A Lightweight Cryptographic Permutation Generator for Critical Infrastructure Protection

Arman Allahverdi*, Kareem Ahmad*, Vincent John Mooney III*†, and Santiago Grijalva*
*School of Electrical and Computer Engineering
†School of Computer Science
Georgia Institute of Technology
Atlanta, United States of America
{aallahverdi3, kahmad8, mooney, sgrijalva6}@gatech.edu

***Abstract*—Critical infrastructure, such as the power grid, often relies on supervisory control and data acquisition devices, including relays, remote terminal units, and field-level sensors whose data and outputs directly influence automated control decisions. These devices typically expose raw measurements to memory or data buses whose contents can be read or modified by an adversary, enabling data manipulation at the edge level. Furthermore, these devices often include stringent resource and computing power constraints, necessitating the use of lightweight security primitives for their protection. This paper presents a lightweight rendition of a data encoding architecture applicable to data encoding and image difference applications. Our architecture achieves significant hardware savings and performance improvements by adding a lightweight feedback register-based pseudorandom number generator to the encoding architecture in place of the SHA-3 hash function. To demonstrate the use of our architecture in critical infrastructure settings, we perform experiments involving the encoding and randomness testing of data from a temperature sensor, along with taking an image difference of a sensor device being monitored by a camera. When implemented on an FPGA and compared to the version of the encoding scheme using SHA-3, our design reduces lookup table utilization by 63% and register utilization by 86%, lowers encoding latency from 830 clock cycles to 384 cycles for the first sample and 256 cycles thereafter, and allows for performing image difference at a rate of 27 frames per second in the encoded domain compared to prior results of 15 frames per second on the same FPGA.**

***Index Terms*—data randomization, encoding, PRNG, SCADA security, critical infrastructure**

I. Introduction

Critical infrastructure increasingly depends on distributed sensing and actuation, particularly in power systems, where supervisory control and data acquisition (SCADA) measurements provided by devices and sensors in the field inform operational decisions in real time. The attack surface at the field edge remains stubbornly large: commodity microcontrollers, exposed data buses, and unprotected sensors hold or transport plaintext data that can potentially be read or manipulated by an adversary, enabling attack vectors on critical infrastructure. Recent work addresses this problem by encoding data at acquisition time and enabling computation directly in the encoded domain via pseudorandom, per-sample permutation lookup tables (LUTs) that obscure plaintext data by implementing a *homomorphism* [6], [7], [14]. Original plaintext data values are rendered unavailable in the memory of the device performing the encoding, remaining obscured until received by a decoding device.

This paper introduces a lightweight cryptographic permutation generator architecture for critical infrastructure data encoding and image difference applications based on a hardware-efficient pseudorandom number generator (PRNG). We validate the proposed lightweight encoding architecture in two application settings central to critical infrastructure and power grid security: (i) field-level sensor data encoding and randomness testing using a temperature sensor and (ii) image difference computation performed entirely in the encoded domain. Moreover, we evaluate the hardware footprint of the proposed architecture on a field programmable gate array (FPGA) board, comparing the FPGA utilization of our proposed architecture to prior work, demonstrating substantial reductions in LUT and register utilization. Finally, we demonstrate reductions in encoding latency and increases in the frames per second (FPS) that can be processed for image difference applications, compared to the prior work.

II. Background and Prior Work

A. RanCode and RanCompute

RanCode, introduced in [14], is a homomorphic encoding architecture that uses pseudorandom LUTs to digitally encode analog data, enabling encoded digital data to be transmitted and computed on downstream without exposing the underlying plaintext data. RanCompute [12], [14] is a companion architecture to RanCode that enables computation to be carried out entirely in the encoded domain, using LUT–based permutations to ensure that all intermediate values and operations remain obscured while still producing a decodable final result, meaning that RanCompute implements a homomorphism. A key advantage of RanCompute is its ability to achieve real-time throughput compared to prior approaches in homomorphic encryption [17]–[19]. When discussing concepts that apply to

