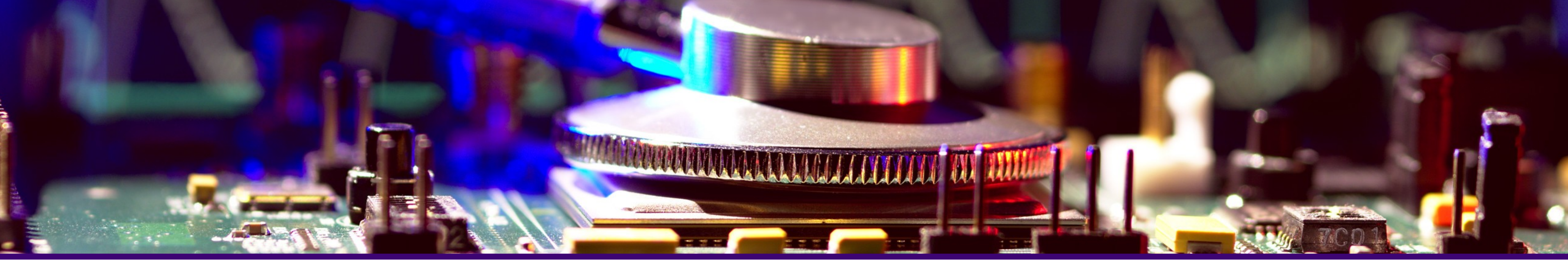


Side-Channel Resistant Implementations of Three Finalists of the NIST Lightweight Cryptography Standardization Process

Abubakr Abdulgadir[1,2], Richard Haeussler[1], Sammy Lin[1], Jens-Peter Kaps[1], and Kris Gaj[1]

