Cryptographic Primitive Design for Reconfigurable Hardware using CMPRs

*Arman Allahverdi and *^Vincent John Mooney III

*School of Electrical and Computer Engineering
^ School of Computer Science
Georgia Institute of Technology
Atlanta, Georgia

Acknowledgement

- 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

© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
    - Feedback Registers
    - Product Registers
    - The Sponge Construction
    - Intel FPGA
    - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

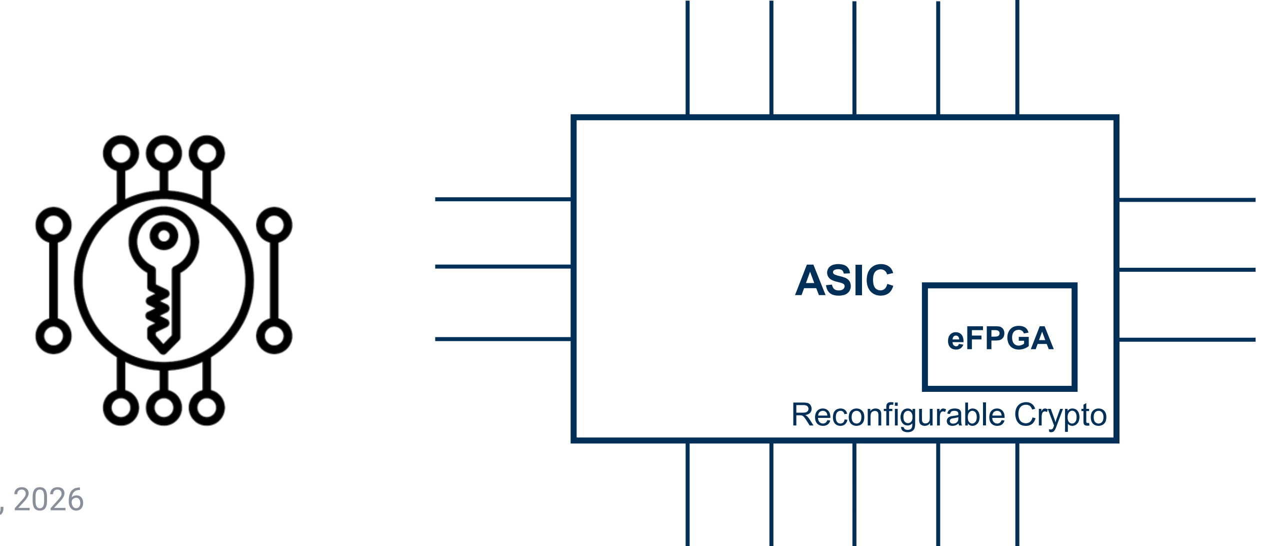
Outline

- Introduction and Motivation
- Background
    - Feedback Registers
    - Product Registers
    - The Sponge Construction
    - Intel FPGA
    - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Introduction: Motivation

- Growing need for cryptographic hardware:
  - Embedded systems
  - Trusted platform modules (TPMs)
  - Resource-constrained devices

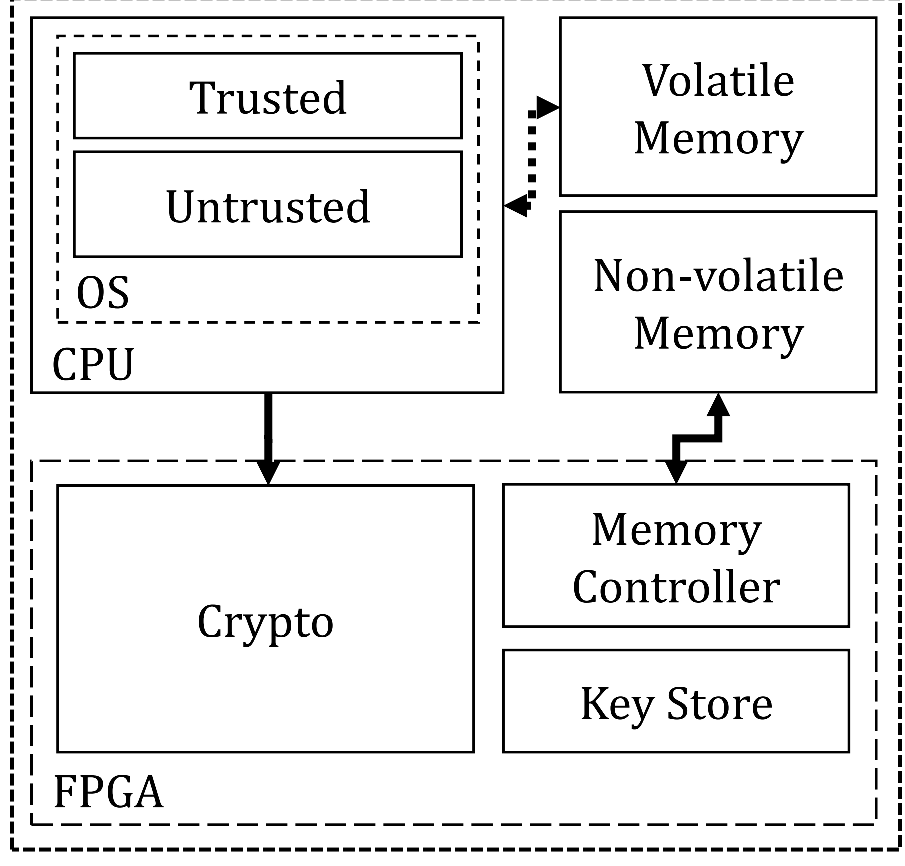
- **Emerging Area of Interest:** Reconfigurable hardware for security (for example, the use embedded FPGAs in the design of hardware security primitives) [9, 10]



© Vincent John Mooney III, 2026

Introduction: Research Impact

- We have discovered a new, scalable class of nonlinear feedback register that can outperform the prior state-of-the-art in lightweight cryptography
  - Scalable to large values (e.g., 256, 1024 and larger) based on Mersenne primes
    - Primes of the form $2^n - 1$ for some integer n
  - Mathematical proof of exponential expected period
  - Hardware-efficient due to register-based construction
  - Exponential number of hardware constructions that allow for hardware-based cryptographic keys in reconfigurable hardware



- In summary, this research provides the first scalable nonlinear feedback structure with provable exponential state and a similarly exponential number of distinct structures