This work has been partially supported by the U.S. Department of Energy (DoE) Office of Cybersecurity, Energy Security, and Emergency Response (CESER) under Cybersecurity for Energy Delivery Systems (CEDS) Agreement Number #DE-CR0000055 to the Georgia Tech Research Corporation: GRIDLOGIC: Hardware/Software Codesign for Deep Grid Visibility and Security

979-8-3315-5720-1/26/$31.00 ©2026 IEEE

both RanCode and its companion architecture RanCompute, we will use the shorthand phrase *RanCode/RanCompute*.

RanCode has been applied to power grid security to provide a hardware-based solution for securing and authenticating SCADA sensors [7]. RanCode/RanCompute requires a cryptographic PRNG to implement its permutation generator, the output of which governs the construction of the permutation LUTs that determine how data values are to be encoded. The SHAKE256 extendable output function (XOF) used in the SHA-3 hashing algorithm [8] has previously been used as the RanCode PRNG.

B. Feedback and Product Registers

1) Feedback Registers: Feedback registers are finite-state systems whose state evolves in discrete time steps according to a fixed update function f, where f is a function of the current state. Feedback registers are commonly implemented as linear feedback shift registers (LFSRs), where the feedback function f is a linear Boolean function, or nonlinear feedback shift registers (NLFSRs), where f is a nonlinear Boolean function. In the case of a Galois LFSR [1], the feedback logic is distributed in between the flip-flops of the LFSR.

For both LFSRs and NLFSRs, the all-zeros state is an *absorbing state*; an LFSR or NLFSR initialized to the all-zeros state will not end up in a different state, irrespective of how many times the register is clocked. An n-bit LFSR or NLFSR is considered *full-period* if, for any n-bit initial state, the register goes through $2^n - 1$ distinct n-bit states before returning to its initial state. To instantiate a full-period LFSR, the feedback polynomial $P(x)$ must be primitive, meaning $P(x)$ is both irreducible (non-factorable) and evenly divides $x^{2^n-1} + 1$ [2]. Full-period NLFSRs, however, are typically tested for via a brute force search of the state space in the case of small register sizes or constructed using methodologies that only allow for limited nonlinearity [5].

2) Mersenne Product Registers: Mersenne Product Registers (MPRs), introduced in [2], are a generalization of Galois LFSRs. An n-bit MPR is defined by specifying an update polynomial $U(x)$ of degree at most $n - 1$ and a feedback polynomial $P(x)$ of degree n.

3) Chaining and Composite Mersenne Product Registers: Chaining is a method for composing several individual Mersenne Product Registers into a larger interconnected register structure referred to as a Composite Mersenne Product Register (CMPR), where the constituent MPRs are connected using *chaining functions* [2]. Chaining functions are sets of Boolean equations that govern how the state of one MPR affects the state update of another MPR. Chaining functions can be made nonlinear by including multiplicative terms in the Boolean equations (e.g., AND gates), making CMPRs an appealing building block for cryptographic schemes [2].

C. The Hybrid Register Stream Cipher

The hybrid register stream cipher, a lightweight synchronous stream cipher proposed and analyzed in [3], combines NLFSR and CMPR theory to instantiate a *hybrid register* used to generate pseudorandom numbers as part of a stream cipher implementation for authenticated encryption. The hybrid register itself is a PRNG based on a 256-bit nonlinear feedback register formed by the interconnection of a 128-bit NLFSR and 128-bit CMPR. It is designed to take as input a 128-bit secret key, along with a 96-bit initialization vector (IV), a parameter that allows for generating distinct encryptions in situations where the same plaintext is encrypted with the same key. The hybrid register is capable of generating up to 2^{81} pseudorandom bits from a single seed and scales to support higher security levels, along with outperforming lightweight stream ciphers like Grain-128AEADv2 in terms of hardware area, power, and latency [3].