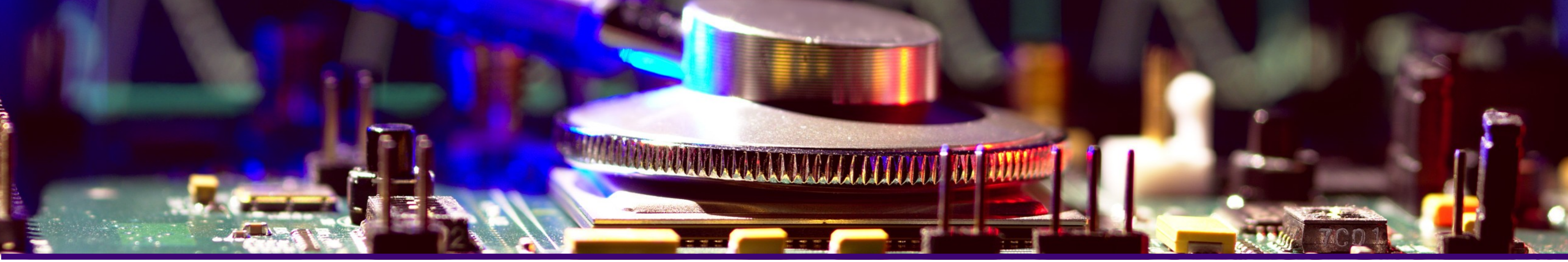
[1]George Mason University

[2]PQSecure Technologies

GEORGE MASON UNIVERSITY

CERG
cryptography.gmu.edu
Cryptographic Engineering Research Group

May - 2022

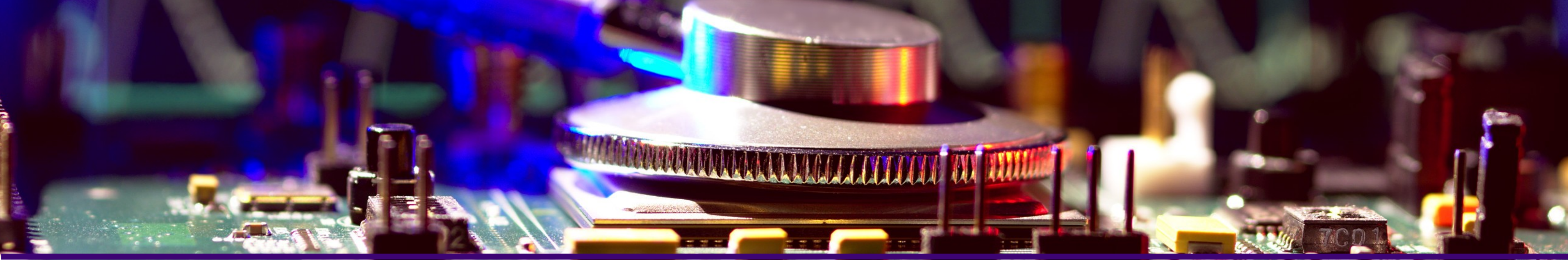


Overview

- Motivation
- Background
- Protected Implementations
- Results
- Conclusion

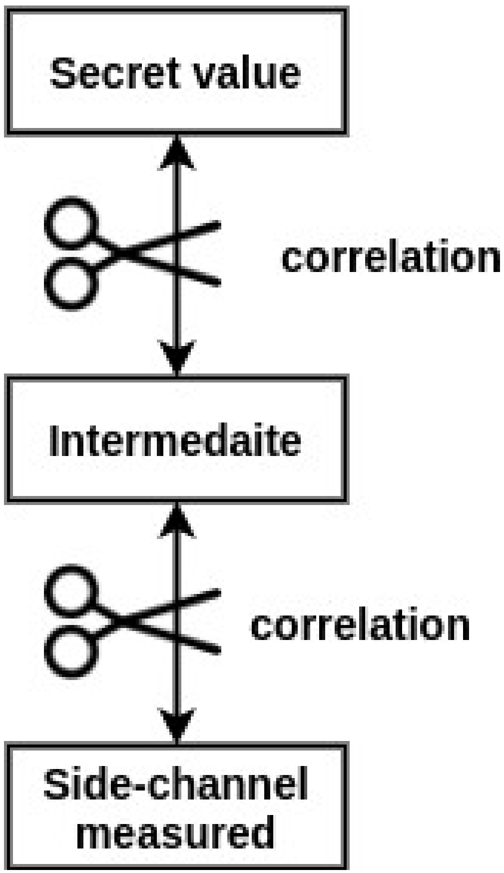
Motivation

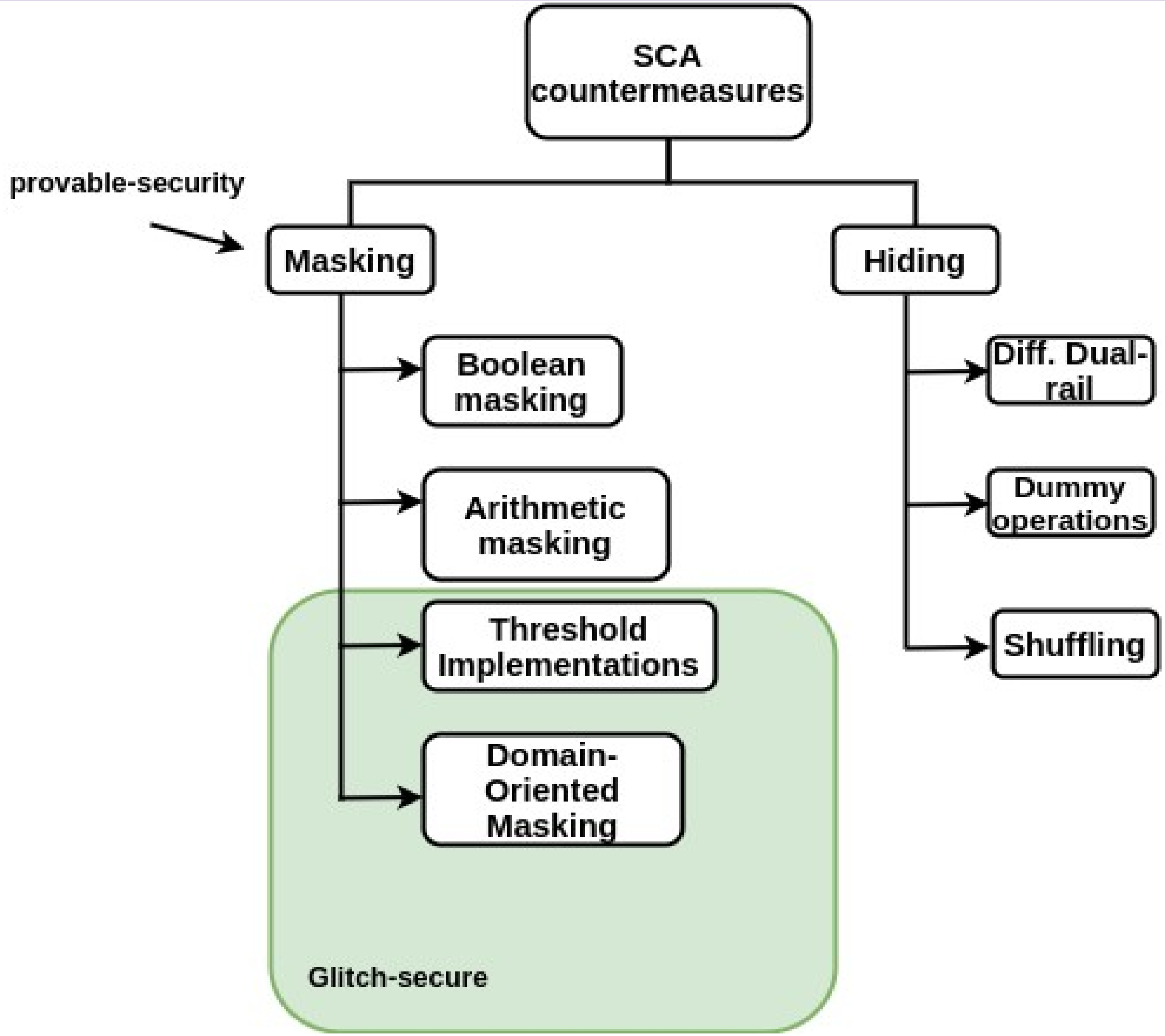
- NIST Lightweight Cryptography (LWC) Standardization Process
  - Round 1 (56 candidates) → Round 2 (32 candidates) → Ten finalists announced March 2021
- First rounds → Security, Software efficiency
- Final rounds → More interest in HW efficiency and **side-channel attack resistance**
- LWC is specially vulnerable to side-channel analysis due to limits on physical security