© Vincent John Mooney III, 2026

Introduction: Literature

- The research covered by this presentation is from the following publications from 2025 and 2026:

Scalable Nonlinear Sequence Generation using Composite Mersenne Product Registers

David Gordon, Arman Allahverdi, Simon Abrelat, Anna Hemingway, Adil Farooq, Isabella Smith, Nitya Arora, Allen Ian Chang, Yongyu Qiang and Vincent John Mooney III

Georgia Institute of Technology, Atlanta, United States of America

[1]

Designed a Family of Three Stream Ciphers

CASH: Lightweight Keyed Hash Function Design for Reconfigurable Hardware

Arman Allahverdi*, Adil Farooq†, Anias Pullen†, Yongyu Qiang†, 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, adilfarooq, apullen6, yqiang7, mooney, sgrijalva6}@gatech.edu

[23]

Designed a Message Authentication Code for FPGAs

© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Background: Feedback Registers and Periodicity

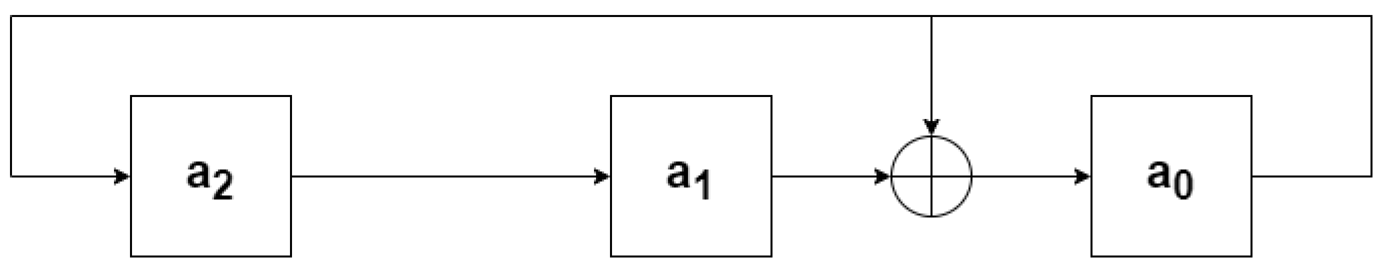
- A feedback register is a register A such that on each clock cycle, its state $\mathrm{A[t]}$ is updated according to some feedback function f, such that
  - $\boldsymbol{A[t+1] = f(A[t])}$
- In the literature, feedback registers show up in many different forms
  - **LFSRs [1]:** f is a linear function of $\mathrm{A[t]}$
  - **NLFSRs:** f is a nonlinear function of $\mathrm{A[t]}$
  - **De Bruijn Sequences, T-Functions, Cross-Joined Pairs [1, 8]:** Derived from LFSR and NLFSR theory, imposing special restrictions on f

- LFSRs and NLFSRs are parametrized by their feedback polynomials $\mathrm{P(x)}$
- For a Galois LFSR, the state updates according to:
  - $\boldsymbol{A[t+1] \; = \; xA[t] \; mod \; P(x)}$

© Vincent John Mooney III, 2026

Background: Feedback Registers and Periodicity

- An n-bit feedback register can hold one of 2^n possible unique states at a given point in time
- The **period** of a feedback register refers to the number of unique states the register undertakes before returning to its starting (initial) state, but it is not always possible to reach a period of 2^n
- LFSRs have a maximum attainable period of $2^n - 1$ for an n-bit register size
- NLFSRs typically also have a maximum attainable period of $2^n - 1$ for an n-bit register size



A 3-bit LFSR (Galois “Internal-XOR” Configuration)

© Vincent John Mooney III, 2026

Background: Feedback Registers and Periodicity

- Both LFSRs and NLFSRs have a maximum attainable period of $2^n - 1$ for an n-bit register size
  - Maximum/full-period LFSRs can be constructed by ensuring $P(x)$ is a *primitive polynomial*
    - A primitive polynomial is an irreducible polynomial whose roots generate all nonzero elements of a finite field
    - The primitivity of a polynomial can be verified using well-known algorithms [11]
  - Full-period NLFSRs exist for small register sizes [7]
  - **No scalable methodology or mathematical process for constructing full-period NLFSRs of arbitrary size**
    - **Limited exceptions (e.g., cross-joined pairs [8], which impose restrictions on the feedback function)**

© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Background: Product Registers

- Product Registers (PRs) [1] are a generalization of Galois LFSRs, allowing for the internal state to be permuted in a general way rather than restricting to a shift

- An n-bit **Product Register** is parametrized by the following two polynomials
  - A **feedback polynomial** $P(x)$, where $\deg\big(P(x)\big) = n$
  - An **update polynomial** $U(x)$, where $\deg(U(x)) \leq n - 1$
    - $U(x)$ and $P(x)$ can be represented either as binary numbers or mathematical expressions
    - **Example:** $1011 = x^3 + x + 1$

- The state of a PR updates according to the following:
  - $\boldsymbol{A[t+1] = U(x)A[t]\ mod\ P(x)}$

© Vincent John Mooney III, 2026

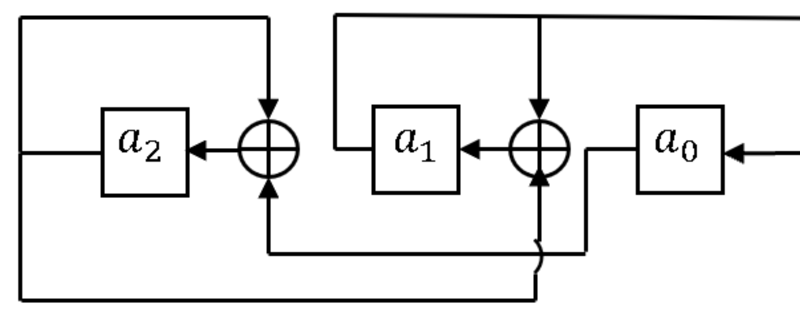
Background: Product Registers

- The state of a PR updates according to the following:
  - $\boldsymbol{A[t+1] = U(x)A[t] \bmod P(x)}$
- From the above, observe that $U(x) = 0, 1$ result in degenerate behavior
  - $U(x) = 0$ causes a PR to remain stuck in the all-zeros state
  - $U(x) = 1$ causes a PR to remain stuck in its initial state
- We omit $U(x) = 0, 1$ from the set of “valid” update polynomials for a PR
  - **=> An n-bit PR has $2^n - 2$ possible valid U(x)**