III. Attack Surface and System Architecture

A. Attack Surface

The attack surface for this work assumes that an attacker with access to a field device that implements RanCode/RanCompute can read on-chip memory, but does not have access to microarchitectural registers that are normally inaccessible to software. An attacker can also physically capture a field device and perform reverse engineering techniques such as decapsulation and delayering. However, it is assumed that any encrypted communications between a RanCode/RanCompute-capable field device and its server remain uncompromised, meaning the feasible attack vectors are primarily physical and local.

B. Original Encoding and Decoding Architectures

The original RanCode encoding architecture [7] is shown in Figure 1. In the architecture of Figure 1, an analog-to-digital converter (ADC) and sensor value encoding are implemented as a single composite circuit. The ADC outputs 16-bit digitized data values to the randomized data encoder. The randomized data encoder encodes the digitized values using LUTs generated from an initial 512-bit key, HS_0, provided via a secure key source. In practice, a key source could be on-chip from a Physically Unclonable Function (PUF) [12] or could be provided by a trusted supervisory computer in a secure location. The randomized data encoder outputs a digitized 16-bit encoded output value, SD_i, for each analog sample. The SD_i values are typically stored in buffer memory or a register. A trusted supervisory computer would also have access to the randomized data decoder circuit of Figure 2, which intakes the SD_i generated by the randomized data encoder and returns the original pre-encoding digitized data (i.e., plaintext).

The permutation generator architecture is shown in Figure 3. RanCode/RanCompute requires a PRNG as part of the permutation generator to continuously produce new, unpredictable permutation keys (the HS_j in Figure 3) that regenerate the LUTs used to encode each digitized 16-bit value. From Figure 3, SHA-3 [8] has previously been used as the RanCode PRNG.

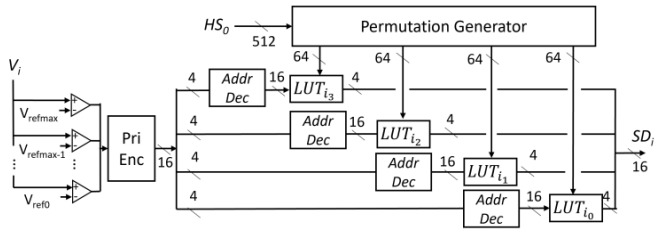

Fig. 1. Randomized Data Encoder Architecture

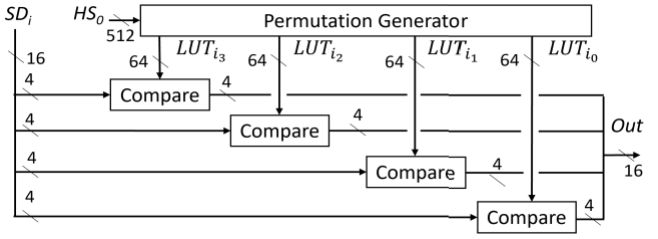

Fig. 2. Randomized Data Decoder Architecture

C. Proposed PRNG and Permutation Generator Architecture

The hybrid register stream cipher defined in [3] takes as inputs a 128-bit key and a 96-bit initialization vector (IV), where varying the 96-bit IV provides unique ciphertexts when the same plaintext message is encrypted using the same key. The RanCode architecture, however, only requires that the PRNG component of the design intake a seed. The output size of the PRNG component must be at least 256 bits in order to generate the four LUTs of Figure 1, each of which requires a 64-bit pseudorandom value.