Background

SCA Countermeasures

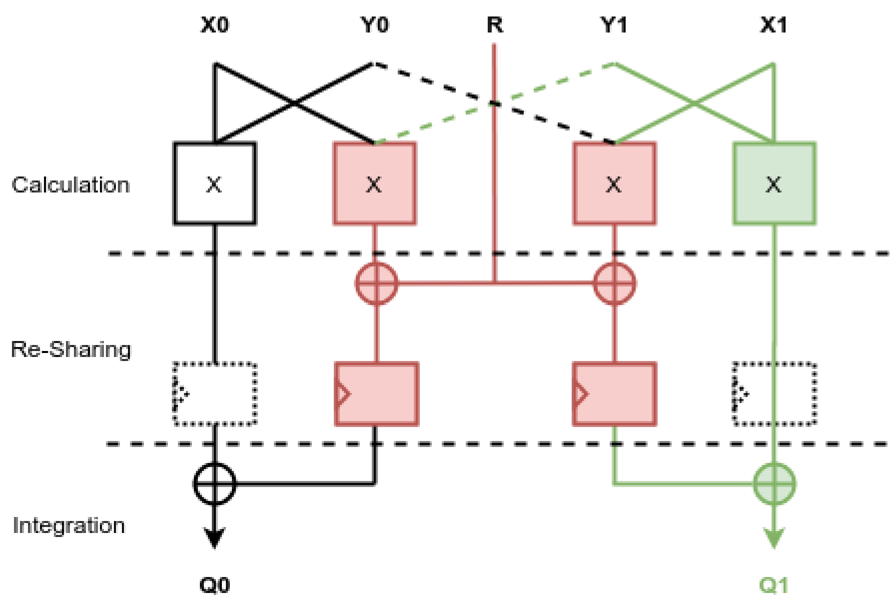




Masking:
- Split data into shares
X = X0 + X1

Domain-oriented Masking

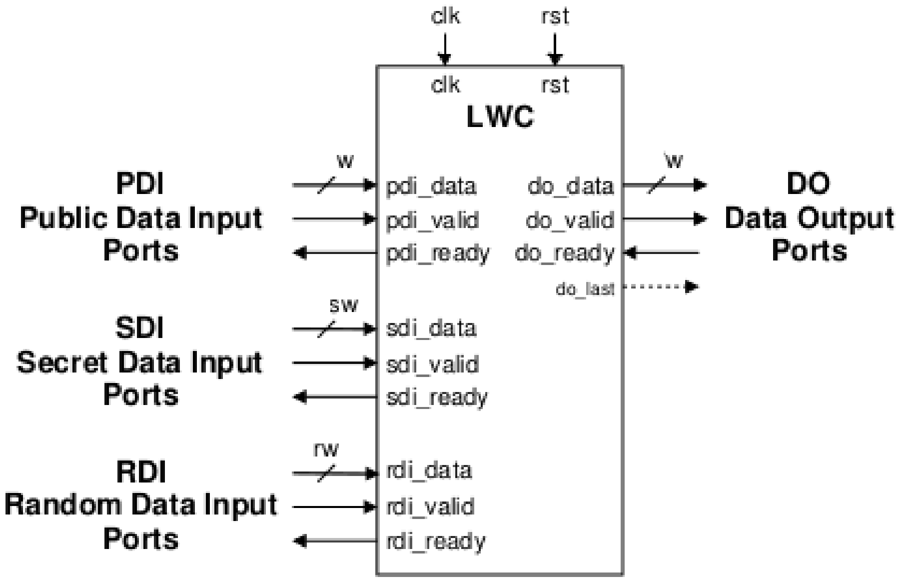
- Published by Gross et. al. In 2016
- Can be used for arbitrary order of protection
- Used d+1 shares for d-order protection



[Gross et al 2016]

Methodology

- All designs comply to the GMU LWC hardware API
- Shares are input serially and stored in SIPOs
- Output stored in a PISO
- Input shares generated in software



Lightweight Ciphers

- NIST LWC finalists studied
  - Elephant
  - TinyJAMBU
  - Xoodyak
- Domain-oriented Masking used for SCA-protection
- Compatible with the GMU LWC Hardware API
- Randomness generated using external PRNG

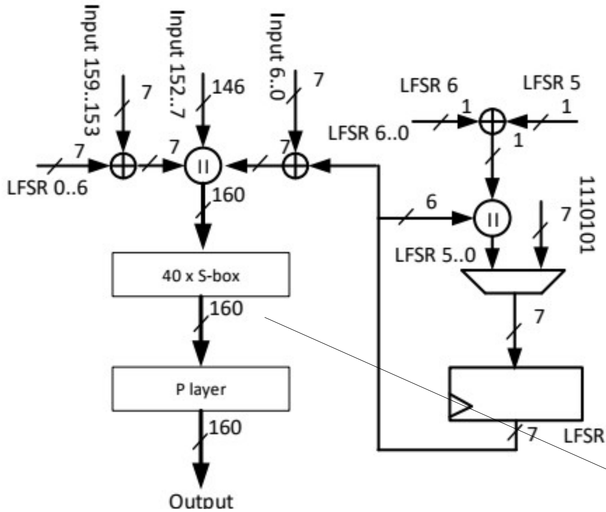
Elephant (1)

