the source of the data (the sensor itself) with a false source such as a function generator. Rather than overwriting the correct data, in this case the adversary directly provides a source of false information. We introduce a method to provide protection against both types of attacks.

## II. Prior Work

One of the earliest publications of the false data injection attack for SCADA systems was in [5]. The attack occurs when a malicious attacker modifies or injects data measurements such that state estimation techniques do not detect the false data as bad. Since the introduction of the attack, numerous techniques have been investigated to prevent a successful employment of the attack. The defenses developed against this attack deal mostly with new methods of detection of expected erroneous or bad data, using techniques such as advanced $\chi^2$ tests or through frequent authentication of the sensor [4][6][7].

In [8], a circuit was developed to implement a hardware permutation generator which continuously and randomly encodes values as they are output from an analog-to-digital converter (ADC). The encoded output values are enciphered and are

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.

978-1-6654-7902-8/22/$31.00 ©2022 IEEE

2022 IEEE Texas Power and Energy Conference (TPEC) | 978-1-6654-7902-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/TPEC54980.2022.9750706

then fed back through an XOR mechanism to provide new encodings for further values. These new (random) encodings are temporarily stored as a permutation on the set of possible bit values (e.g., 4-bit values). The outputs of the ADC are directly clocked into registers in an encoded format, with no version of the unencoded values ever existing in any memory storage on the encoding device. A physically separate device then performs decoding.

## III. Problem Statement and Attack Surface

We aim to minimize the attack surface available to an adversary attempting to spoof SCADA field level sensors. Specifically, we aim to design a sensor architecture that thwarts attempts by an adversary to deploy a software or hardware false data injection attack. We model an adversary who gains control of the software operating on a local device such as an IED. The adversary additionally has physical access to critical sensors, such as temperature sensors monitoring components of the generating units or transformers. We assume the adversary has no control over any software or hardware in the supervisory control or master station of the SCADA system. To protect against this adversary, we propose integrating a sensor with an encoding circuit to directly output encoded digital values at the time of sampling. As shown in Fig. 2, this could be accomplished with a public key cryptographic algorithm [9]. We aim to minimize and potentially even eliminate any unencoded data in buffer memory. Without access to unencoded data, false data injection attacks replacing such data become extraordinarily difficult if not impossible.

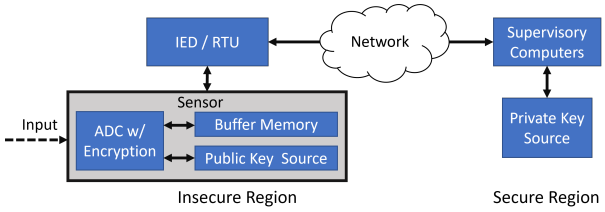


Fig. 2. Generic Model for an Integrated Sensor and Encryption Device

## IV. Design Methodology

In this section we showcase our design implementing Fig. 2. We then explain how the encoding and decoding components operate in the SCADA model.

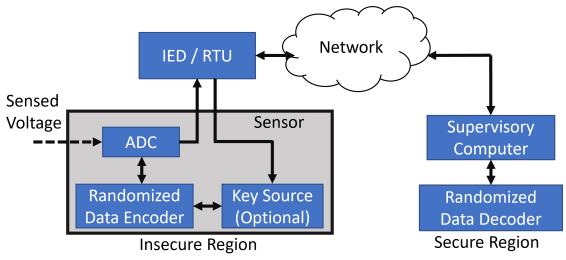


Fig. 3. Target Solution for Utilizing an Integrated Sensor and Encryption Module

### A. Overall Design

We integrated ADC and encryption in a sensor as a single composite circuit as shown in Fig. 3. The ADC is a standard flash ADC, which outputs to the Randomized Data Encoder (RDE). The RDE encodes the ADC output in accordance with a key, which is provided via the key source. The key source could be on chip from a Physically Unclonable Function (PUF) or could be provided via the Supervisory Computer located in the secure area via a True Random Number Generator (TRNG) [10] [11]. The sensor interfaces to an IED, which in turn communicates with the Supervisory Computer. The Supervisory Computer has access to the Randomized Data Decoder (RDD) to extract the unencoded sensed data.