In the RanCode/RanCompute architecture of Section II-A, the PRNG output was 512 bits: 256 bits were used to generate LUTs, and the entire 512-bit output was used to re-seed SHA-3 in preparation for the next encoding. In this work, we modify the original iterative permutation generator architecture of Figure 3. Since the hybrid register can generate a stream of up to 2^{81} pseudorandom bits (approximately 3×10^8 petabytes)

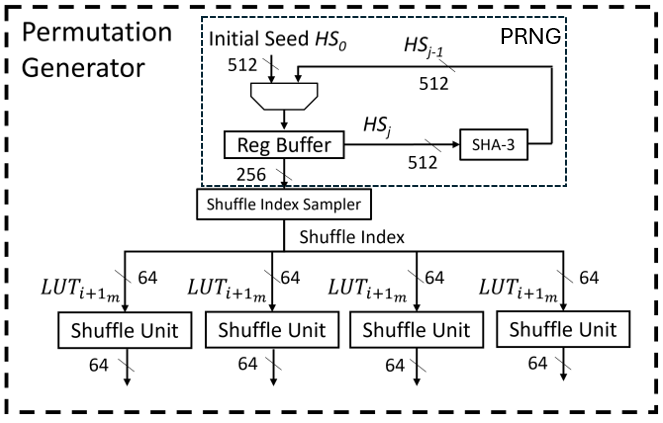

Fig. 3. Original Permutation Generator Architecture

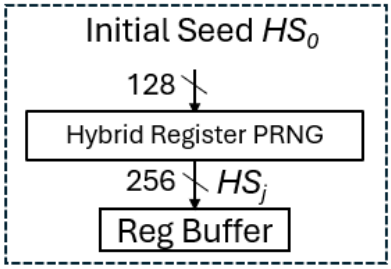

Fig. 4. HS_j Generation in the Modified Permutation Generator Architecture

from a single seed, our proposed architecture eliminates the re-seeding of the PRNG component whenever a new digitized value is received for encoding. As a result, the hybrid register PRNG will take a 128-bit key as input, which we consider to be HS_0 provided by a key source, and subsequently generates 256-bit HS_j ($j > 0$) values for the generation of the four permutation lookup tables, with each permutation table generated using a 64-bit slice of HS_j. The modified portion of the permutation generator architecture is shown in Figure 4. Thus, we make the following modifications to the hybrid register stream cipher for use with RanCode:

- Replacement of the 96-bit IV with a constant 96-bit vector of all 1's in the portion of the hybrid register initial state that previously contained the IV, meaning the only input to the PRNG is the 128-bit seed and that the PRNG has a 128-bit security claim.
- Removal of the authentication-related components from [3], meaning only the 256-bit hybrid register is integrated with RanCode/RanCompute.
- Generation of a 256-bit pseudorandom output, used for generating four permutation LUTs.

IV. Experimental Evaluation

In this section, we showcase the use of RanCode/ RanCompute with the hybrid register PRNG in experiments relevant to critical infrastructure protection and validate the randomness of encoded data. We also showcase a hardware implementation of RanCode/RanCompute with SHA-3 on an FPGA board and compare it to RanCode/RanCompute utilizing the lightweight hybrid register PRNG.

A. Sensor Data Encoding and Randomness Testing

The RanCode architecture is strongly applicable to encoding sensor data in power and industrial control systems to enhance security at the field level. Encoding sensor data ensures that unencoded data does not exist in plaintext in the local memory of the sensor hardware, assisting in the prevention of both software and hardware false data injection attacks. Within SCADA and power systems, RanCode provides inferred authentication of sensors: if an adversary replaces a sensor, a supervisory computer detects only randomized and invalid data. The RanCode approach allows critical measurements such as voltage, current, and temperature to be securely transmitted from field devices to central servers, significantly improving

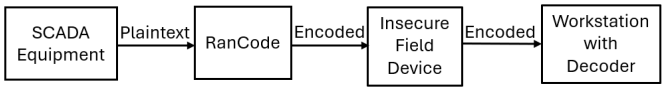

Fig. 5. Emulated Data Flow for the Sensor Encoding Experiment

the resilience of critical infrastructure against cyber-physical attacks. To demonstrate the application of RanCode to a sensor data encoding scenario, we have designed a temperature sensor device that incorporates both a temperature sensor integrated circuit and an Arduino Uno. For the temperature sensor, we use the Texas Instruments LM95172 temperature sensor [13]. The Arduino is used to configure the temperature sensor IC and allow for a serial connection to another computer containing the RanCode/RanCompute encoder and decoder circuits.

