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
† School of Cybersecurity and Privacy
Georgia Institute of Technology
Atlanta, Georgia

Georgia Tech

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

Georgia Tech

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

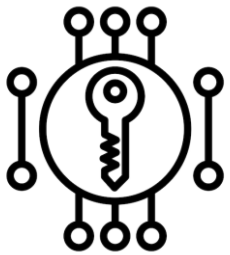
Outline

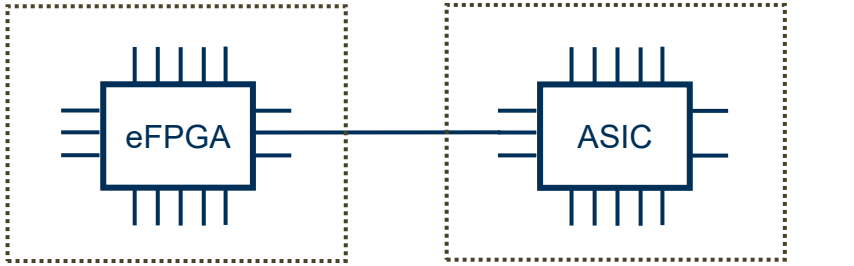
- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

Introduction and Motivation

- Cryptographic hash functions are essential for data integrity and entity authentication
- Message authentication codes (MACs), or keyed hash functions, play an essential role in entity authentication
- Growing need for cryptography in constrained environments:
  - Embedded systems
  - Trusted platform modules (TPMs)
  - Dedicated cryptoprocessors
- **Emerging Area of Interest:** reconfigurable hardware for security (for example, the use embedded FPGAs in the design of hardware security primitives) [9, 10]





Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

Background: Feedback Registers and Periodicity

- A feedback register is a register A such that on each clock cycle, its state $A[t]$ 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 $A[t]$
  - **NLFSRs:** f is a nonlinear function of $A[t]$
  - **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 $P(x)$
- For a Galois LFSR, the state updates according to:
  - $\boldsymbol{A[t+1] = xA[t] \; mod \; P(x)}$

Background: Feedback Registers and Periodicity

- A n 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

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, with limited exceptions (e.g., cross-joined pairs [8], which impose restrictions on the feedback function)**

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

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 $\deg(U(x)) \leq n - 1$

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

Background: Product Registers

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

- 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

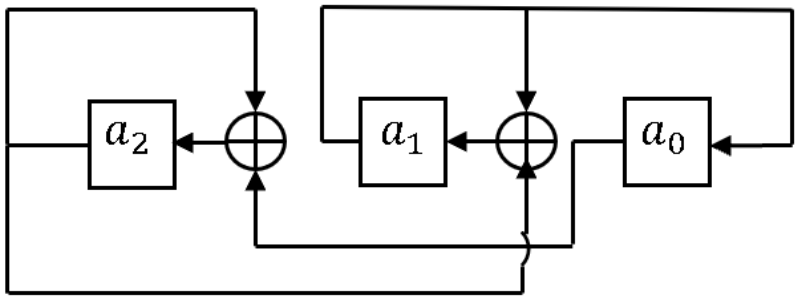
- We omit $U(x) = 0, 1$ from the set of "valid" update polynomials for a PR
  - => **An n-bit PR has $2^n - 2$ possible valid update polynomials**
- We consider Galois LFSRs to be a subset of PRs, namely, they are the special case where $U(x) = x$

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Background: Mersenne Product Registers

- A Mersenne Product Register (MPR) is a Product Register whose size is a Mersenne exponent
  - Mersenne exponent: Integer n (usually prime) 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
- From Definition 20 of [1], an n-bit MPR achieves period $2^n - 1$ provided $P(x)$ is primitive and $U(x) \neq 0, 1$
  - For an MPR, $\deg(P(x))$ is prime; thus, any irreducible (unfactorable) $P(x)$ is also primitive [11]