### B. Randomized Data Encoder

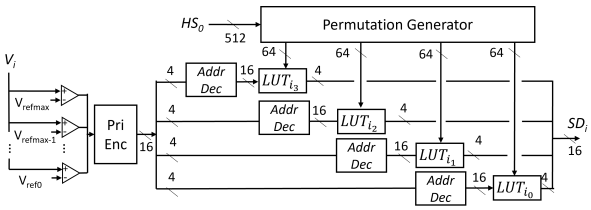


Fig. 4. Randomized Data Encoder and Analog to Digital Converter

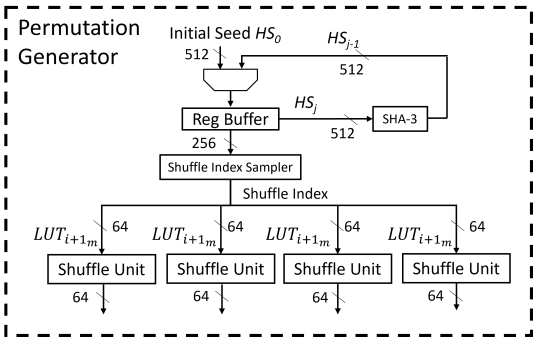


Fig. 5. Permutation Generator Implementing the Knuth Shuffle Algorithm

The Randomized Data Encoder from Fig. 3 is implemented as shown in Fig. 4. The circuit is derived from the circuit developed in [8]. The RDE encodes an analog voltage value $V_i$ ingested through an $n$-bit flash ADC. The encoding is conducted by having the analog sample value act as the select lines for multiple look-up tables labeled $LUT_{i_m}$, collectively $LUT_i$, where $i$ indicates sequential generations of the values in the look-up tables and $m$ designates the specific table ($\{m : 0 \leq m \leq 3\}$ in Fig. 4). Note that the select lines are never clocked in to any register or other digital storage element (i.e., the select lines only contain transitory pulses) [8][12]. $LUT_{i_m}$ holds a permutation of the set consisting of all 4-bit values (4-bit in Fig. 4). The four encodings are then concatenated together as a 16-bit value. As each 16-bit $SD_i$ sampled data value is clocked into a register, the permutations $LUT_{i_m}$ are updated via the Permutation Generator.

The Permutation Generator is shown in Fig. 5. The key component of the Permutation Generator is the Shuffle Unit. The Shuffle Unit produces new permutations by utilizing a hardware

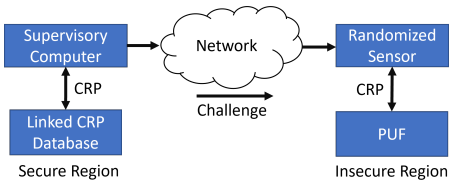


Fig. 6. New $HS_0$ Synchronization via an On-Chip PUF

implementation of the Knuth shuffle algorithm on four bits [13]. The Shuffle Index is used to choose which permutation is produced. There are four Shuffle Indices, which are 64-bit subsets of the 512-bit value *Hash Source* ($HS_j$). The 64-bit Shuffle Indices are selected via the Shuffle Index Sampler. $HS_j$ is produced by applying the 512-bit $HS_{j-1}$ as the input to a SHA-3 module [14]. The first $HS_j$, $HS_0$, comes from either a TRNG located within the secure region, or from an on-chip PUF. $HS_0$ is known as the initial seed. Either source of $HS_0$ allows the Supervisory Computer to have direct control over the choice of the new $HS_0$. The known initialization allows the secure Supervisory Computer in possession of the initial seed, $HS_0$, to compute all future values of the possible encoding permutations, $LUT_i$, and thus retrieve the unencoded outputs (given the encoded outputs have been properly received, e.g., over an encrypted channel). The RDE outputs a single encoded output value, $SD_i$, after each sample which is typically stored in a register or buffer memory.