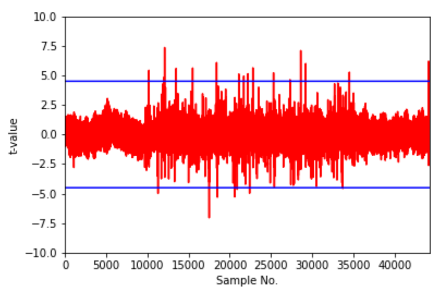
- 4-bit Sbox
- Converted to ANF
- Expression optimized to reduce the number of AND gates

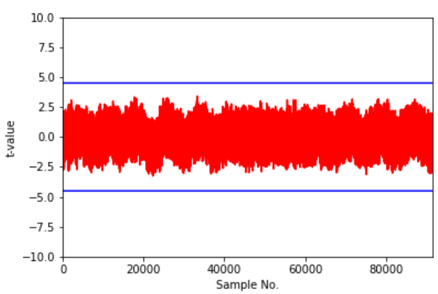
$$F[x, w, v, u] = [EDB0214F7A859C36]$$

$$F[0] = 0 + u + v + w \cdot v + x$$
$$F[1] = 1 + u + w \cdot v + x \cdot u + x \cdot v + x \cdot w + x \cdot w \cdot v$$
$$F[2] = 1 + v + w + x \cdot u + x \cdot w \cdot v$$
$$F[3] = 1 + v \cdot u + w + x + x \cdot u + x \cdot v + x \cdot v \cdot u + x \cdot w \cdot u$$

Elephant(2)







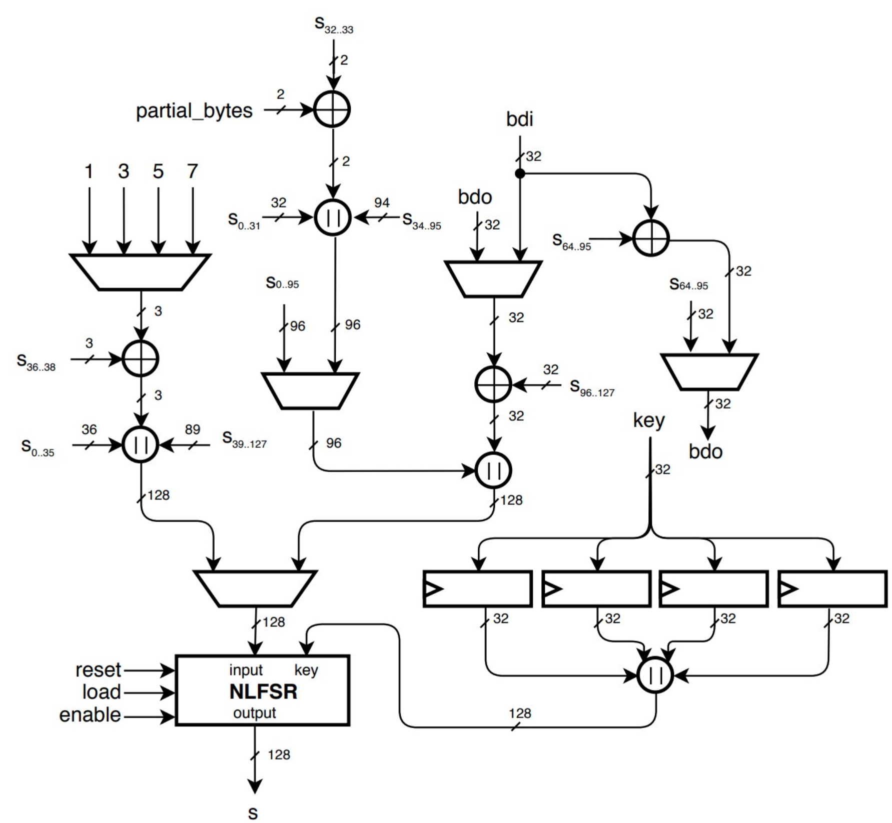
S-box

$$F[s3, s2, s1, s0] = [EDB0214F7A859C36]$$

$$\begin{aligned}
F[0] =& s0 + s3 + ((s1 + s2) \cdot s1)\mathbf{(2,1)}\\
F[1] =& not(s0 + s1 + ((s1 + s2) \cdot s1)\mathbf{(2,1)} +\\
& (s3 \cdot (s0 + s1) \cdot (s0 + s2))\mathbf{(3,1)} +\\
& ((s1 + s2) \cdot (s0 + s3) \cdot s3)\mathbf{(3,2)})\\
F[2] =& not(s1 + s2 + ((s1 + s2) \cdot (s1 + s3))\mathbf{(2,2)} +\\
& ((s1 + s2) \cdot s1)\mathbf{(2,1)} +\\
& (s3 \cdot (s0 + s1) \cdot (s0 + s2))\mathbf{(3,1)} +\\
& ((s1 + s2) \cdot (s0 + s3) \cdot s3)\mathbf{(3,2)})\\
F[3] =& not(s0 + s2 + s3 + ((s1 + s2) \cdot (s1 + s3))\mathbf{(2,2)} +\\
& ((s0 + s3) \cdot (s0 + s1))\mathbf{(2,3)} + ((s1 + s2) \cdot s1)\mathbf{(2,1)} +\\
& ((s1 + s2) \cdot (s0 + s3) \cdot s3)\mathbf{(3,2)});
\end{aligned}$$