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

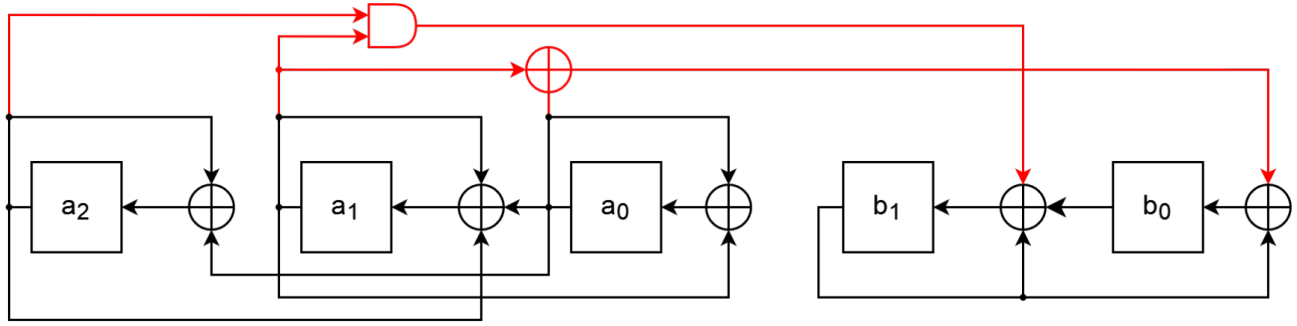
Background: Mersenne Product Registers

- 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 $2^3 - 1 = 7$
  - Fixed 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

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)

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

Selected Baselines

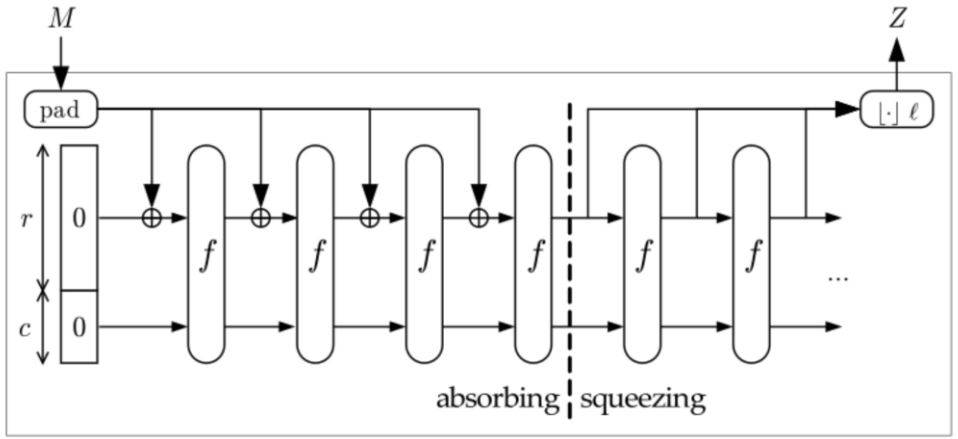
- Throughout this presentation, we will compare the security claims and hardware implementation efficiency of our work to the following finalist algorithms from the NIST Lightweight Cryptography Competition (LWC) (2018-2023):
  - Ascon [12] (NIST LWC Winner)
    - Sbox-based hash
  - Romulus [13]
  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

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Design Specification: CASH Architecture

- The CASH 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

Georgia Tech

Design Specification: CASH 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	$x + 1$	$x^2 + x + 1$

- Feedback polynomials 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

Design Specification: CASH Architecture

- Sponge Construction parameters:
  - $r = 64$
  - $c = 128$
- We base our security claim on the 128-bit key embedded within the update polynomials, not strictly the Sponge Construction security bounds
- CASH operates according to the *CASH permutation*, which is used with the Sponge Construction
- Notation:

Symbol	Meaning
S	Internal state of the CMPR
$S.\texttt{nextstate()}$	Next-state function of the CMPR
$S.\texttt{swap()}$	Swaps the upper and lower halves of S
$S.\texttt{permute()}$	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