© Vincent John Mooney III, 2026

Background: Mersenne Product Registers

- A Mersenne Product Register (MPR) is a Product Register whose size is a Mersenne exponent
  - Mersenne exponent: Integer n such that $2^n - 1$ is a prime number
  - Ex: 2-, 3-, and 5-bit PR are MPRs
  - **Larger MPRs:** 61-, 89-, 127-bit
  - After $n = 127$, gaps between Mersenne exponents become larger



A 3-bit MPR with $P(x) = x^3 + x + 1$ and $U(x) = x^2$

- An n-bit MPR achieves period $2^n - 1$ provided $P(x)$ is primitive and $U(x) \neq 0, 1$

© Vincent John Mooney III, 2026

Background: Mersenne Product Registers

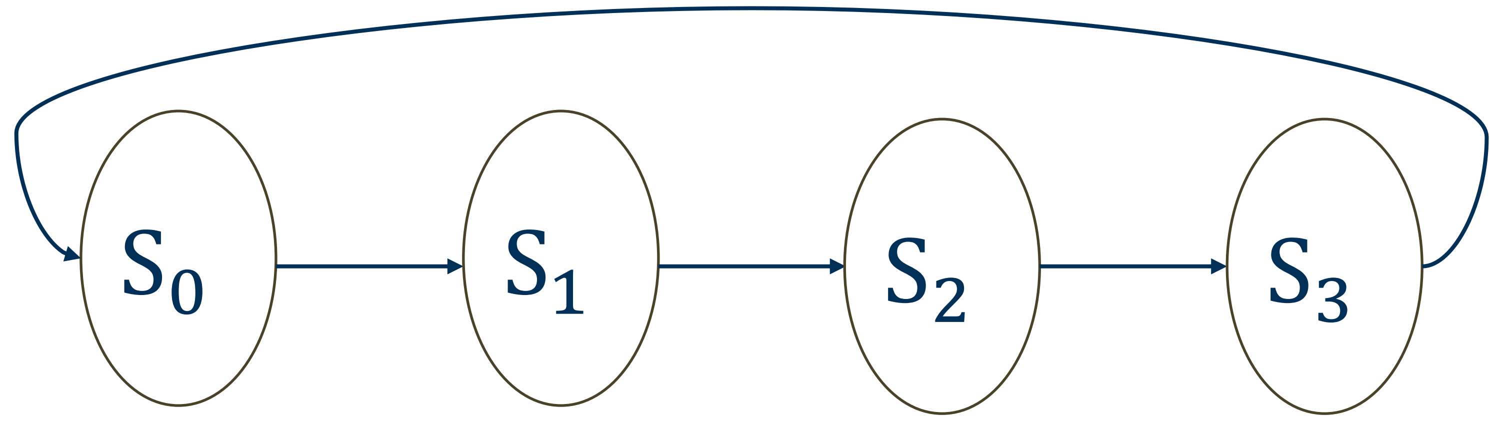
- The set of possible states of an MPR always remains the same
- Changing $U(x)$ permutes the order in which an MPR traverses its possible states, even when the MPR is seeded to the same initial state
- Example from [1]:
  - 3-bit MPR
  - Fixed primitive $\mathrm{P(x)} = \mathrm{x}^3 + \mathrm{x} + 1$ => MPR has full period of $2^3 - 1 = 7$
  - **Initial State:** 001
  - Vary $\mathrm{U(x)}$

Update	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
x	001	010	100	101	111	011	110
$x+1$	001	011	101	010	110	111	100
x^2	001	100	111	110	010	101	011
x^2+1	001	101	110	100	011	010	111
x^2+x	001	110	011	111	101	100	010
x^2+x+1	001	111	010	011	100	110	101

© Vincent John Mooney III, 2026

Background: Mersenne Product Registers

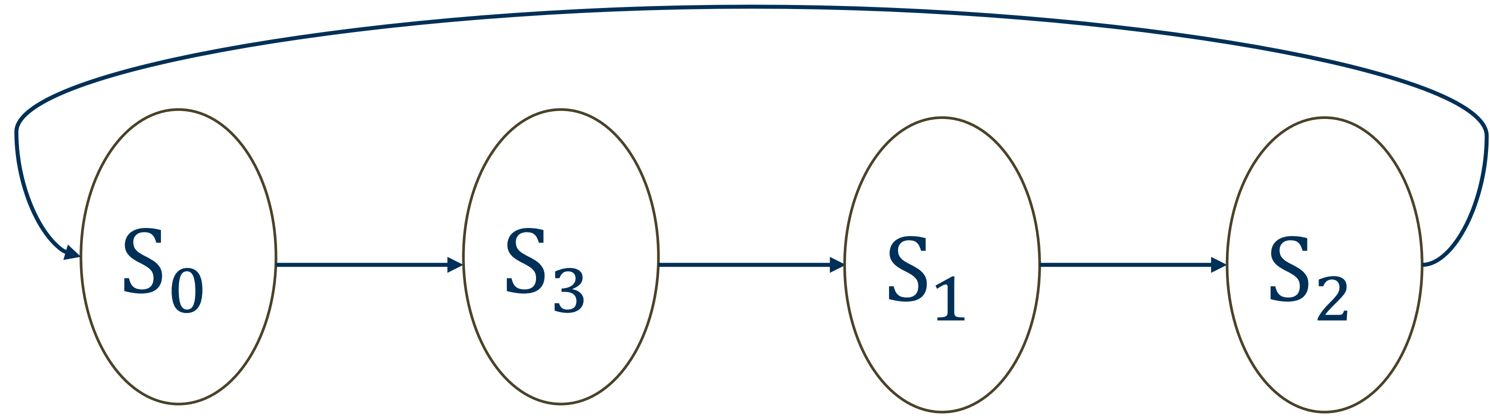
- **State Machine Analogy:** Changing $U(x)$ alters the structure of the FSM representation of an MPR