In our sensor data encoding experiment, the temperature sensor device measures the ambient temperature in degrees Celsius and continuously forwards analog temperature measurements across one hour at a rate of one measurement per second to a desktop computer running the encoder circuit. Temperature samples are encoded as fixed-point 16-bit digital values in the order in which they arrive. Lastly, each encoded temperature sample is forwarded to a computer implementing the decoder circuit. This experiment seeks to emulate the data flow in Figure 5. In total, $3600 \times 16 = 57{,}600$ bits of sensor data are encoded. We applied the NIST Statistical Test Suite for Pseudorandom Number Generators [21] to the $57{,}600$ bits of encoded data to ensure that the encoded data exhibits random behavior; our encoded temperature sensor data passed all tests in the suite.

B. Image Difference Evaluation

An application of RanCode/RanCompute is image difference, which RanCode/RanCompute can perform by treating each pixel in an image as a numeric value. Image difference can be used, for example, to detect motion in the video feed of security cameras [15]. RanCode/RanCompute can encode pixel data at acquisition time using LUTs derived from the hybrid register PRNG outputs. When applied to image difference, RanCode/RanCompute enables computation of per-pixel differences entirely within the encoded domain. The same randomized encoding that protects sensor data from tampering can be applied to images, detecting subtle changes without exposing the original image content.

To demonstrate the use of RanCode/RanCompute with the hybrid register PRNG in image difference, we feed the grayscale images in Figures 6 and 7 to the randomized data encoder circuit of Figure 1. Grayscale images are used to ensure that each pixel can be processed as an 8-bit value, meaning each pixel requires two 4-bit LUTs to encode. Once each image has been encoded, the absolute value of the difference between the pixels is taken to compute the image difference. For two pixels in position (i, j) in images A and B, denoted as A_{ij} and B_{ij}, the difference is taken as $|A_{ij} - B_{ij}|$. In our experiment, both images show a temperature

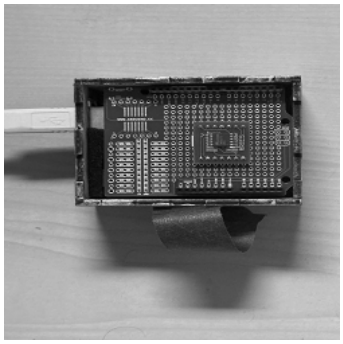

Fig. 6. Unencoded Grayscale Temperature Sensor Image

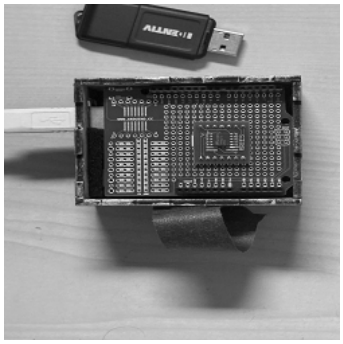

Fig. 7. Unencoded Grayscale Temperature Sensor with Flash Drive Image

sensor device; however, in Figure 7, a flash drive has been placed above the sensor. With the flash drive representing an unauthorized device that poses a security concern, this scenario is broadly applicable to camera-based monitoring systems within power stations and substations or other forms of critical infrastructure monitoring.

The encoded images are shown in Figures 8 and 9, and the encoded image difference is shown in Figure 10. Next, we feed the encoded image difference of Figure 10 into the randomized data decoder circuit of Figure 2, which yields the decoded grayscale image of Figure 11. In the decoded result, regions where the original images differ significantly are brighter, whereas regions with little or no difference are darker.