Design Specification: CASH Architecture

- 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
```

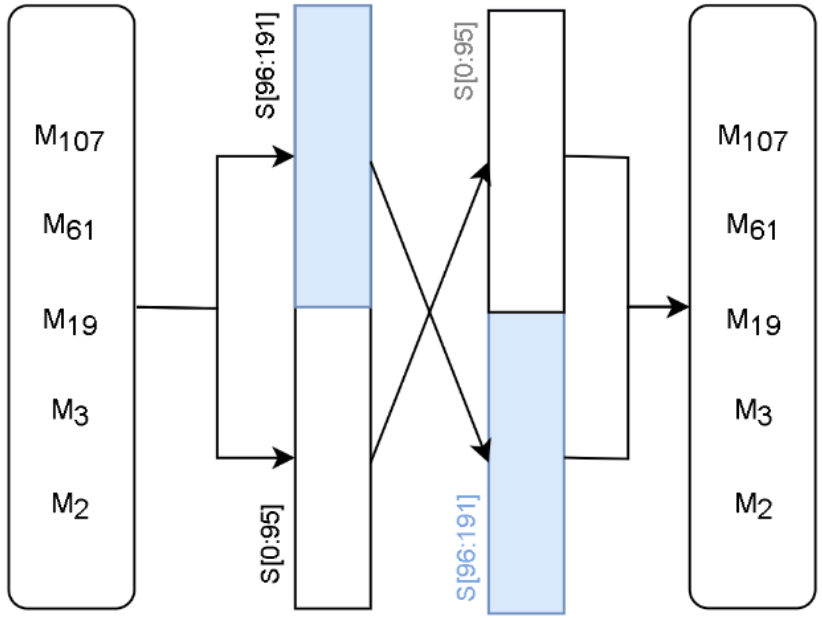


Fig. 2. CASH Permutation

Georgia Tech

Design Specification: CASH Architecture

- CASH Hash Generation:
  - 256-bit digest

Algorithm 2 CASH Algorithm

procedure INITIALIZE
  $S \leftarrow 1^{192}$
  $S.permute()$
end procedure
procedure ABSORB
  $M_0, \ldots, M_{j-1} \leftarrow M \| 1 \| 0^*$ ▷ Sponge Padding
  for $i \leftarrow 0 \ldots j-1$ **do**
    $S \leftarrow S \oplus M_i$
    $S.permute()$
  end for
end procedure
procedure SQUEEZE
  for $i \leftarrow 0 \ldots 3$ **do**
    $S.permute()$
    $H_i \leftarrow S(63), \ldots, S(0)$
  end for
end procedure
$H \leftarrow H_0 \| H_1 \| H_2 \| H_3$

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

Design Specification: Security Claims

- We present the security claims for CASH alongside our selected baselines from the NIST LWC:
  - Ascon (NIST LWC Winner)
  - Xoodyak
  - PHOTON-Beetle
  - Romulus

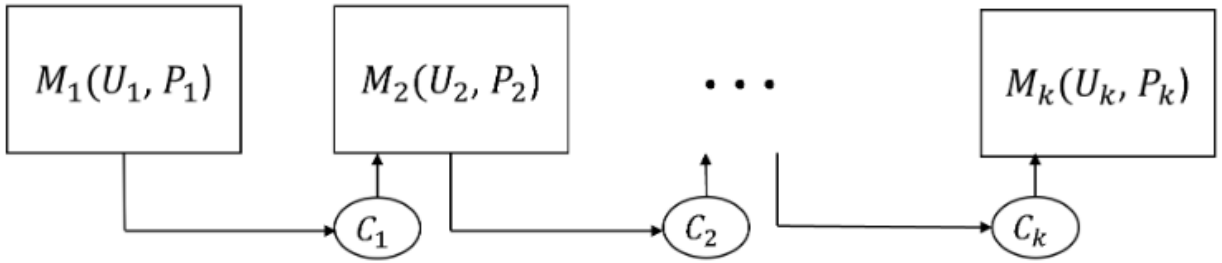
Design	Collision Resistance	(Second) Preimage Resistance
CASH	2^{128}	2^{128}
ASCON	2^{128}	2^{128}
Xoodyak	2^{128}	2^{128}
PHOTON-BEETLE	2^{112}	2^{128}
Romulus	2^{121}	2^{121}

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Security Analysis: CMPR Invertibility Theorem

- We provide a theorem on the invertibility of CMPRs to justify their use as permutations with the Sponge Construction
  - Sponge Construction security proofs assume f is bijective (one-to-one and onto)
  - Many proofs can still hold if f is a transformation (noninvertible), but most Sponge Construction literature assumes the availability of a pseudorandom permutation
- Consider the following generic CMPR, with k MPRs chained together:
  - Chaining functions only connect successive MPRs