### C. Key Source

A possible source for $HS_0$ is through the usage of a PUF. A PUF utilizes physical variations in device manufacturing to produce unique bitstrings, akin to fingerprints for the device. Depending on the design of the PUF, numerous different output bitstrings can be produced based on which "challenge" input bitstring is given to the PUF. The mapping of various challenges to "response" outputs creates challenge-response pairs (CRPs). A PUF is thus used as a key source by utilizing a response as the value $HS_0$, as shown in Fig. 6. The Supervisory Computer chooses a challenge from a database of previously recorded CRPs. The response is the new $HS_0$, and the supervisory computer will transmit the challenge to the sensor. The sensor will utilize the challenge to produce the new $HS_0$ from the on-chip PUF. By utilizing this method, the actual key value $HS_0$ is never transmitted in either plaintext or ciphertext format, and the risk of key compromise by an adversary is reduced [10].

### D. Randomized Data Decoder

The RDD circuit is shown in Fig. 7, which implements the component labeled "Randomized Data Decoder" in Fig. 3. This circuit decodes the $SD_i$ provided by the RDE circuit into the expected output of a traditional sensor. The RDD accomplishes the decoding with a circuit composed of largely similar components to the RDE circuit introduced in Fig. 4. The RDD utilizes the same Permutation Generator from Fig. 5 as the RDE circuit. With the RDD's $HS_0$ seed initialized to the same value as the RDE's $HS_0$, the RDD circuit produces an equivalent $LUT_{i_m}$ value sequence as the RDE circuit. The component labeled "Compare" in Fig. 7 is shown in detail in Fig. 8. The "Compare" block consists of a comparator for each 4-bit value in the $LUT_{i_m}$ mapping. For a given 4-bit input value, one of the sixteen comparators will output a logic '1'. This comparator output is used as the input to a priority encoder. Due to the arrangement of the priority encoder inputs, this structure recovers the unencoded version of the $SD_i$ input. Thus the original encoding is undone and the RDD circuit outputs the value which would have been obtained from a standard sensor.

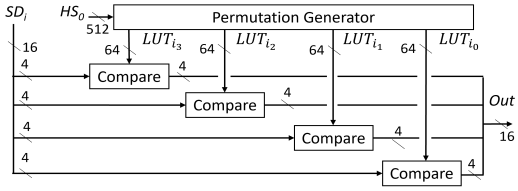


Fig. 7. Randomized Data Decoder

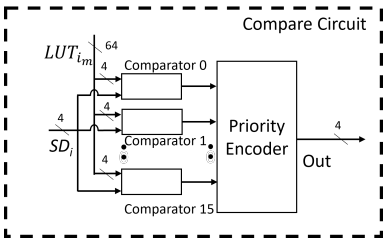


Fig. 8. RDD Circuit Compare Component

### E. Sensor Operation

We now will describe the high-level functionality of the RDE and RDD circuits. As shown in Fig. 3, the RDE and RDD circuits are interfaced through an IED, with the RDE residing in a region deemed insecure and the RDD in a region deemed more secure. Initially the RDE and RDD are synchronized with a common $HS_0$. The source of $HS_0$ can be stored on chip in a mechanism such as a PUF or delivered to the RDE via the supervisory control network external to the RDE, utilizing standard network security protocols for data transmission. During operation, each $HS_j$ provides independent and identically distributed encoding mappings for the $LUT_{i_m}$ values. After the encodings have been used to create an $SD_i$, the SHA-3 module is used to create $HS_{j+1}$. No $HS_j$ is used to create more than one of the $SD_i$ values. The produced $SD_i$s are sent to the interfaced IED, which then sends the the values to the supervisory control network for decoding. The supervisory control network decodes the $SD_i$s using an RDD circuit with the same $HS_j$ sequence as the RDE circuit. If required, the supervisory control network can resynchronize the RDE and RDD circuits by issuing a new $HS_0$. New $HS_0$ values will always be chosen by the supervisory control network, and $HS_0$ values will never be re-used.

This operation of the RDE and RDD circuitry provides inferred authentication of the sensor to the supervisory computer. If the sensor is replaced or the value $HS_0$ is altered, the supervisory computer will see data which appears random. The server may then assume the data is due to a malfunction in the sensor, or due to a adversarial attack. Either reaction will disregard the sensor's data.

## V. Security

In this section we explore how the developed RDE circuit provides inferred authentication and integrity in an environment with an attack surface as described in Section III.