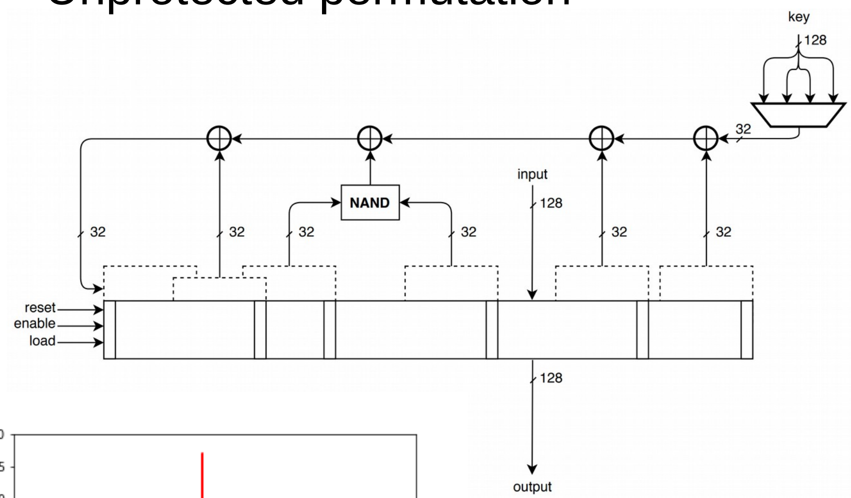
TinyJAMBU (1)

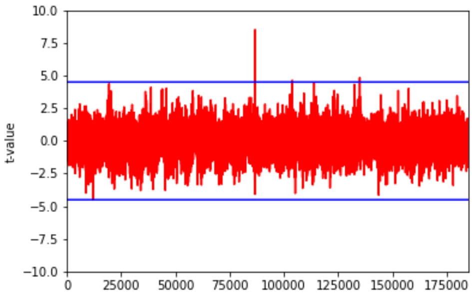
- NLFSR-based permutation
- AND gates used for non-linearity
- Utilized DOM AND gates in protected design

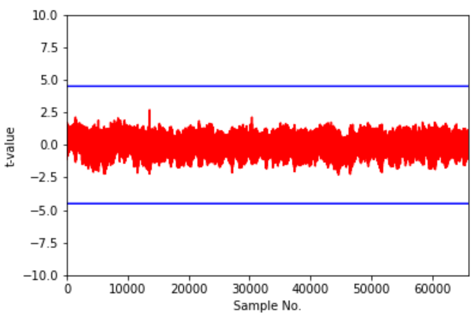


TinyJAMBU (2)