- **Theorem:** Consider a CMPR composed of k MPRs $M_i, 1 \leq i \leq k$, where each MPR has its respective update polynomial $U_i, 1 \leq i \leq k$: $(M_1, U_1), \ldots (M_k, U_k)$. Let $C_i, 1 \leq i \leq k$, denote the chaining functions between each pair of MPRs. Moreover, let $x_i, 1 \leq i \leq k$ denote the state of the i-th MPR. Then, the CMPR state update equation $(x_1, \ldots, x_k) \rightarrow (U_1 x_1, \ldots, U_k x_k \oplus C_k(x_1, \ldots x_{k-1}))$ is invertible.

Security Analysis: CMPR Invertibility Theorem

- **Proof:** Since the state space of a CMPR is finite ($f : A \rightarrow A$), it suffices to show the CMPR state update is onto. Given a particular CMPR state $(y_1, \ldots, y_k)$, we give an explicit method to find a preimage state $(x_1, \ldots, x_k)$ which maps to $(y_1, \ldots, y_k)$ after one state update:

  - Since we require $U_i \neq 0, 1$ for an MPR and F_{2^n} is a field, then for each U_i we can find a modular inverse U_i^{-1} satisfying $U_i U_i^{-1} = id$.

  - Set $x_1 = U_1^{-1} y_1$, and inductively, for each $i \geq 2$, set the following:
  - $x_i = U_i^{-1}(y_i \oplus C_{i-1}(x_1, \ldots, x_{i-1}))$

  - By construction, the state $(x_1, \ldots, x_k)$ so defined satisfies the following:
  - $\left(U_1 x_1, U_2 x_2 \oplus C_1(x_1), \ldots, U_k x_k \oplus C_{k-1}(x_1, \ldots, x_{k-1})\right) = (y_1, \ldots, y_k)$

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

Security Analysis: Statistical Analysis I

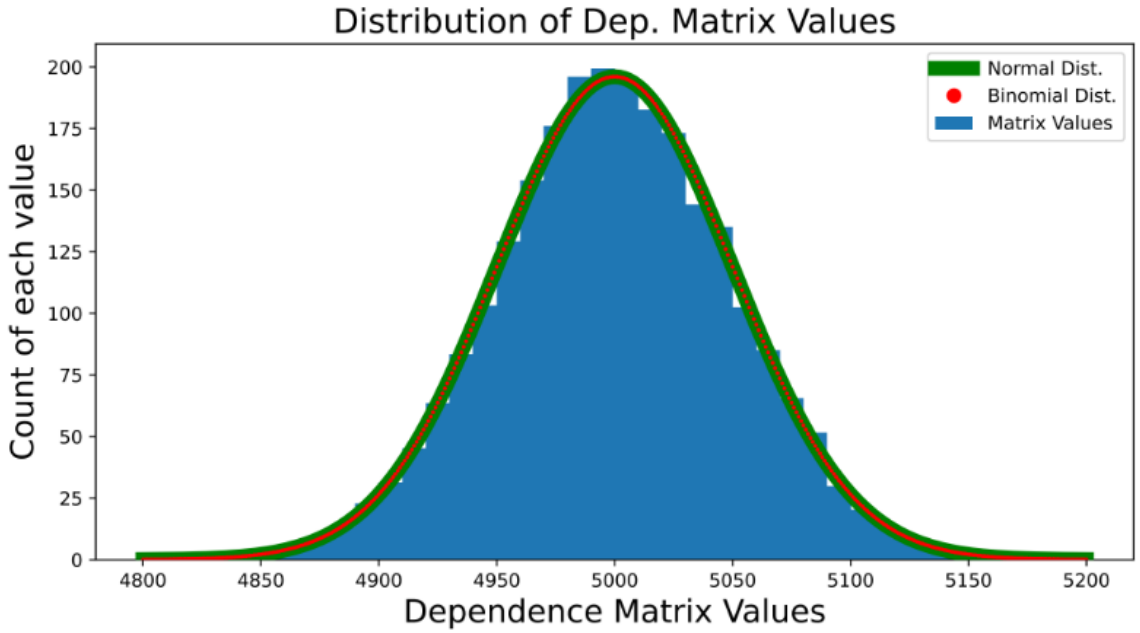
- NIST Statistical Test Suite (STS) [18]:
  - The NIST STS applies various statistical tests that examine the (pseudo)randomness of a bitstream independently, without taking into consideration the seed(s) used to generate the bitstream
  - 40,000 256-bit digests generated from 40,000 uniformly-sampled 128-bit keys and a constant 2048-bit message
  - Dataset Size: 10.24MB
  - All 15 tests within the NIST STS were passed