© Vincent John Mooney III, 2026

Background: Mersenne Product Registers

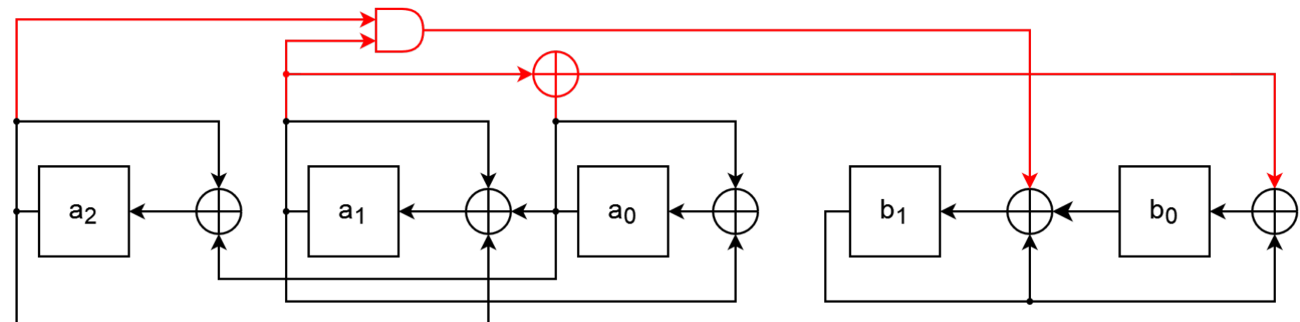
- **State Machine Analogy:** Changing $U(x)$ alters the structure of the FSM representation of an MPR



© Vincent John Mooney III, 2026

Background: Composite Mersenne Product Registers

- On their own, MPRs are linear, meaning they cannot be used as a standalone cryptographic primitive
- It is possible to connect MPRs to form a larger, nonlinear structure called a Composite Mersenne Product Register (CMPR)
- MPRs can be connected using **chaining functions**, or sets of Boolean functions that allow the state of an MPR to affect another MPR, to create a CMPR
- CMPRs can be made nonlinear by using multiplicative terms (e.g., AND gates) in the chaining functions



A 5-bit CMPR formed by chaining a 3-bit MPR into a 2-bit MPR (Chaining function shown in red)

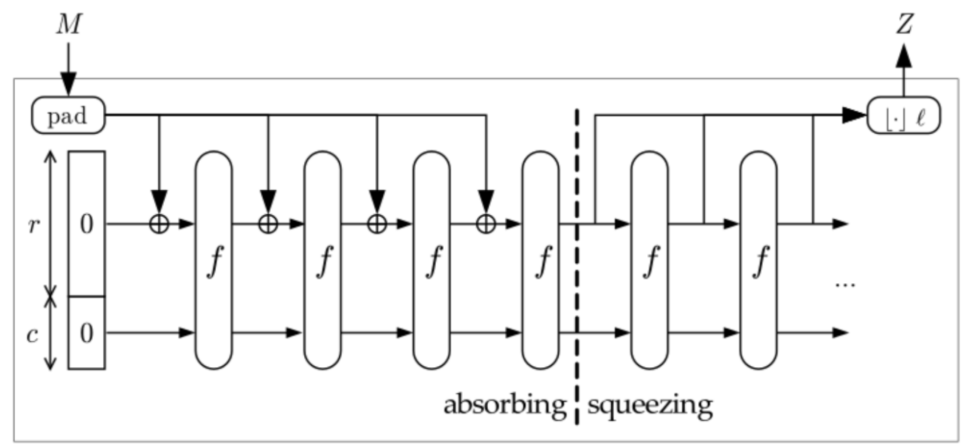
© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Background: The Sponge Construction

- Our architecture integrates a CMPR-based permutation with the Sponge Construction [16, 17] to create a message authentication code



The Sponge Construction with a Generic Permutation f

- Generally, the Sponge Construction partitions the state of a finite-sized permutation f into a rate r and capacity c, where the latter portion is not directly accessible to an adversary
- We opt to use the Full-State Keyed Sponge Construction [17], which allows for faster intake of input data

© Vincent John Mooney III, 2026

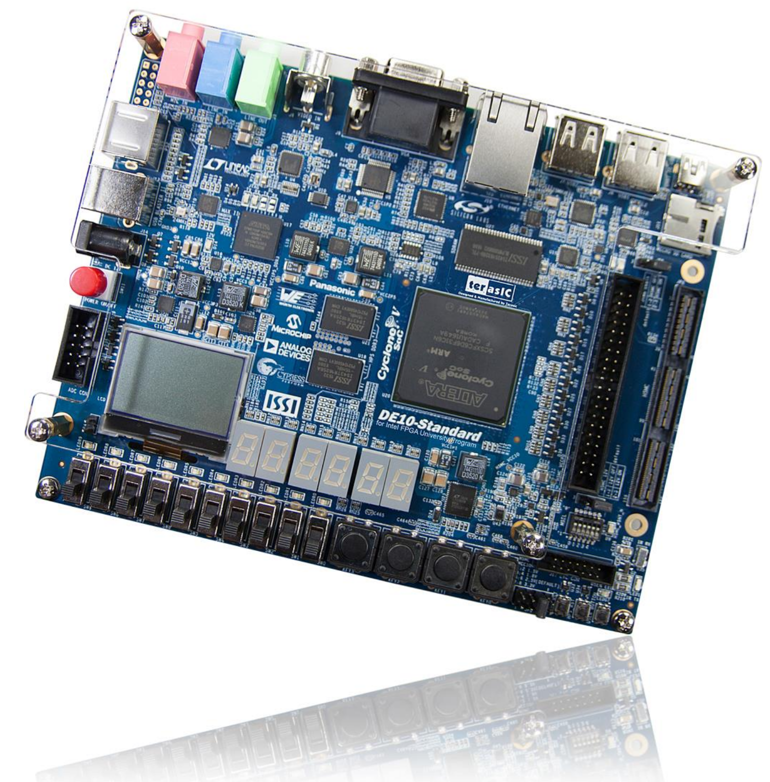
Georgia Tech

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Background: Intel FPGA

- The Intel DE10-Standard board [22] is an FPGA board belonging to the Cyclone V device family
  - Dual-core ARM Cortex-A9 CPU
  - 28nm process (TSMC 28LP)