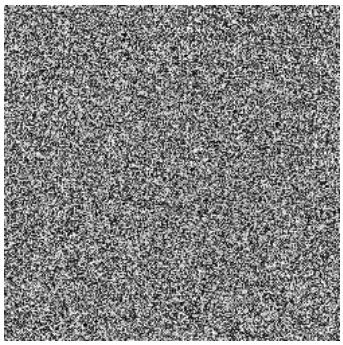

Fig. 8. Encoded Grayscale Temperature Sensor Image

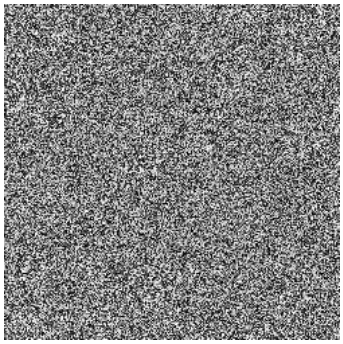

Fig. 9. Encoded Grayscale Temperature Sensor with Flash Drive Image

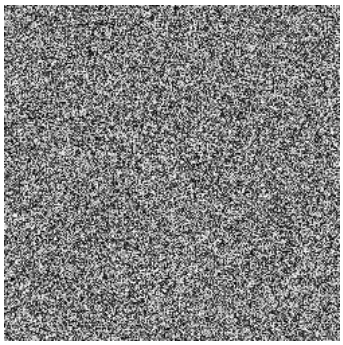

Fig. 10. Encoded Image Difference

C. Hardware Implementation Analysis

We present an FPGA hardware implementation comparison between RanCode with SHA-3 as the PRNG component and RanCode with the hybrid register PRNG, comparing the FPGA utilization in terms of registers, adaptive logic modules (ALMs), digital signal processing (DSP) blocks, and clock frequency. ALMs are combinational logic elements in the Intel Quartus Prime Lite Edition software [9]. Each design was synthesized onto the Intel DE10-Standard board [10]. The Intel DE10-Standard board is a Cyclone®V FPGA [11] with 28nm process technology [11]. The FPGA implementation results, shown in Tables I and II, demonstrate a 63% reduction in ALM usage, 86% reduction in register usage, and, for the RanCode encoder circuit, a 20 MHz increase in operating frequency. In Tables I and II, the two rows for the permutation generator and shuffle unit do not include the PRNG or register buffer circuitry of Figures 1 and 2; as a result, zero registers are utilized in those two rows. Additionally, the utilization of DSP blocks, which is exclusively due to the permutation generator and shuffle unit circuits (independent of the PRNG), does not change across the two RanCode implementations.

Fig. 11. Decoded Image Difference

TABLE I
DE10-STANDARD FPGA HARDWARE IMPLEMENTATION RESULTS FOR RANCODE WITH SHA-3

Circuit	ALMs	Registers	DSP	Frequency (MHz)
Encoder	4044	3694	44	180
Decoder	4088	3461	44	200
Permutation Gen.	1346	0	44	200
Shuffle Unit	418	0	11	200

TABLE II
DE10-STANDARD FPGA HARDWARE IMPLEMENTATION RESULTS FOR RANCODE WITH HYBRID REGISTER PRNG

Circuit	ALMs	Registers	DSP	Frequency (MHz)
Encoder	1490	503	44	200
Decoder	1534	470	44	200
Permutation Gen.	1346	0	44	200
Shuffle Unit	418	0	11	200

The drastic FPGA utilization reduction across Tables I and II is due to replacing the SHA-3 core in the original RanCode specification with the hybrid register PRNG. Whereas SHA-3 requires the implementation of five distinct permutation components, θ, ρ, π, χ, and ι [8], each of which requires dedicated lookup tables and registers, the hybrid register PRNG only requires 256 flip-flops, with AND and XOR logic gates for feedback and chaining function logic.