  - Secondary dataset generated using a constant, uniformly-sampled 128-bit key and 40,000 uniformly-sampled 2048-bit messages
  - All 15 tests within the NIST STS were passed by the secondary dataset as well

Security Analysis: Statistical Analysis II

- Bit Contribution Statistical Tests (B.C. Tests) [19]:
  - The B.C. tests are a class of tests that analyze both bitstreams and seeds to determine the existence of a statistical relationship between the two
  - Seeds (inputs) are flipped bit-by-bit and outputs are compared, with the expectation that ~50% of output bits should be affected by a one-bit flip in the input
  - Equivalently, the B.C. tests check for diffusion, or the effects of single-bit differentials applied to the input
  - The output of the B.C. tests is a dependence matrix D, where each entry D_{ij} represents the number of times out of N trials that a bit flip in bit i of the input causes a bit flip in bit j of the output
  - For a good cryptographic primitive, the D_{ij} should be normally distributed [19]

Security Analysis: Statistical Analysis II

- Bit Contribution Statistical Tests (B.C. Tests) [19]:
  - $N = 100$ trials
  - Each trial consists of 128 queries to the CASH permutation, using 128 bit-flipped variants of a 128-bit key, and then comparing differences in the 192-bit permutation output
  - Results indicate strong diffusion and sensitivity of the permutation to bit flips in the key



Georgia Tech

Outline

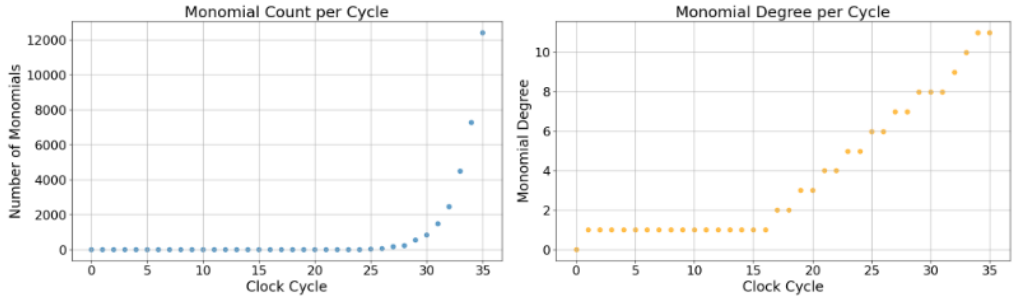
- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Security Analysis: Differential Cryptanalysis

- Differential cryptanalysis [20] seeks to uncover patterns in how differences between pairs of inputs affect resulting differences in the outputs of a cryptographic algorithm
- For systems based on feedback registers, conditional differential cryptanalysis [21] is the primary concern
  - Specialized methodology for imposing conditions on public and secret input variables in the initial state of a register