### A. Approach to Show Security

We will show how an adversary fails to perform a false data injection attack. We start by providing our own red-team analysis as the most ideal attack scenario wherein an adversary has a super ability to temporarily manipulate the sensor environment (specifically, control the temperature exactly). We show that with sensor environment manipulation, the adversary has an exponentially small chance of successfully performing the false data injection attack. We follow by claiming an adversary without the ability to manipulate the sensor environment must also fail in any attack attempt.

### B. Attack Model

An adversary attempts to provide a false data stream from an IED or RTU that is seen as correct from the point of view of the supervisory levels of control in the SCADA system. We assume that an adversary has compromised portions of the network and can examine the entire stream of $SD_i$ outputs.

Now we will explain an attack attempt by an adversary. The goal of the attack is to reduce the search space required to determine the sequence of $HS_j$ values (and hence the encodings performed by $LUT_{i_m}$) on the device. Discovering these values will allow the adversary to forge false encoded data such that decoded values appear as correct values to the SCADA network.

For demonstration of security, we model a powerful adversary who can influence any parameter (i.e., the temperature of a bulk power transformer) for a limited number of samples such that the exact voltage encoded by the ADC is known. The adversary subsequently receives the $SD_i$ outputs via network intrusion. Due to the functionality of the RDE, the adversary receives one $SD_i$ value correlating to a single $HS_j$, and then $HS_j$ is overwritten by $HS_{j+1}$. The limited sequence of $SD_i$ values corresponding to known input voltages is the only information available to the adversary. $SD_i$ values from different $HS_j$ values are independent due to the functionality of the SHA-3 cryptographic hash function.

To test for the correctness of an assumed $HS_j$ from the untested candidate values for $HS_j$, the adversary randomly chooses a candidate *potential* $HS_j$ ($PHS_j$). For each $PHS_j$ value a stream of values $\{PHS_{j+1}, PHS_{j+2}, etc.\}$ is calculated. The adversary then uses a separate standalone RDE circuit to encode the known stream of input values using $\{PHS_j, PHS_{j+1}, ..., PHS_{j+n}\}$. If the stream of reproduced encoded outputs $\{SD_i, SD_{i+1}, ..., SD_{i+(2n+1)}\}$ produced from starting point $PHS_j$ match the actual $SD_i$ values output for some number of consecutive trials, then an adversary can reasonably assume $PHS_j$ is equal to $HS_j$. For $n$ $SD_i$ values tested, a sequence of encodings from an incorrect $PHS_j$ values will match the true $SD_i$ sequence with probability $(\frac{1}{16})^n$. We will now will explore the level of security the device has against the adversary who has successfully conducted this attack to determine the location of a single element in a series of $LUT_{i_m}$ values.

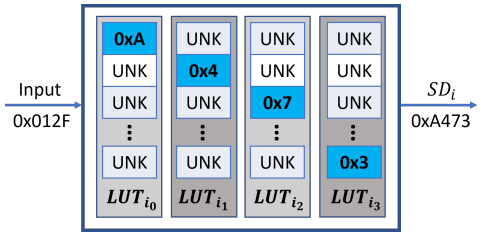


Fig. 9. $LUT_{i_m}$ mapping determination through input to output inspection.

In order to reduce the number of $PHS_j$ values required for testing, the adversary attempts to prune known incorrect values. The adversary has a collection of unencoded inputs correlating to encoded outputs. With this the adversary can determine a single address location for one element in the set of values stored in $LUT_{i_m}$. As an example, in Fig. 9, the adversary causes a voltage correlating to an input of 0x012F, and the current $LUTi_m$ mappings output 0xA473. The adversary now knows that any permutation which does not result in the mapping of 0x012F to 0x1473 is incorrect, without needing to test the sequence of $PHS_j$ values. With the limited $LUT_{i_m}$ permutation knowledge, the Knuth shuffle algorithm used to produce the permutation can be used in reverse to determine candidate $PHS_j$ values correlating to the {0, 1, 2, F} to {A, 4, 7, 3} mapping. The adversary will now test the chosen $PHS_j$ values against the output $SD_i$ values as previously described. The authors know of no method to determine more $LUT_{i_m}$ mapping values than this using our attack surface assumptions, which would allow the pruning of more $PHS_j$ values. In other words, one sixteenth of the 256-bit $PHS_j$ input to the shuffle units can be determined without explicit testing, but no more than this.