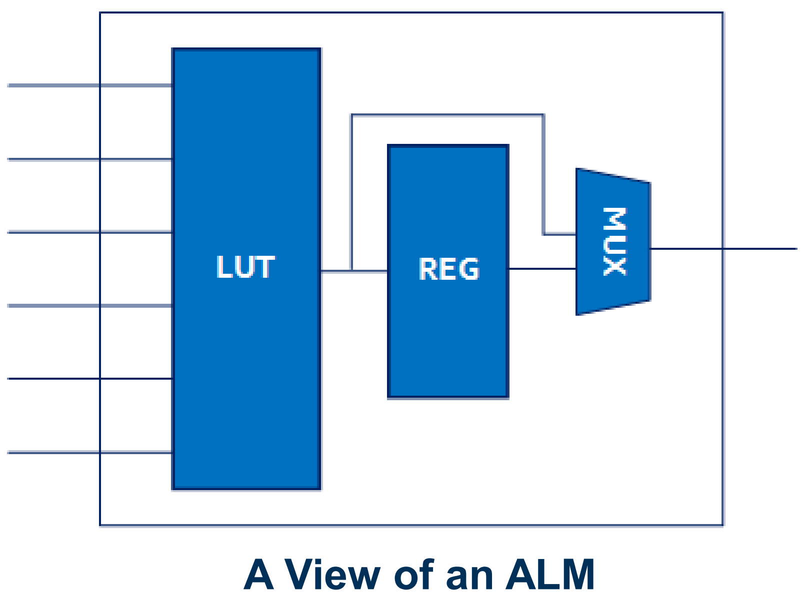


© Vincent John Mooney III, 2026

Background: Intel FPGA

- Intel FPGAs quantify resource utilization using the **adaptive logic module (ALM)**

- An ALM contains the following:
  - 4x (one-bit) registers
  - 8-input fracturable LUT
  - 2x full adders



A View of an ALM

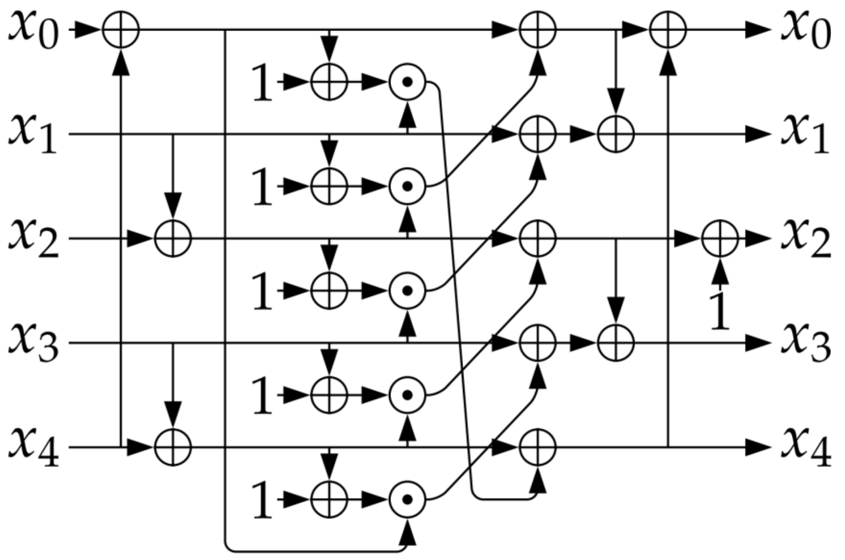
© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Selected Baselines

- Throughout this presentation, we will compare our work to the following finalist algorithms from the NIST Lightweight Cryptography Competition (LWC) (2018-2023):
  - Ascon [12] (NIST LWC Winner)
    - Sbox-based hash
  - Romulus [13]
    - Block cipher-based hash
  - Xoodyak [14]
    - Sbox-based hash
  - PHOTON-Beetle [15]
    - Sbox-based hash



Boolean Logic Representation of the Ascon S-box [12]

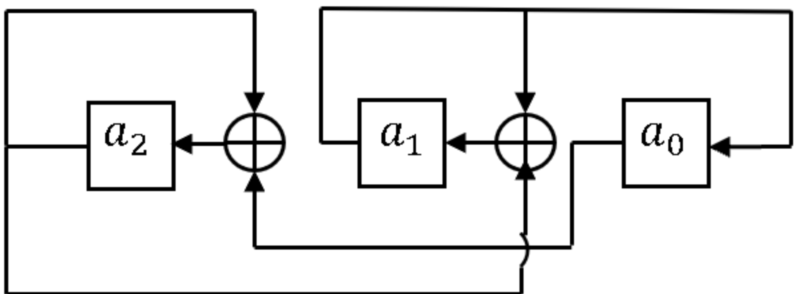
© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Product Registers on FPGAs

- Product Registers can be implemented efficiently on FPGAs
- **Example #1:** 3-bit MPR, $U_3(x) = x^2, P_3(x) = x^3 + x + 1$



- 3x FF
- 2x XOR
- => 1 Intel ALM required to implement

© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Hardware-Embedded Keys using CMPRs

- **Recall:** The $U(x)$ space for an MPR is exponential with respect to register size
  - => On reconfigurable hardware, $U(x)$ can be used as a hardware-based cryptographic key
  - Re-keying requires resynthesis
- We have designed a keyed reconfigurable permutation based on a 192-bit CMPR for use with a message authentication code (MAC) scheme
  - **Targeted Applications:** Due to the re-keying overhead, we focus on applications that do not require frequent key updates, such as:
    - Secure boot
    - TPMs
    - Device identification
- **C**MPR-based **A**rea-efficient **S**ponge **H**ash (CASH) [23]

© Vincent John Mooney III, 2026

Hardware-Embedded Keys using CMPRs: Architecture

- We design the CASH permutation utilizing the following 192-bit CMPR:

MPR Size	U(x)	P(x)
107 bits	105-bit Key Fragment	$x^{107} + x^{59} + x^{54} + x^{39} + 1$
61 bits	23-bit Key Fragment	$x^{61} + x^{44} + x^{19} + x^{15} + 1$
19 bits	$x^{17} + 1$	$x^{19} + x^{5} + x^{2} + x + 1$
3 bits	$x^{2} + x$	$x^{3} + x + 1$
2 bits	$x + 1$	$x^{2} + x + 1$

- The P(x) are chosen to be primitive, to ensure full periodicity for each MPR
- 105 + 23 = 128 bits of the update polynomials for the 107- and 61-bit MPRs are designated as key bits and allowed to vary
- **Chaining Functions:**
  - Only connect from one MPR to the next (107 -> 61 -> 19 -> 3 -> 2)
  - Balanced (equal number of 0's and 1's in their truth tables)
  - (At most) 4-input XOR, 4-input AND