  - Demonstrated success against NLFSR-based algorithms
- We argue the resilience of CMPRs to (conditional) differential cryptanalysis based on the polynomial representation of CMPR I/O versus NLFSR I/O

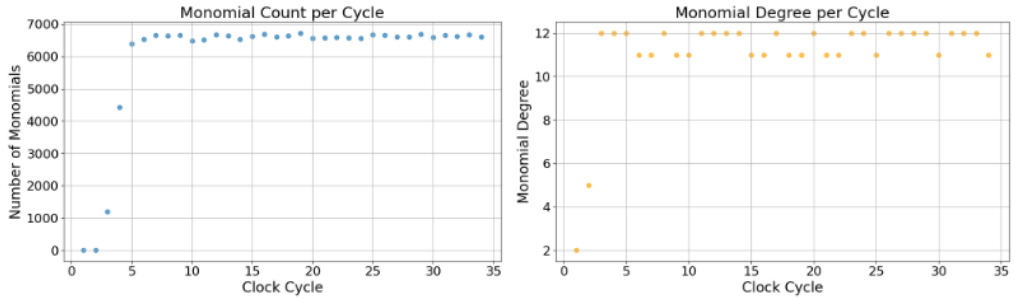
Security Analysis: Differential Cryptanalysis

- We investigate polynomial I/O representations for the following cases:
  - LSB of a 17-bit CMPR (from [1]) across 35 clock cycles
  - LSB of a 17-bit NLFSR (from [7]) across 35 clock cycles
  - Both registers seeded to a uniformly-chosen 17-bit value
- NLFSR case:
  - Slow growth in monomial count and degree
  - No change in monomial count until LSB is processed by feedback polynomial
  - Possible to track differentials in the early rounds using the methods of [21]
- CMPR case:
  - Rapid growth and saturation in monomial count and degree
  - Unclear how to track differentials since the round-to-round polynomial representation changes quickly



(a) Monomial Count (b) Monomial Degree

Fig. 5. Monomial Count and Degree for the LSB of a 17-bit NLFSR



(a) Monomial Count (b) Monomial Degree

Fig. 6. Monomial Count and Degree for the LSB of a 17-bit CMPR

Georgia Tech

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

Hardware Implementation

- Synthesized the following designs onto the Intel DE10-Standard [22] FPGA board:
  - CASH
  - Ascon
  - Romulus
  - Xoodyak
  - PHOTON-Beetle
- Per the CASH specification, 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

Hardware Implementation

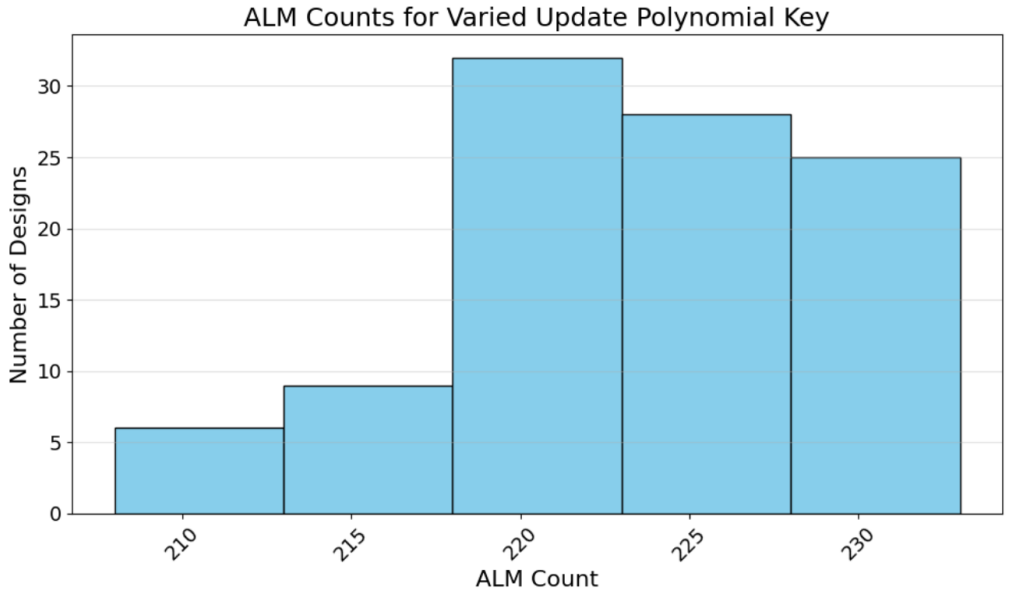
- FPGA synthesis comparison:
  - Adaptive Logic Modules (ALMs) are the combinational logic block in the Intel FPGA suite
  - η represents throughput-per-ALM
- CASH achieves between 44.5% and 75% lower FPGA utilization than the selected NIST LWC algorithms

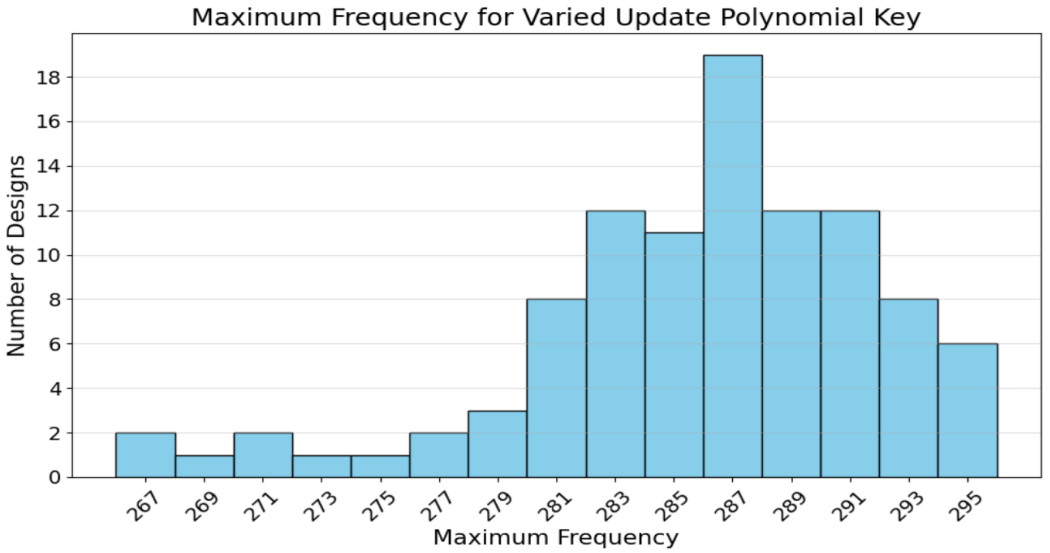