### C. Level of Security

In Section V-B we introduced a powerful attack by an adversary to reduce the search space required for determining the in-use $HS_j$. Now we will identify how much this attack has reduced the search space from a full brute force search of 512 bits. As discussed in Section IV-B, the addresses of the elements in $LUT_{i_m}$ are determined by the Knuth shuffle algorithm. The algorithm is deterministic and can be modeled as taking an index as input and outputting a specific permutation. The algorithm allows all possible permutations of a set. For a given set with $k$ elements, there are $k!$ permutations. With knowledge of the address of one element, as found in Section V-B, there are $k-1$ unknown element locations and $(k-1)!$ possible permutations for the remaining unknown element locations. Each possible permutation has a unique shuffle index subset associated with it. With four $LUT_{i_m}$ components in the architecture shown in Fig. 4, a single unencoded input value to encoded $SD_i$ mapping contains four unknown $LUT_{i_m}$ permutations. More generally, with $m$ $LUT_{i_m}$s holding permutations on sets of $k$ elements, the total permutations possible, and thus the total number of shuffle indices possible for a given encoding subset determined by the adversary's attack is given by the following:

$$P(k, m) = ((k-1)!)^m \tag{1}$$

Each possible $LUT_{i_m}$ encoding correlates to multiple possible shuffle index subsets, however. Thus far we have made the assumption that each permutation correlates to a single $HS_j$ subset. In our design, though, 64 bits of the $HS_j$ are used to select from the $16! \approx 2^{48}$ permutations. Each permutation held in $LUT_{i_m}$ has a possible $\frac{2^{64}}{\sim 2^{48}} \approx 2^{16}$ corresponding indices. Referring to the 256-bit shuffle index as $l_s$, the number of indices per permutation is given by:

$$I(k, m, l_s) = \frac{2^{\frac{l_s}{m}}}{k!} \tag{2}$$

Hypothetical pruning techniques do not apply for disregarding similar indices for a given permutation (i.e., assume a technique exists to allow ignoring the additional possible indices from Equation 2). This is because two identical permutations held in $LUT_{i_m}$ with different indices will have different future values $LUT_{i+1_m}$ due to the properties of the SHA-3 cryptographic hash function.

Combining Equations 1 and 2 we obtain the following:

$$P(k, m, l_s) = (\frac{2^{\frac{l_s}{m}}}{k!}((k-1)!))^m \tag{3}$$

The structure of the RDE utilizes only half of the $HS_j$ values for each sample. The unused half does not affect the current encoding, but will cause different future values $\{PHS_j, PHS_{j+1}, ..., PHS_{j+n}\}$ and must be accounted for. With the length of $HS_j$ as $l_h$ we obtain the following overall equation:

$$P(k, m, l_s, l_h) = (\frac{2^{\frac{l_s}{m}}}{k!}((k-1)!))^m \times 2^{(l_h - l_s)} \tag{4}$$

With the shown RDE circuit, we encode four 4-bit values at a time, giving a value for $k$ of $2^4 = 16$ and $m = 4$. We additionally have the values $l_h = 512$ and $l_s = 256$. Utilizing these values we obtain the total number of possible values of $HS_j$ for a given sequence of two $SD_i$ outputs as follows:

$$P(k, m, l_s, l_h) = (\frac{2^{\frac{256}{4}}}{16!}((16-1)!))^4 \times 2^{(512-256)} = 2^{496} \tag{5}$$

The number of possible $HS_j$ values for a known specific voltage to $SD_i$ encoding is exactly $2^{496}$ with our RDE circuit parameters. This large search space for a brute force attempt to determine which $HS_j$ sequence matches to known encodings is beyond an adversary's capabilities. This search space can be confirmed by alternatively noting that the attack to determine encodings has removed permutation uncertainty from $\frac{1}{16}$ of the index, for half of the total length of $HS_j$. With a 512-bit $HS_j$, removing 1/16 of the bits from half of the total length results in 496 bits of uncertainty ($\frac{512}{16*2} = 16$, and $512 - 16 = 496$.) This provides a level of security harder than determining a collision for a SHA-3 512-bit hash output and easier than determining a pre-image.