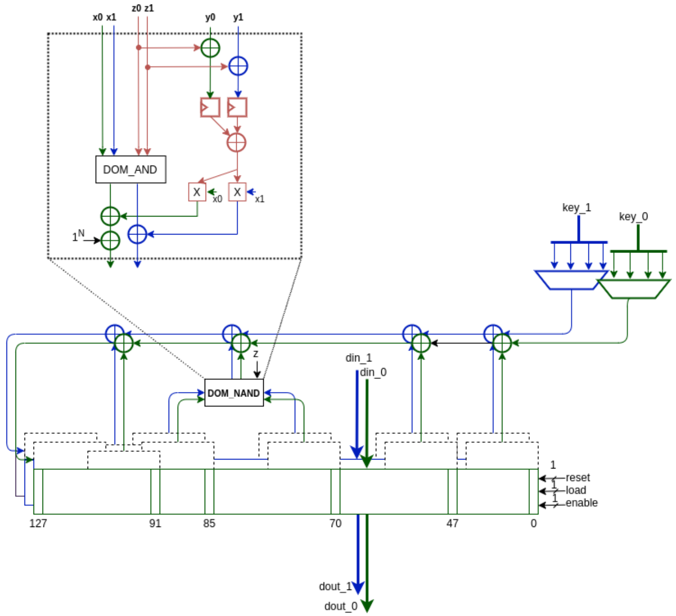
Unprotected permutation

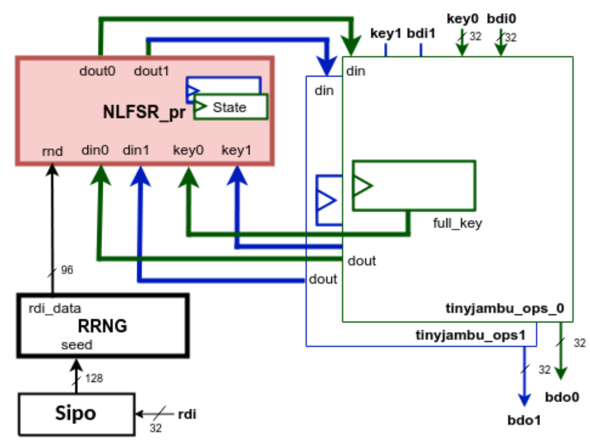






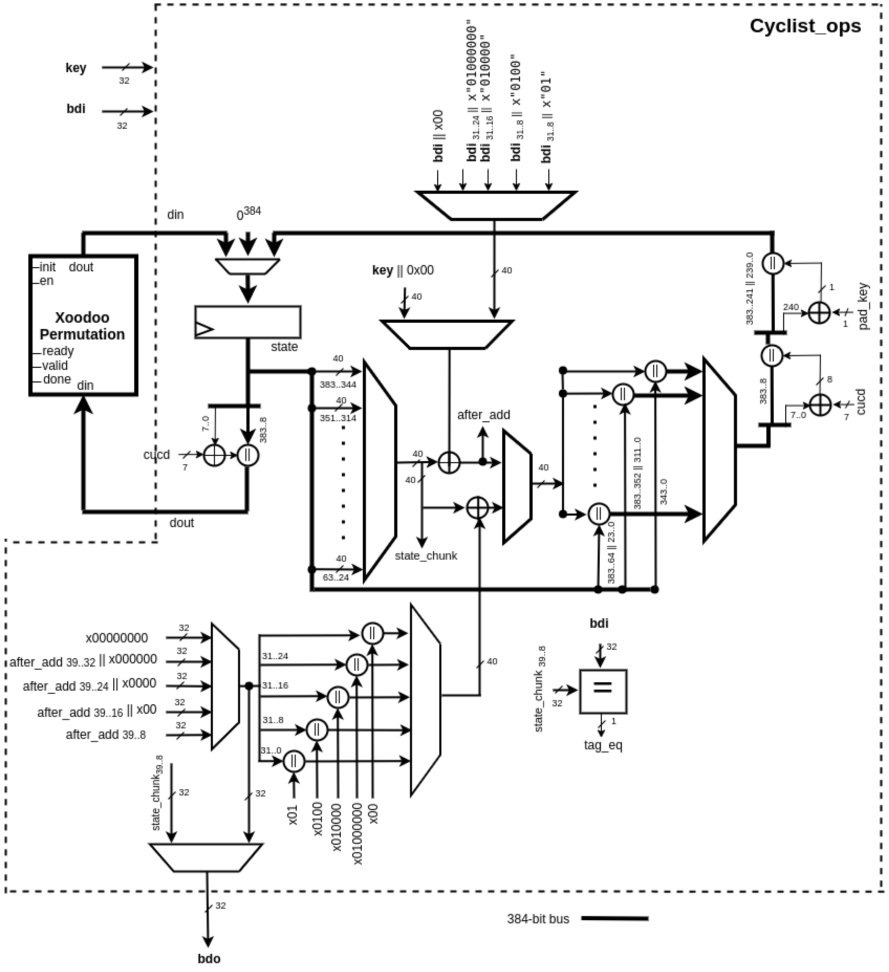
Protected permutation





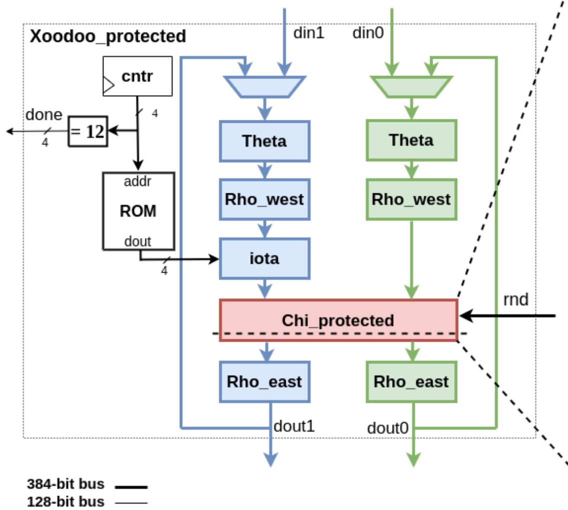
Xoodyak (1)

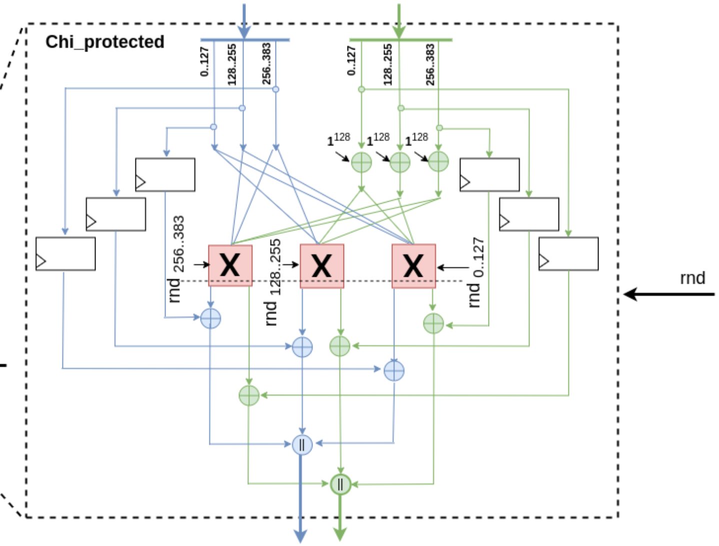
- Use Xoodoo permutation (Keccak-f inspired)
- Uses the χ operation for nonlinearity