© Vincent John Mooney III, 2026

Hardware-Embedded Keys using CMPRs: Permutation

- Sponge Construction parameters:
  - $r = 64$
  - $c = 128$
- We base our security claim on the 128-bit key embedded within the update polynomials
- CASH operates according to the *CASH permutation*, which is used with the Sponge Construction

Symbol	Meaning
S	Internal state of the CMPR
S.nextstate()	Next-state function of the CMPR
S.swap()	Swaps the upper and lower halves of S
S.permute()	Denotes the permutation in Algorithm 1
n	CMPR size
j	Number of n-bit message blocks
1^i	An i-bit string of 1's
k	128-bit key
M_i	192-bit message block
M	Variable-length message
H_i	64-bit digest block
H	256-bit digest
$*$	Denotes a quantity of arbitrary length
$\|\|$	Concatenation

© Vincent John Mooney III, 2026

Hardware-Embedded Keys using CMPRs: Permutation

- CASH Permutation:
  - 32 clock cycles per permutation call

Algorithm 1 CASH Permutation Algorithm

```
1: procedure PERMUTE
2:     for j ← 0...3 do
3:         for i ← 0...7 do
4:             S.nextstate()
5:         end for
6:         if j ≠ 3
               S.swap()
7:         end if
8:     end for
9: end procedure
```

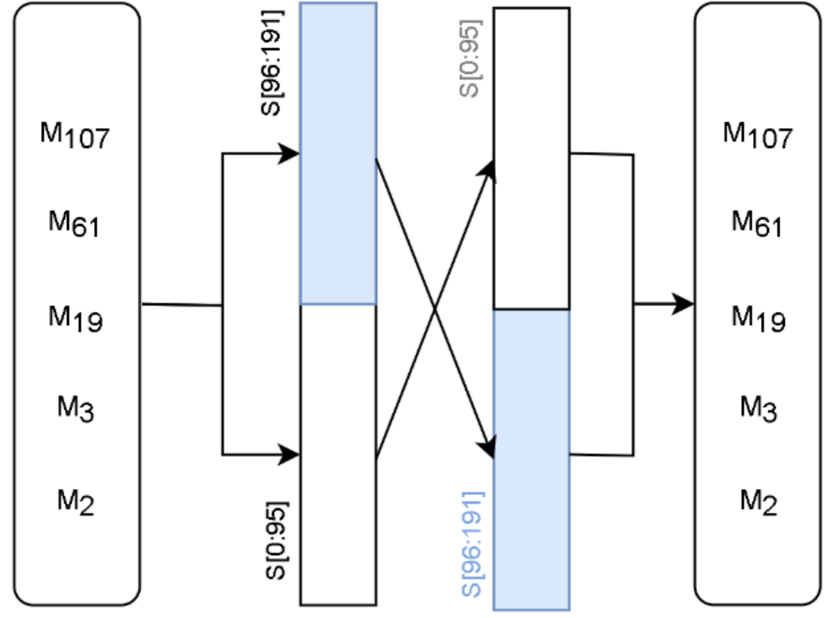


Fig. 2. CASH Permutation

© Vincent John Mooney III, 2026

Hardware-Embedded Keys using CMPRs: Hash

- CASH Hash Generation:
  - 256-bit digest

Algorithm 2 CASH Algorithm

1: **procedure** INITIALIZE
2: $S \leftarrow 1^{192}$
3: $S.permute()$
4: **end procedure**
5: **procedure** ABSORB
6: $M_0, \ldots, M_{j-1} \leftarrow M \| 1 \| 0^*$ ▷ Sponge Padding
7: **for** $i \leftarrow 0 \ldots j-1$ **do**
8: $S \leftarrow S \oplus M_i$
9: $S.permute()$
10: **end for**
11: **end procedure**
12: **procedure** SQUEEZE
13: **for** $i \leftarrow 0 \ldots 3$ **do**
14: $S.permute()$
15: $H_i \leftarrow S(63), \ldots, S(0)$
16: **end for**
17: **end procedure**
18: $H \leftarrow H_0 \| H_1 \| H_2 \| H_3$

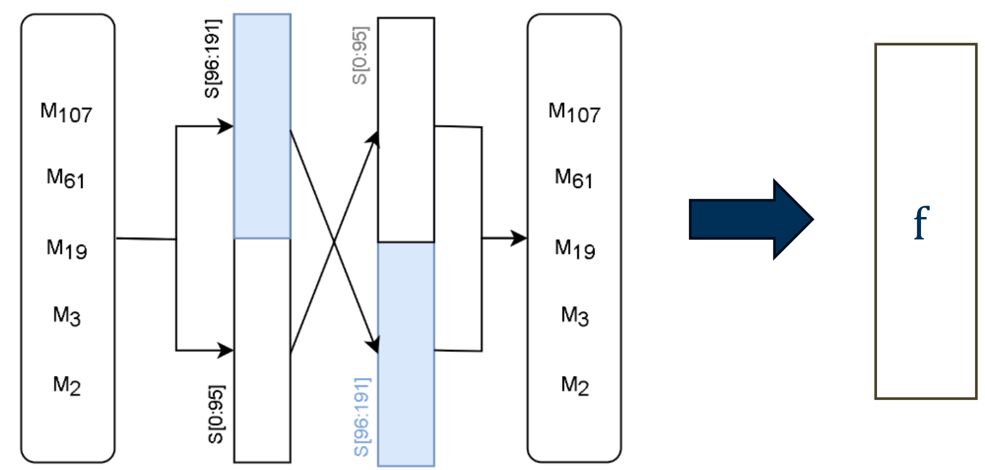
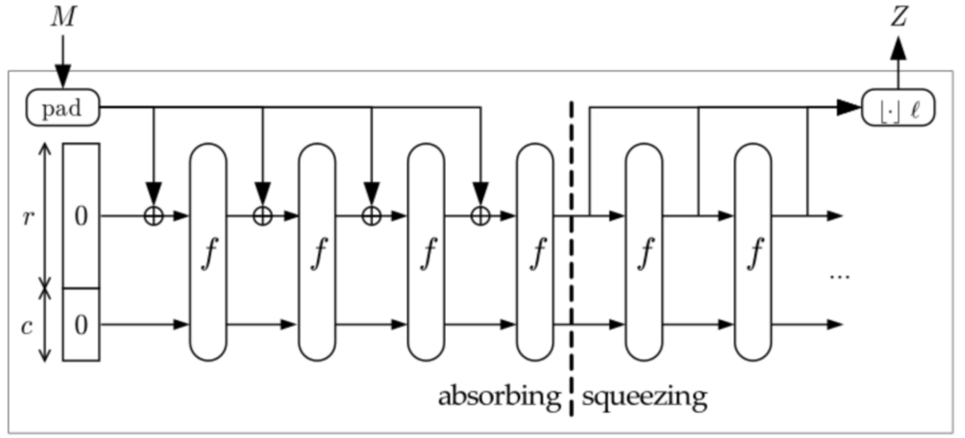