Design	ALMs	F_{max} (MHz)	Throughput (Mbps)
CASH	222	289	578.0
ASCON	400	239	1274.7
Xoodyak	853	218	2325.3
PHOTON-Beetle	892	242	645.3
Romulus	453	301	963.2

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

Hardware Implementation

- CASH area and clock speed variation:
  - To examine how the FPGA utilization and maximum frequency vary when CASH is re-keyed, we generate 100 uniformly-sampled key values and examine the ALM count and $\mathrm{F}_{\mathrm{max}}$ 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)**





Georgia Tech

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

Georgia Tech

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 and nonlinear chaining functions, we proposed a lightweight permutation based on a 192-bit internal state
- We provided a theorem on CMPR invertibility to justify the use of CMPRs as permutations with the Sponge Construction
- We investigated differential cryptanalysis of CMPRs, using both experimental results for the propagation of single-bit differentials (statistical testing) and analysis of the polynomial representation of CMPR I/O
- 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

Conclusion

- Future work on this topic includes:
  - 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 the amount of required clock cycles (and improve throughput)

Outline

- Introduction and Motivation
- Background
  - Feedback Registers
  - Product Registers
  - Mersenne Product Registers & Composite Mersenne Product Registers
  - Selected Baselines
- Design Specification
  - CASH Architecture
  - Security Claims
- Security Analysis
  - CMPR Invertibility Theorem
  - Statistical Analysis
  - Differential Cryptanalysis
- Hardware Implementation
- Conclusion
- References

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.

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.

References

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