Due to the properties of the cryptographic hash function used, SHA-3, there are no known correlations between the hash input and hash output. The authors know of no method to carry forward the discovered 32 bits from one generation of $HS_j$ to the next $HS_{j+1}$. The discovery process for determining any $HS_j$ appears independent of the discovery process for future values. Only with a fully correct $HS_j$ is any knowledge gained of $HS_{j+1}$, $HS_{j+2}$, ..., $HS_{j+n}$. We believe the circuit provides a level of security at least as strong as SHA-3 [14].

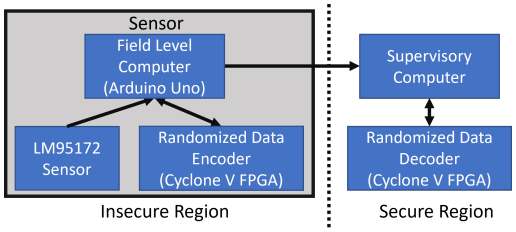


Fig. 10. Tested RDE Implementation

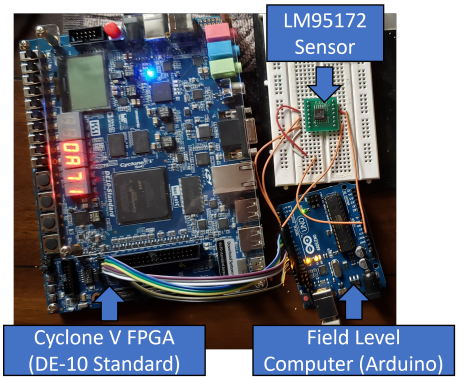


Fig. 11. Physical RDE Implementation

## VI. Test Methodology

For our tests we have utilized a temperature sensor as the target application. We emphasize that the proposed technique is valid for any type of analog measurement, such as voltages, currents, or power measurements derived from power instrument transformers. Both the encoding and decoding circuits are implemented via VHDL on separate FPGAs. The implemented test architecture is shown in Fig. 10 and Fig. 11. The tests utilize the LM95172 commercial temperature sensor and the Intel Cyclone V SoC [15][16]. For this proof of concept the LM95172 transmits the 16-bit temperature recording to the Cyclone V via SPI for encoding. To simulate a software or hardware replacement of the sensor data, we bypass the RDE circuit and pass unencoded data through the RDD circuit.

For simulation, ModelSim - Intel FPGA Edition 10.5b revision 2016.10 was utilized. For synthesis a solution targeted to the Cyclone V 5CSXFC6D6F31C6 on the TerAsic DE-10 Standard Development Kit was developed. All synthesis was conducted in Quartus Prime 20.1.1 Build 720. We utilized the mid-range SHA-3 core module provided by [17].

### A. Operating and Synthesis Results

The RDE circuit was run ten times on 20,000 data samples with different $HS_0$ starting values, for a total of 200,000 samples. The temperatures measured were ambient room temperatures, and the temperature was changed for each test. Each of the produced $SD_i$ samples were then decoded through a synchronized RDD circuit. The RDD circuit maintained synchronization such that all outputs decoded to the correct plaintext values. The plaintext outputs were verified by comparison to unencoded temperature recordings obtained in parallel with the

$SD_i$ values. We additionally verified the RDD output showed no suspected bad data via a $\chi^2$ test [6]. The $\chi^2$ tests were conducted with a confidence level $P$ of 0.01.

To simulate a software or hardware attack, the unencoded temperatures recorded in parallel were passed through the RDD circuit. This RDD operation decoded data which was never encoded originally, simulating a replacement of the sensor by a device with no encoding functionality. An additional $\chi^2$ test was performed on this decoded output with extensive bad data resulting from bypassing the RDE circuit. A further attack was simulated by attempting to decode one set of 20,000 $SD_i$s with an RDD circuit set to the $HS_0$ value used for a test at a different background temperature, simulating an attempt to replace the sensor with a guessed $HS_0$ value. The decodings obtained from this test produced extensive bad data from a $\chi^2$ test as expected.