Fig. 2. CASH Permutation



© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Hardware Implementation

- Synthesized the following designs onto the Intel DE10-Standard [22] FPGA board:
  - **CASH**
  - **Ascon (NIST Lightweight Cryptography Standard)**
  - **Romulus (NIST Lightweight Cryptography Competition Finalist)**
  - **Xoodyak (NIST Lightweight Cryptography Competition Finalist)**
  - **PHOTON-Beetle (NIST Lightweight Cryptography Competition Finalist)**
- The key is embedded in $U(x)$; therefore, the CASH synthesis run uses a constant key at synthesis time
- To better equalize the hardware comparison, the NIST LWC designs are modified to also use a constant key
  - => All synthesized designs benefit, to an extent, from constant optimization

© Vincent John Mooney III, 2026

Hardware Implementation

- FPGA synthesis comparison:
  - Adaptive Logic Modules (ALMs) are the combinational logic block in the Intel FPGA suite
  - η represents throughput-per-ALM
- CASH achieves **between 44.5% and 75% lower FPGA utilization** than the selected NIST LWC algorithms
  - Utilization evaluated via ALMs, since ALMs contain LUTs and register bits

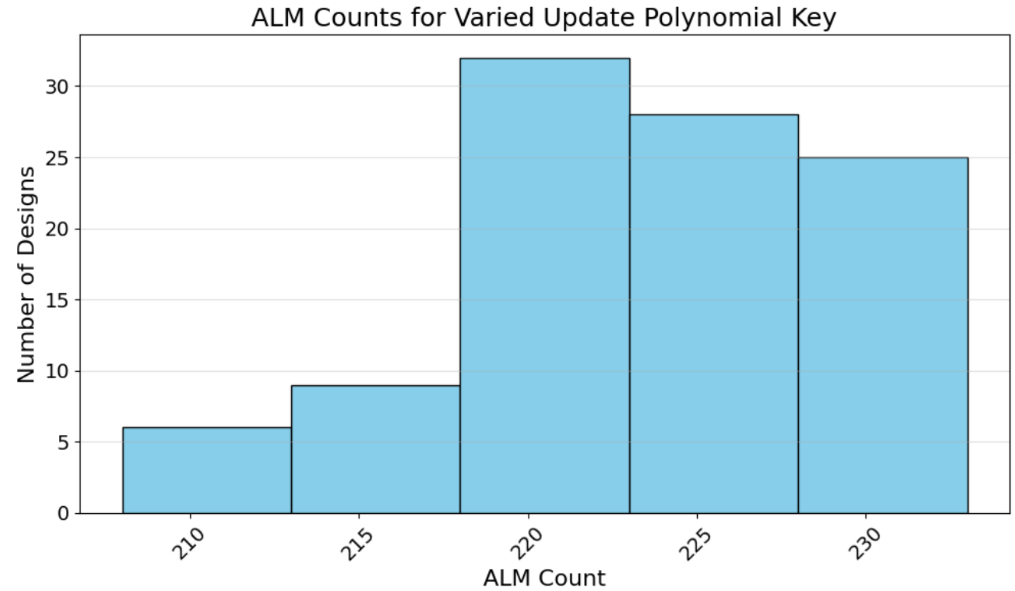
Design	ALMs	F_{max} (MHz)	Throughput (Mbps)
CASH	222	289	578.0
ASCON	400	239	1274.7
Xoodyak	853	218	2325.3
PHOTON-Beetle	892	242	645.3
Romulus	453	301	963.2

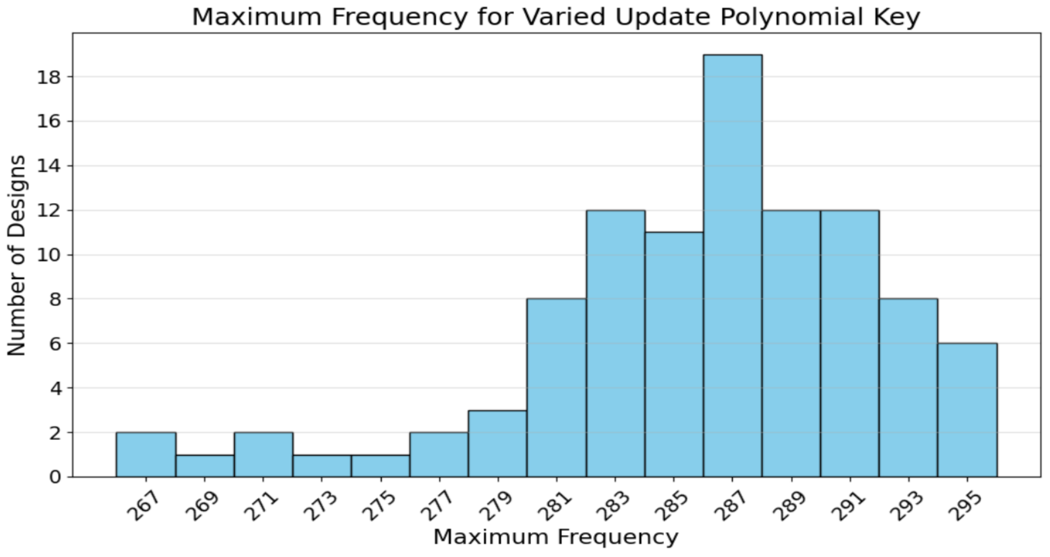
Design	Latency (Cycles)	ADP (ALM · ns)	η (Mbps/ALM)
CASH	32	768.17	2.60
ASCON	12	1673.64	3.19
Xoodyak	12	3912.84	2.73
PHOTON-Beetle	12	3685.95	0.72
Romulus	40	1504.98	2.13