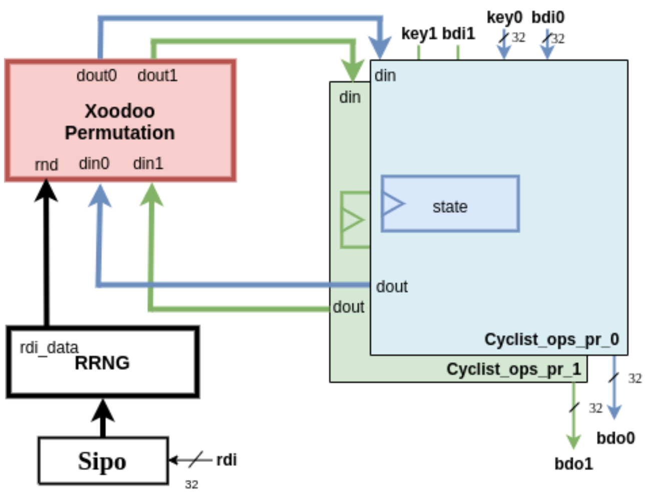
Xoodyak (2)

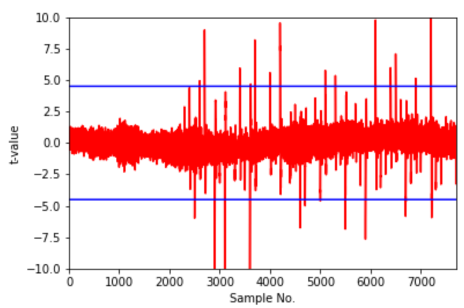
Protected permutation

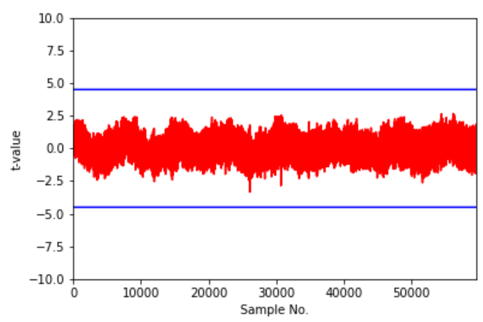




Full design

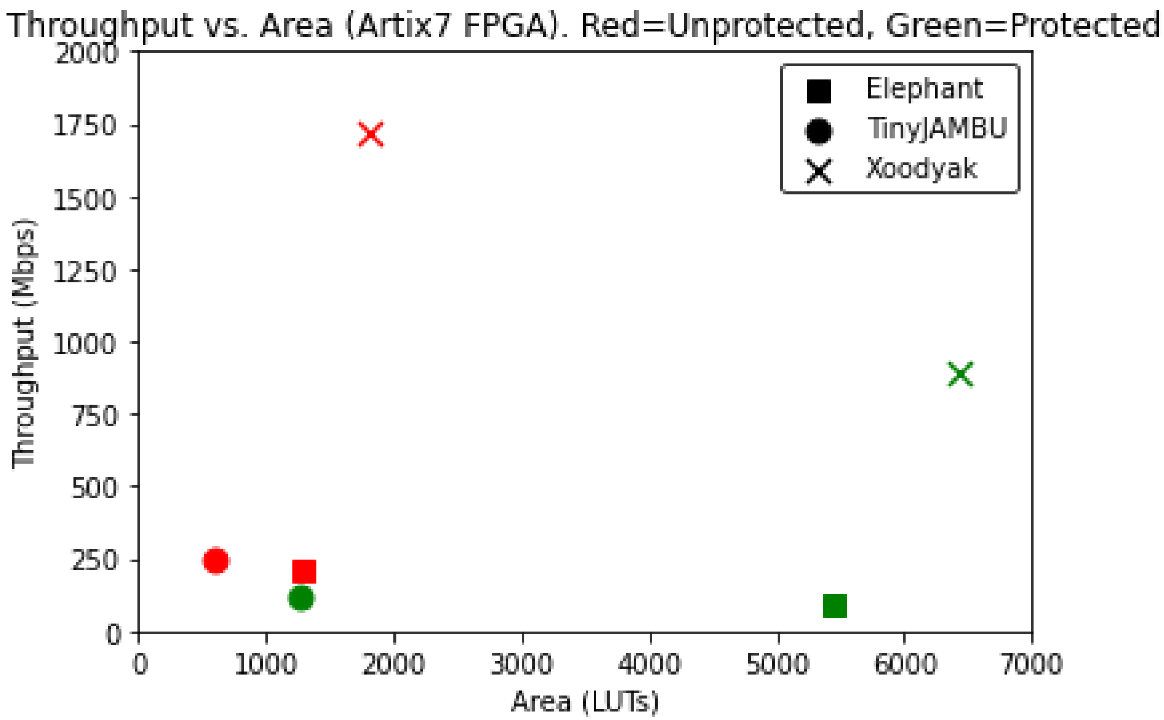






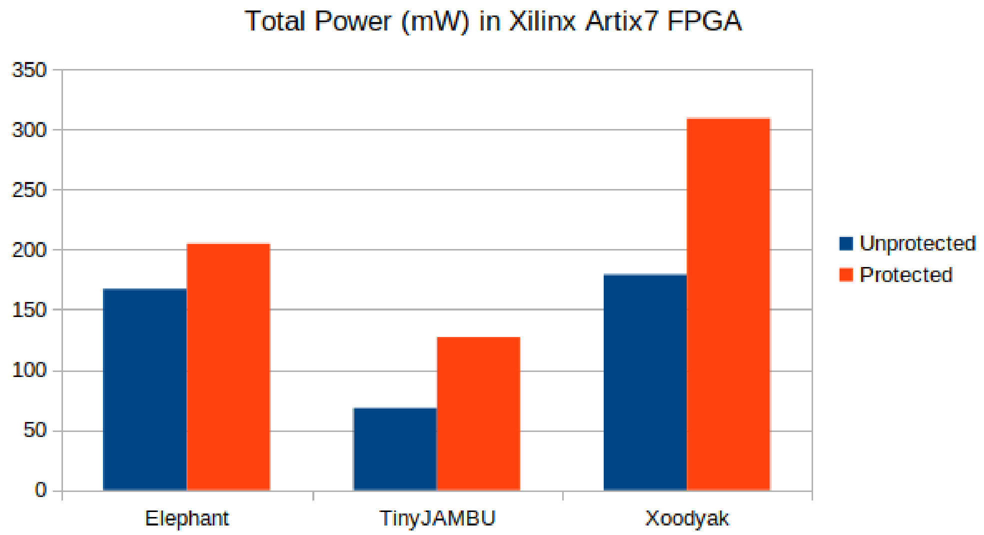
Comparison of Three LWC finalists (1)

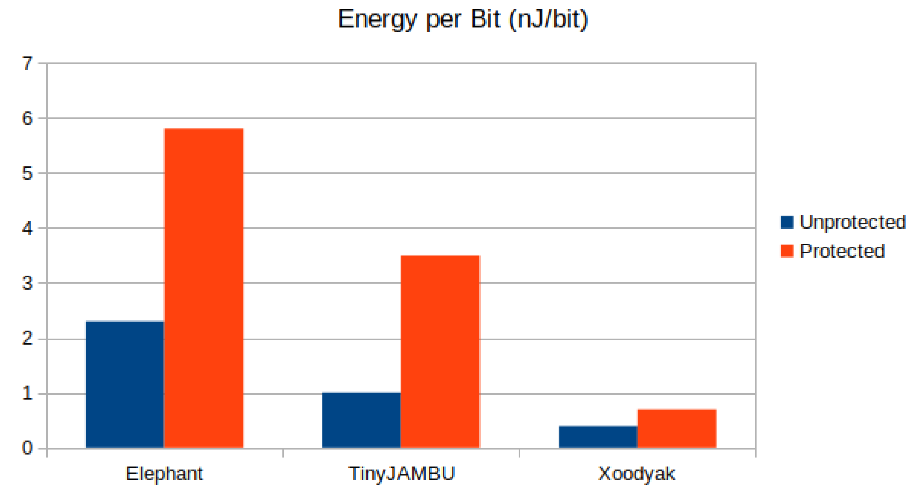
- Throughput vs. area



Comparison of Three LWC finalists (2)

- Power Measured using vector-based simulation (post-route)