FPGA resource utilization metrics and throughput are shown in Table I. We provide the utilization results for both the RDE and RDD circuits, as well as for the permutation generator and shuffle unit sub-components. The circuit is implemented on the low-cost Cyclone V FPGA, and larger FPGAs could easily run multiple instances of the RDD circuit in parallel. The majority of the logic and register utilization for both the RDE and RDD circuit is due to the SHA-3 module, though DSP blocks are used by both the RDE and RDD circuit as part of the permutation generator [13]. In actual testing throughput limitations were due to the limits in rate of conversion in the commercial temperature sensor itself [15].

TABLE I
FPGA UTILIZATION FOR THE RDE AND RDD CIRCUIT

| Circuit | (A) | (B) | (C) | (D) |
|---|---|---|---|---|
| RDE Circuit | 4044 | 3694 | 44 | >180 |
| RDD Circuit | 4088 | 3461 | 44 | >200 |
| Permutation Gen. | 1346 | 0 | 44 | >200 |
| Single Shuffle Unit | 418 | 0 | 11 | >200 |

TABLE II
(A) LOGIC (ALM) BLOCKS. (B) REGISTERS. (C) DSP BLOCKS. (D) MAXIMUM FREQUENCY (MHZ).

## VII. FUTURE WORK

The sensor developed in this paper relies upon a flash ADC style architecture. Future work would explore other ADC styles such as a $\Delta\Sigma$ ADC or successive approximation ADC. Future work additionally exists in integration with a PUF and deployment with power IEDs in a real SCADA system.

## VIII. SUMMARY

In this paper we have presented an architecture integrating an ADC with a deterministic hardware encoding method to provide enhanced security against false data injection attacks in SCADA systems. The lightweight method provides security more difficult than determining a collision for a SHA-3 512-bit hash. By providing inferred authentication via a public-key style of encryption, the RDE and RDD circuits can provide enhanced protection against both hardware and software false data injection style attacks. This additional security is particularly beneficial in vulnerable remote sensors which are parts of critical infrastructure. With continuing cyber-attacks occurring in critical sectors such as the power grid, it is vital to continue research into preventing potential attacks. The proposed method can provide enhanced security to power grid devices through hardware encoding.

## REFERENCES


[1] M. S. Thomas and J. D. McDonald, *Power system SCADA and smart grids*, 2015.

[2] G. Liang, S. R. Weller, J. Zhao, F. Luo, and Z. Y. Dong, "The 2015 Ukraine blackout: Implications for false data injection attacks," *IEEE transactions on power systems*, vol. 32, no. 4, pp. 3317–3318, 2017.

[3] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," *Survival*, vol. 53, no. 1, pp. 23–40, 2011. [Online]. Available: https://doi.org/10.1080/00396338.2011.555586

[4] 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.

[5] Y. Liu, P. Ning, and M. Reiter, "False data injection attacks against state estimation in electric power grids," in *Proceedings of the 16th ACM conference on computer and communications security*, ser. CCS '09. ACM, 2009, pp. 21–32.

[6] R. A. Donnelly, *Statistics*, third edition, first american ed., 2016.

[7] A. O. Gomez Rivera, E. M. White, and D. K. Tosh, "Robust authentication and data flow integrity for p2p scada infrastructures," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 557–564.

[8] 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.

[9] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC Press/Taylor & Francis, 2015.

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

[11] M. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Nist special publication 800-90b: Recommendation for the entropy sources used for random bit generation," *US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA*, 2018.

[12] A. S. Sedra, K. Smith, and A. Chandorkar, "Microelectronic circuits: Theory and applications," pp. 1014–1017, 2013.

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

[14] M. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015-08-04 2015.

[15] National Semiconductor, "13-bit to 16-bit 200°c digital temp sensor with 3-wire interface," LM95172 Datasheet, December 16, 2009.

[16] Intel, "Cyclone v device datasheet," CV-51002, November 27, 2019.

[17] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, "Keccak in vhdl," Team Keccak, 01 2021. [Online]. Available: https://keccak.team/hardware.html