© Vincent John Mooney III, 2026

Hardware Implementation

- Impact of re-keying on FPGA utilization and clock speed:
  - Generate 100 uniformly-sampled key values and examine the ALM count and F_{max} for each case
  - CASH is not constant-area with respect to changes in the key, but the FPGA utilization and clock speed remain tightly distributed across different keys
  - **Since the key is embedded within the FPGA bitstream, we recommend the use of standard bitstream protection mechanisms with CASH (e.g., bitstream encryption and limiting of debug/readback interfaces)**





© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - The Sponge Construction
  - Intel FPGA
  - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Conclusion

- CASH provides a keyed hash function, or message authentication code, with a 2^{128} security claim based on a hardware-embedded key
- By leveraging the properties of CMPRs, namely the exponential update polynomial space, we proposed a lightweight permutation based on a 192-bit internal state
- On an FPGA target, we compared CASH to 4 selected NIST LWC finalists with similar security claims and found that CASH performs strongly in terms of FPGA utilization and maximum frequency
- **Future work:**
  - Exploring the use of the (primitive) feedback polynomial space as keys
  - The design of permutation-based encryption schemes based on CMPRs
  - Investigation of side channel analysis of CMPRs and CMPR-based cryptosystems
  - Optimization of CMPR-based permutations to reduce latency and improve throughput

© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
    - Feedback Registers
    - Product Registers
    - The Sponge Construction
    - Intel FPGA
    - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

References

[1] 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.

[2] 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.

[3] I. T. L. Computer Security Division, "Lightweight Cryptography | CSRC," *CSRC | NIST*, Jan. 03, 2017. https://csrc.nist.gov/projects/lightweight-cryptography

[4] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Submission to NIST," 2019. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf

[7] E. Dubrova, "A list of maximum period NLFSRs," *IACR Cryptology ePrint Archive*, 2012. [Online]. Available: https://eprint.iacr.org/2012/166.pdf

[8] E. Dubrova, "A scalable method for constructing Galois NLFSRs with period $2^n - 1$ using cross-join pairs," *IEEE Transactions on Information Theory*, vol. 1, no. 59, pp. 703–709, 2013

[9] B. Glas, A. Klimm, K. D. Muller-Glaser, and J. Becker, "Configuration measurement for FPGA-based trusted platforms," in *Proc. IEEE/IFIP Int. Symp. Rapid System Prototyping*, Paris, France, 2009, pp. 123–129.doi: 10.1109/RSP.2009.28.

[10] H. Zhao and P. Ratazzi, "A lightweight hardware-assisted security method for eFPGA edge devices," *IEEE Internet of Things Journal*, vol. 11, no. 13, pp. 23673–23682, Jul. 2024. doi: 10.1109/JIOT.2024.3391661.

[11] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1997, doi: 10.1201/9780429466335.

© Vincent John Mooney III, 2026

Georgia Tech

References

[12] J.P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia, “ASCON v1.2: Submission to NIST,” 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf

[13] Z. Bao, A. Chakraborti, N. Datta, J. Guo, M. Nandi, T. Peyrin, and K.Yasuda, “PHOTON-Beetle authenticated encryption and hash family,” 2021. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/photon-beetle-spec-final.pdf

[14] C. Guo, T. Iwata, M. Khairallah, K. Minematsu, andT. Peyrin, “Romulus v1.3,” 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/romulus-spec-final.pdf

[15] J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer, “Xoodyak, a lightweight cryptographic scheme,” 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/Xoodyak-spec-round2.pdf

[16] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “On the indifferentiability of the sponge construction,” in *Advances in Cryptology – EUROCRYPT 2008*, N. Smart, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 181–197,ISBN: 978-3-540-78967-3.

[17] B. Mennink, R. Reyhanitabar, and D. Viz´ar, “Security of full-state keyed sponge and duplex: Applications to authenticated encryption,” in *Advances in Cryptology– ASIACRYPT 2015*, T. Iwata and J. H. Cheon, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 465–489, ISBN: 978-3-662-48800-3.

[18] L. E. Bassham et al., “A statistical test suite for random and pseudorandom number generators for cryptographic applications,” Natl. Inst. Stand. Technol., Gaithersburg, MD, USA, Tech. Rep. 800-22,Apr. 2010. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf

[19] S. Pugh, M. S. Raunak, D. R. Kuhn, and R. Kacker, “Systematic testing of lightweight cryptographic implementations,” Natl. Inst. Stand.Technol., Gaithersburg, MD, USA, Tech. Rep., Nov. 6, 2019.

[20] E. Biham and A. Shamir, “Differential cryptanalysis of DES-like cryptosystems,” Journal of Cryptology, vol. 4, no. 1, pp. 3–72, Jan. 1991, doi: 10.1007/BF00630563.
[21] S. Knellwolf, W. Meier, and M. Naya-Plasencia, “Conditional differential cryptanalysis of NLFSR-based cryptosystems,” in Advances in Cryptology – ASIACRYPT 2010, M. Abe, Ed. Berlin, Germany: Springer, 2010, pp. 130–145.

© Vincent John Mooney III, 2026

Georgia Tech

References

[22] Terasic, Inc., “DE10-Standard,” 2017. [Online]. Available:https://www.terasic.com.tw/cgibin/page/archive.pl?Language=English\&CategoryNo=165\&No=1081

[23] A. Allahverdi, A. Farooq, A. Pullen, Y. Qiang, V. Mooney, and S. Grijalva, “CASH: Lightweight Keyed Hash Function Design for Reconfigurable Hardware,” 2026 IEEE International Symposium on Hardware-Oriented Security and Trust, Washington, D.C., 2026

© Vincent John Mooney III, 2026

Outline

- Introduction and Motivation
- Background
    - Feedback Registers
    - Product Registers
    - The Sponge Construction
    - Intel FPGA
    - Selected Baselines
- Product Registers on FPGAs
- Hardware-Embedded Keys using CMPRs
- Hardware Implementation
- Conclusion
- References
- Appendix

Appendix: Additional Experimental Results

Ascon:

- Registers: 597
- LUTs: 328

CASH:

- Registers: 214
- LUTs: 381

© Vincent John Mooney III, 2026