Beyond reduced FPGA utilization, RanCode/RanCompute using the hybrid register PRNG also offers a lower encoding latency. In [14], it was determined that the number of clock cycles that comprise the RanCode encoding latency was dominated by the PRNG component, which was found to be 830 clock cycles for SHA-3. For the hybrid register PRNG, the latency to generate a 256-bit value for a single encoding is only 384 clock cycles, formed by the sum of the 128 clock cycles required to initialize the hybrid register PRNG and the 256 clock cycles required to generate a 256-bit output. For subsequent encodings, since the hybrid register PRNG does not undergo the 128-cycle initialization phase again, the latency is only 256 clock cycles per encoding. Moreover, even when operated continuously at 200 MHz, the hybrid register PRNG will take $\frac{2^{81}}{(2.0\times10^8)(86400\times365)} \approx 3.8 \times 10^8$ years to reach its limit of 2^{81} bits, indicating that using the hybrid register PRNG with RanCode/RanCompute is suitable for real-world scenarios involving long operational lifetimes.

We can also compute the number of pixel operations per frame for RanCode/RanCompute using the hybrid register PRNG. Assuming 720p images as input, which have a standard

horizontal resolution of 1280 pixels, the number of pixels in a single frame is $1280 \times 720 = 921{,}600$ pixels per frame. From Table II, the encoder circuit can operate at 200 MHz. Given that the latency for querying a RanCode/RanCompute LUT is 8 clock cycles [14], the per-pixel compute time t_p is $t_p = 8/200$ MHz$^{-1} = 40$ ns. The frame time, t_f, is then $t_f = 921{,}600 \times 40$ ns ≈ 37 ms. Therefore, the maximum throughput for RanCode/RanCompute using the hybrid register PRNG is $1/37$ ms$^{-1} \approx 27$ FPS, compared to the 15 FPS result of [14]. Our results indicate that image difference with RanCode/RanCompute and the hybrid register PRNG can potentially be performed on camera footage from standard commercial and consumer CCTV security cameras [16], [20].

V. Discussion and Conclusion

The methods described in this paper reduce the hardware implementation cost and improve the performance of a data encoding architecture by targeting the dominant bottleneck: randomness provisioning for the permutation generator component. By replacing the SHA-3 core in RanCode/RanCompute with the hybrid register PRNG, we retain the cryptographic randomness required to generate pseudorandom LUTs while alleviating the hardware implementation burden of a SHA-3 core. The result is a compact RanCode/RanCompute instantiation that is simpler to deploy on resource-constrained systems, offering reduced encoding latency with respect to both clock cycles and operating frequency, and the ability to perform 720p image difference at a higher refresh rate. Limitations and future directions of this work include the following:

- Further cryptanalytic study of the hybrid register when used as a PRNG rather than an authenticated stream cipher.
- Key provisioning and management protocols for field deployment.
- The specification of a real-time reconfigurable hardware architecture for RanCode/RanCompute.

Collectively, our findings suggest a pragmatic path to hardening field-level data security by moving computation into the encoded domain and keeping plaintext out of insecure field devices by ensuring that the PRNG component of the encoding scheme is sufficiently compact and fast to support real-world critical infrastructure applications.

References

[1] M. Goresky and A. M. Klapper, "Fibonacci and Galois representations of feedback-with-carry shift registers," in *IEEE Transactions on Information Theory*, vol. 48, no. 11, pp. 2826-2836, Nov. 2002, doi: 10.1109/TIT.2002.804048.

[2] D. Gordon, A. Allahverdi, S. Abrelat, A. Hemingway, A. Farooq, I. Smith, N. Arora, A. Chang, Y. Qiang, and V. Mooney, "Scalable nonlinear sequence generation using Composite Mersenne Product Registers," *IACR Commun. Cryptol.*, vol. 1, no. 4, Jan. 2025, doi: 10.62056/a3tx11zn4.

[3] A. Allahverdi and V. Mooney, "A hardware-efficient AEAD stream Cipher based on a hybrid nonlinear feedback register structure," *2025 IEEE International Conference on Cyber Security and Resilience (CSR)*, Chania, Crete, Greece, 2025, pp. 1016-1023, doi: 10.1109/CSR64739.2025.11130096.