- Xeda/Vivado were used to simulate/calculate power



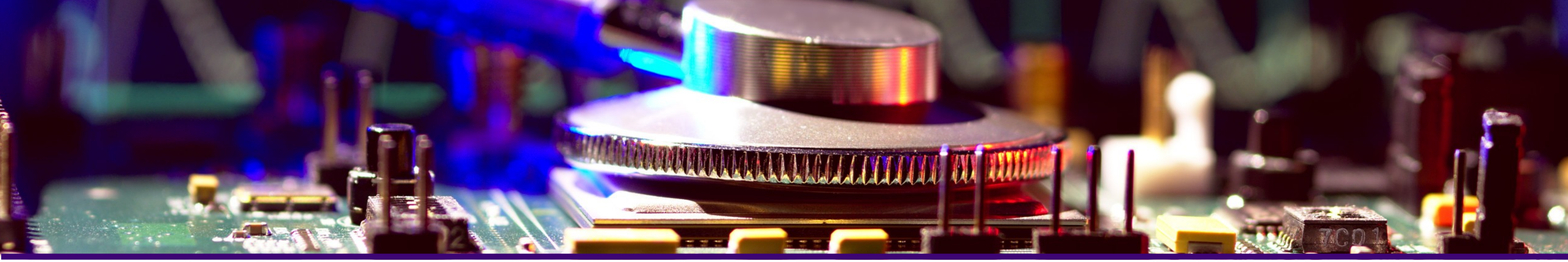


Conclusions

- We showed that among protected Elephant, TinyJAMBU and Xoodyak, our implementations of:
  - Xoodyak has the highest throughput an most energy efficient
  - TinyJAMBU is the most resource and power efficient

Future Work

- High-order designs
- Investigation of other protection methods



Thank you for listening

?