[4] E. Dubrova, "A List of Maximum Period NLFSRs," *Cryptology ePrint Archive*, Paper 2012/166, 2012. [Online]. Available: https://eprint.iacr.org/2012/166

[5] E. Dubrova, "A scalable method for constructing Galois NLFSRs with period $2^n - 1$ using cross-join pairs," in *IEEE Transactions on Information Theory*, vol. 59, no. 1, pp. 703-709, Jan. 2013, doi: 10.1109/TIT.2012.2214204.

[6] K. Hutto, S. Grijalva and V. Mooney, "RanCompute: Computational Security in Embedded Devices via Random Input and Output Encodings," *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 2022, pp. 1-8, doi: 10.1109/MECO55406.2022.9797150.

[7] K. Hutto, S. Grijalva and V. Mooney, "Hardware-Based Randomized Encoding for Sensor Authentication in Power Grid SCADA Systems," 2022 IEEE Texas Power and Energy Conference (TPEC), College Station, TX, USA, 2022, pp. 1-6, doi: 10.1109/TPEC54980.2022.9750706.

[8] M. J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Jul. 2015, doi: https://doi.org/10.6028/nist.fips.202.

[9] Intel Corporation, "Intel® Quartus® Prime Lite Edition Design Software Version 20.1.1 for Windows," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/software-kit/660907/intel-quartus-prime-lite-edition-design-software-version-20-1-1-for-windows.html (accessed: Oct. 7, 2025)

[10] Terasic Inc., "DE10-Standard," 2017. [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English\&CategoryNo=165\&No=1081 (accessed: Sept. 26, 2024)

[11] Intel Corporation, "Cyclone® V FPGA and SoC FPGA," 2013. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/v.html (accessed: Oct. 7, 2025)

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

[13] Texas Instruments, *LM95172 13-Bit to 16-Bit 200°C Digital Temperature Sensor with 3-Wire Interface*, Rev. B, Dallas, TX, USA: Texas Instruments, Mar. 2013. [Online]. Available: https://www.ti.com/product/LM95172

[14] K. Hutto, "Remote Sensor Security Through Encoded Computation and Cryptographic Signatures," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, 2024. [Online]. Available: https://hdl.handle.net/1853/75289

[15] M. A. Smith and T. Chen, "9.1 – Image and video indexing and retrieval," in *Handbook of Image and Video Processing* (2nd ed.), A. Bovik, Ed., ser. Communications, Networking and Multimedia. Burlington, MA, USA: Academic Press, 2005, pp. 993–996. ISBN: 978-0-12-119792-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780121197926501212

[16] H. Keval and M. A. Sasse, "To catch a thief – you need at least 8 frames per second: the impact of frame rates on user performance in a CCTV detection task," in *Proc. 16th ACM Int. Conf. on Multimedia (MM '08)*, New York, NY, USA: ACM, 2008, pp. 941–944, doi: 10.1145/1459359.1459527.

[17] R. Geelen *et al.*, "Basalisc: Programmable hardware accelerator for BGV fully homomorphic encryption," *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2023, no. 4, pp. 32–57, Aug. 2023. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/11157

[18] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proc. 3rd Innovations in Theoretical Computer Science Conf. (ITCS '12)*, Cambridge, MA, USA: ACM, 2012, pp. 309–325. ISBN: 9781450311151. [Online]. Available: https://doi.org/10.1145/2090236.2090262

[19] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology – ASIACRYPT 2017*, T. Takagi and T. Peyrin, Eds. Cham: Springer Int. Publishing, 2017, pp. 409–437. ISBN: 978-3-319-70694-8.

[20] O. Elharrouss, N. Almaadeed, and S. Al-Maadeed, "A review of video surveillance systems," *J. Vis. Commun. Image Represent.*, vol. 77, 2021, Art. no. 103116. ISSN: 1047-3203. doi: 10.1016/j.jvcir.2021.103116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1047320321000729

[21] A. Rukhin *et al.*, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22 Rev. 1a, Apr. 2010. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf