Self-Modifiable Reversible Circuits

Michael Stephen Fiske
Aemea Institute
mf@aemea.org

Abstract

Self-modifiable reversible circuits are introduced. The purpose of these circuits is twofold: first, reversible circuits can reduce the energy consumption in a digital computer by orders of magnitude. Second, self-modifiable reversible circuits provide computational advantages in cybersecurity and in AI. Our focus is on integrating quantum randomness into reversible circuits that implement cryptographic algorithms. These new circuits hinder Eve listening to the hardware computations because the physical locations of the bit computations are capable of moving randomly across different parts of the circuit. These circuits are likely to hinder Eve from capturing cryptographic keys even if she uses acoustic, electromagnetic, or power cryptanalysis. At the hardware level, these circuits also provide new obfuscation tools that help keep an algorithm private.

Keywords: bit, circuit, energy, finite field, obfuscate, quantum, random, reversible, self-modifiable, S-box

1. Introduction

In his foundational work, physicist Landauer (1961) observed that computers were currently implemented with physical devices (gates) that compute logical functions which are not 1-to-1 (*non-invertible*). Function $f : X \to Y$ is 1-to-1 if for all $x_1, x_2 \in X$ such that $x_1 \neq x_2$, then $f(x_1) \neq f(x_2)$. AND function $\wedge : \{0,1\}^2 \to \{0,1\}$ is non-invertible because $\wedge(1,0) = \wedge(0,1) = 0$.

Landauer posited that performing computation which is mathematically non-invertibile creates physical irreversibility and hence requires a minimal heat generation. Any non-invertible math operation executed in a physical device must increase entropy and produce heat. An irreversible AND gate (implemented with transistors) erases one bit of information, and generates heat. Physicist Bennett (1973) showed that it is theoretically possible to build computing machines that are logically reversible. Landauer and Bennett discovered a fundamental relationship between physics, information and computation. Self-modifiable reversible circuits extend reversible computing, and enable new cybersecurity applications.

1.1. Utility in Repair and Cryptography

Self-modifiable reversible computing enables a circuit or machine to repair itself. From a scientific perspective, self-modifiability introduces a new type of stability to the theory of dynamical systems. From a technological viewpoint, a hardware circuit can repair an error in its design. Applying self-modifiability to repair also can protect against various attacks by Mallory when a circuit is executing cryptographic instructions.

Unpredictable computation of algorithms can be achieved when randomness is integrated into a self-modifiable circuit. A self-modifiable circuit can execute the same sequence of cryptographic instructions across different gates and different physical locations on the circuit. Acoustic noise (Genkin et al., 2014), power (Kocher et al., 1999) and timing (Bonneau & Mironov, 2006) cryptanalysis, and electromagnetic eavesdropping (Quisquater & Samyde, 2001) were all used to extract cryptographic keys. Overall, unpredictably moving the execution of machine instructions across the circuit has the potential to substantially hinder these attacks.

1.2. Roadmap

Section 4 defines a universal reversible gate σ_3. Section 5 introduces self-modifiable circuits and shows how a σ_3 gate can swap the outputs of two subcircuits Σ_f and Σ_g. By routing the outputs of Σ_f and Σ_g to different physical locations in the circuit, computation can unpredictably move across a circuit. Section 6 motivates unpredictable circuit computation and discusses Kerckhoff's principle. Section 7 builds the AES S-box (NIST, 2001) with reversible circuits. Section 8 explains why quantum randomness is preferred. Section 9 obfuscates S-box circuits, by unpredictably moving the location of S-box computations.

URI: https://hdl.handle.net/10125/112248
978-0-9981331-9-5
(CC BY-NC-ND 4.0)

2. Related Work

"Conservative Logic" (Fredkin & Toffoli, 1982) is a foundational paper that proposes a new mathematical model of computation, based on fundamental principles of physics in its axioms. Using reversible gates, this model studies energy efficiency and performance of computing processes. The model does not integrate randomness into reversible gates, nor dynamically change how the computation is executed.

Yao (1982) presented secure multi-party computation[1], where Alice and Bob jointly compute a function on their private inputs without revealing those inputs to each other. In (Barak et al., 2010), the authors proved that it is impossible to obfuscate a program: the proof depends upon assuming that one-way functions exist.[2]

Quantum random number generators (QRNGs) rely upon the Heisenberg uncertainty principle (Heisenberg, 1927), and are used in various applications (Collantes & Escartin, 2017). To the best of our knowledge, randomness has not been combined with reversible circuits to provide advanced cryptographic functionality: for example, unpredictably moving the physical location of the hardware computation.

3. Math Notation and Definitions

$\{0,1\}^n$ is the set of all bit strings of length n. A Boolean function is $f : \{0,1\}^n \rightarrow \{0,1\}$. $|A|$ is the number of elements in set A. $|\{0,1\}^n| = 2^n$.

$A \oplus B$ equals A XOR B. $0 \oplus 0 = 1 \oplus 1 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$. Symbol $\neg : \{0,1\} \rightarrow \{0,1\}$ represents NOT. $\neg(0) = 1$ and $\neg(1) = 0$. Symbol $\vee : \{0,1\}^2 \rightarrow \{0,1\}$ represents OR. $\vee(1,1) = \vee(1,0) = \vee(0,1) = 1$ and $\vee(0,0) = 0$. Symbol $\wedge$ represents AND. $\wedge(0,0) = \wedge(1,0) = \wedge(0,1) = 0$ and $\wedge(1,1) = 1$.

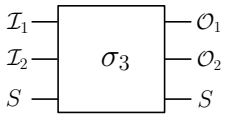
4. Reversible Gates

An n-bit *reversible gate* has n binary inputs and n binary outputs that computes a function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ such that $f(\{0,1\}^n) = \{0,1\}^n$. This means f is 1-to-1: every n-bit output of a reversible gate is reversible to a unique n-bit input. A 1-bit NOT gate is reversible because $0 \rightarrow 1$ and $1 \rightarrow 0$.

Fig. 1 shows a 3-bit reversible gate, named σ_3, that swaps inputs $\mathcal{I}_1$ and $\mathcal{I}_2$ when input $S = 0$. $\mathcal{I}_1$ and $\mathcal{I}_2$ are called σ_3's first and second input lines, respectively. S is the *swap input line*. Fig. 1 shows σ_3's logic. If input $S = 1$, no swapping occurs: $\mathcal{O}_1 = \mathcal{I}_1$ and $\mathcal{O}_2 = \mathcal{I}_2$. If $S = 0$, then output lines $\mathcal{O}_1 = \mathcal{I}_2$ and $\mathcal{O}_2 = \mathcal{I}_1$.

[1] Yao's protocols depend upon one-way functions.

[2] No one has mathematically proven that one-way functions exist.

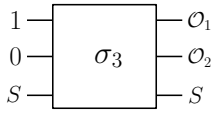


3-bit Reversible Gate Logic

$\mathcal{I}_1$	$\mathcal{I}_2$	S	$\mathcal{O}_1$	$\mathcal{O}_2$	S
0	0	0	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	1	1	0
0	0	1	0	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	1

Fig. 1. 3-bit Swap Gate σ_3

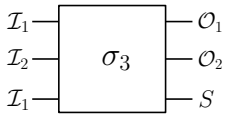
When input $\mathcal{I}_1 = 1$ and $\mathcal{I}_2 = 0$, Fig. 2 shows that reversible gate σ_3 computes a NOT gate: $\mathcal{O}_2 = \neg S$. When input line $S = \mathcal{I}_1$, Fig. 3 shows that gate σ_3 computes $\mathcal{O}_1 = \text{OR}(\mathcal{I}_1, \mathcal{I}_2)$ and $\mathcal{O}_2 = \text{AND}(\mathcal{I}_1, \mathcal{I}_2)$.



$\mathcal{O}_2 = \neg S$

$\mathcal{I}_1$	$\mathcal{I}_2$	S	$\mathcal{O}_1$	$\mathcal{O}_2$	S
1	0	0	0	1	0
1	0	1	1	0	1

Fig. 2. Gate σ_3 with $\mathcal{I}_1 = 1$ and $\mathcal{I}_2 = 0$



$\mathcal{O}_1 = \mathcal{I}_1 \vee \mathcal{I}_2$. $\mathcal{O}_2 = \mathcal{I}_1 \wedge \mathcal{I}_2$.

$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_1$	$\mathcal{O}_1$	$\mathcal{O}_2$	S
0	0	0	0	0	0
0	1	0	1	0	0
1	0	1	1	0	1
1	1	1	1	1	1

Fig. 3. Gate σ_3 with $S = I_1$

From the logic in Figs. 2 and 3, Fig. 4 shows a σ_3 gate circuit with outputs $\mathcal{O}_4 = \neg(\mathcal{I}_1 \vee \mathcal{I}_2)$ and $\mathcal{O}_2 = \neg(\mathcal{I}_1 \wedge \mathcal{I}_2)$. A gate is *universal* if a circuit composed of one or more gates can compute any function $f : \{0,1\}^n \rightarrow \{0,1\}$, where one output line computes $f(b_1, \ldots, b_n)$. NAND is universal, so σ_3 is universal.

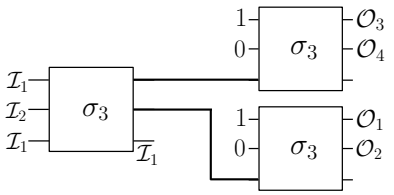


Fig. 4. $\mathcal{O}_4 = \neg(\mathcal{I}_2 \vee \mathcal{I}_1)$. $\mathcal{O}_2 = \neg(\mathcal{I}_2 \wedge \mathcal{I}_1)$

5. Self-Modifiable Circuits

A hardware circuit is called *self-modifiable* if the Boolean function computed at a gate output can be changed without changing the physical connections of the input and output lines and not adding, replacing or removing the electronic components (e.g., transistors).

Fig. 5 shows an example of a self-modifiable circuit built from four σ_3 reversible gates. By switching input S between 0 and 1, one Boolean function can replace another. For both cases $S = 0$ or $S = 1$, Fig. 3 indicates that input line 1 of the purple σ_3 gate receives $\mathcal{I}_1 \vee \mathcal{I}_2$ and input line 2 receives $\mathcal{I}_1 \wedge \mathcal{I}_2$.

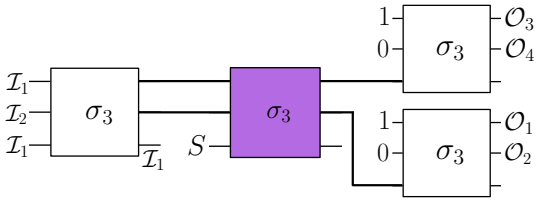


Fig. 5. A Self-Modifiable Circuit

If $S = 1$, then no swap occurs at the purple σ_3 gate. Hence, the circuit in Fig. 5 computes $\mathcal{O}_4 = \texttt{NOR}(\mathcal{I}_1, \mathcal{I}_2)$ and $\mathcal{O}_2 = \texttt{NAND}(\mathcal{I}_1, \mathcal{I}_2)$, as shown in Fig. 4.

If $S = 0$, then inputs $\mathcal{I}_1 \vee \mathcal{I}_2$ and $\mathcal{I}_1 \wedge \mathcal{I}_2$ into the purple σ_3 gate are swapped. Thus, the circuit in Fig. 5 computes $\mathcal{O}_4 = \texttt{NAND}(\mathcal{I}_1, \mathcal{I}_2)$ and $\mathcal{O}_2 = \texttt{NOR}(\mathcal{I}_1, \mathcal{I}_2)$.

When S flips from 1 to 0, the NOR function is replaced by the NAND function at output $\mathcal{O}_4$; and the NAND function is replaced by the NOR function at $\mathcal{O}_2$.

Fig. 5 demonstrates a reversible circuit that can be *dynamically reconfigured* without physically moving or severing the connections between the gates. Theorem 1 expresses the general case.

Theorem 1. Consider Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ and $g : \{0,1\}^n \rightarrow \{0,1\}$. There exists a reversible circuit composed of a finite number of σ_3 gates so that one part of the circuit named Σ_f, with output line $\mathcal{O}_1$, computes f and another part named Σ_g with output line $\mathcal{O}_2$ computes g. The circuit can self-modify so that $\mathcal{O}_1$ computes g and $\mathcal{O}_2$ computes f.

PROOF. Since σ_3 is a universal gate, a finite number of σ_3 gates can be composed as a subcircuit Σ_f so that one of the output lines computes f. Similarly, a finite number of σ_3 gates can be composed as a subcircuit Σ_g so that one of the output lines computes g.

In Fig. 6, a circuit adds one σ_3 gate to subcircuits Σ_f and Σ_g. σ_3's first input line receives the output from subcircuit Σ_f that computes f. σ_3's second input line receives the output from subcircuit Σ_g that computes g.

When $S = 1$, then output line $\mathcal{O}_1$ computes f and output line $\mathcal{O}_2$ computes g. If S flips from 1 to 0, then g's computation replaces f's computation at $\mathcal{O}_1$ and f's computation replaces g's computation at $\mathcal{O}_2$. ■

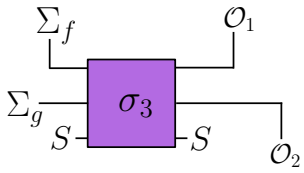


Fig. 6. A Self-Modifiable Circuit

By routing outputs $\mathcal{O}_1$ and $\mathcal{O}_2$ to different physical locations in the circuit, and integrating randomness, computation can unpredictably move across a circuit.

6. Motivation for Unpredictable Circuits

In section 9, self-modifiable reversible circuits unpredictably compute a critical part of AES (NIST, 2001). Self-modifiability and randomness are integrated into reversible circuits so that the execution of AES cryptography is more difficult to attack.

A fundamental idea is that every time AES executes with self-modifiable reversible circuits, the hardware always computes AES, yet how and where the gates perform the computation unpredictably changes. Since Eve cannot transparently see where and how the gates compute AES, Kerckhoff's principle (Kerckhoff, 1883) is violated. This is an advantage – not a disadvantage.

This advantage can be viewed through the lens of evolutionary biology. The Red Queen hypothesis (Van Valen, 1976) proposes that organisms constantly adapt and change to maintain their survival fitness in response to coevolving competitors, parasites, and predators.

The hypothesis is a primary explanation for the evolution of sexual reproduction (Langerhans, 2008). Some empirical studies (Clay & Kover, 1996; King et al., 2011) offer evidence that sexual reproduction provides an organism with greater resistance to infection due to variability in its offspring. Abstractly, offspring variability poses a new computational problem in each generation that parasites must overcome in order to infect the offspring.

During Kerckhoff's lifetime, there was no notion of machine instructions executing a cryptographic algorithm, nor Eve listening to and/or Mallory sabotaging the machine's instructions. Bellovin & Bush (2002) do not discuss uncompromised execution of machine instructions. The authors quote Kerckoff: "*Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconv'enient tomer entre les mains de l'ennemi.*" In (Bellovin & Bush, 2002, p. 1), their translation roughly states: "*the system must not require secrecy and can be stolen by the enemy without causing trouble.*"

A cryptographic algorithm's transparency has value so that the community can search for vulnerabilities and repair them if found. However, Kerckhoff's principle is applied too broadly because it does not value unpredictably executing in hardware the same algorithm each time it executes. Algorithmic transparency and execution predictability address different threat models.

Randomness in reversible circuits creates variability in the execution of the AES S-box. Acoustic and electromagnetic cryptanalysis are typically effective because Eve can obtain the machine instructions, and observe the order of their execution. Also, the timing of execution of the instructions is fixed for a particular processor.

Consequently, Eve can listen and Mallory can disrupt the execution (Valbuena, 2020) of a register machine's instructions (Hennessy & Patterson, 2012) due to structural instability (Fiske, 2020). This is why unpredictably executing cryptographic instructions enhances cybersecurity. Unpredictable execution can be applied to other types of algorithms too.

7. AES S-box Circuits

An S-box circuit is designed because its nonlinearity is a critical part of AES.[3] The S-box is more difficult to implement in a circuit than the affine operations computed in each AES round, and the affine operations can be built from XOR gates that implement the S-box.

One might ask why not use a lookup table? First, a table, based on finite fields $\mathbb{F}_{2^n}$, requires too much memory even for small values of n. $\mathbb{F}_{2^{16}}$ has 2^{16} distinct polynomials and each polynomial requires 16 bits. Set $\beta(n) = n2^n$. For $n = 16$, a table needs $\beta(16) = 1,048,576$ bits of memory. For $n = 32$ and $n = 64$, a table consumes $\beta(32) = 137,438,953,472$ bits, and $\beta(64) = 1,180,591,620,717,411,303,424$ bits.

Second, obfuscating even a small table with 256 elements requires far more gates in the circuit. Third, the obfuscation techniques described in section 9 are applicable to other cryptographic operations and other algorithms beyond cryptography because σ_3 is universal. Fourth, the irreducible polynomial is fixed when a table is used. If Alice and Bob execute a public key exchange, they can send extra bits that determine the irreducible polynomial, and hide the S-box from Eve.

Let $\mathbb{F}_2$ be the finite field with two elements 0 and 1. In $\mathbb{F}_2$, $\oplus$ (XOR) is addition, and $*$ (AND) is multiplication[4]. Polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$ is irreducible in the polynomial ring $\mathbb{F}_2[x]$. Let $\big(m(x)\big)$ be the ideal generated by $m(x)$ in $\mathbb{F}_2[x]$. Since $m(x)$ is irreducible, $\big(m(x)\big)$ is a maximal ideal in $\mathbb{F}_2[x]$, so $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$ is a field, and is isomorphic to $\mathbb{F}_{2^8}$.

A polynomial $\sum_{k=0}^{7} a_k x^k$ in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$ is expressed as an 8-bit array. $[0,1,0,1,0,1,1,1]$ represents $x^6+x^4+x^2+x+1$. Procedures 1-5 compute p^{-1}, when $p = [p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8]$ is non-zero.[5]

7.1. Procedures 1-6 Compute the S-box

Procedure 1. *Add Polynomials* $r(x) + w(x)$

```
function  add_polynomials(r, w)
  return  [ r1 ⊕ w1, r2 ⊕ w2, r3 ⊕ w3, r4 ⊕ w4,
            r5 ⊕ w5, r6 ⊕ w6, r7 ⊕ w7, r8 ⊕ w8 ]
end
```

Procedure 2. *Multiply Polynomial* $x * p(x)$

```
function  multiply_by_x(p)
   return  [p2, p3, p4, p5 ⊕ p1, p6 ⊕ p1, p7, p8 ⊕ p1, p1]
end
```

Procedure 3. *Multiply Polynomial* $x^n * p(x)$

```
function  multiply_by_x_to_n(p, n)
    r = p
    if  (n > 0)
       for  j  in  1 : n
          r = multiply_by_x(r)
       end
    end
    return  r
end
```

Procedure 4. *Multiply Polynomials* $p(x) * q(x)$

```
function  multiply(p, q)
    w = [0,0,0,0,0,0,0,0];      n = 0;      j = 8;
    while  (n ≤ 7)
       if  (q_j == 1)
          r = multiply_by_x_to_n(p, n)
          w = add_polynomials(w, r)
       end
       increment  n
       decrement  j
    end
    return  w
end
```

[3] AES cryptography requires a nonlinear operation; otherwise, it can be easily broken.

[4] Sometimes multiplication operator $*$ is omitted. For example, $x^2p(x)$ represents $x^2 * p(x)$, where $p(x)$ is a polynomial.

[5] Polynomial coefficient $a_k = p_{8-k}$ for $k = \{0, \ldots, 7\}$.

Example 1. Polynomials $p(x) = x^6 + x^4 + x$ and $q(x) = x^2+1$ are multiplied in the field $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$. Since the coefficients are added in $\mathbb{F}_2$, $x^n + x^n = 0$. A computation yields $p(x) * q(x) = (x^6 + x^4 + x) + (x^8 + x^6 + x^3) \mod m(x) = (x^8 + x^4 + x^3 + x) + (x^8 + x^4 + x^3 + x + 1) = 1$.

As bit arrays, $p = [p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8] = [0, 1, 0, 1, 0, 0, 1, 0]$ and $q = [q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8] = [0, 0, 0, 0, 0, 1, 0, 1]$.

Before carrying out a computation below, set $[w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8] = [1, 0, 1, 0, 0, 1, 0, 0]$ Procedure 4 calls procedure 3, which calls procedure 2:

$$\begin{aligned} x^2p(x) &= x * \big(x * p(x)\big) \\ &= x * [p_2, p_3, p_4, p_5 \oplus p_1, p_6 \oplus p_1, p_7, p_8 \oplus p_1, p_1] \\ &= x * [1, 0, 1, 0, 0, 1, 0, 0] \\ &= x * [w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8] \\ &= [w_2, w_3, w_4, w_5 \oplus w_1, w_6 \oplus w_1, w_7, w_8 \oplus w_1, w_1] \\ &= [0, 1, 0, 0 \oplus 1, 1 \oplus 1, 0, 0 \oplus 1, 1] \\ &= [0, 1, 0, 1, 0, 0, 1, 1]. \end{aligned}$$

Per procedure 3, $1 * p(x) = [0, 1, 0, 1, 0, 0, 1, 0]$. On the last step $p(x) * q(x) = [0, 1, 0, 1, 0, 0, 1, 0] \oplus [0, 1, 0, 1, 0, 0, 1, 1] = [0, 0, 0, 0, 0, 0, 0, 1]$, which corresponds to 1 as a polynomial. ■

For $a \neq 0$ in $\mathbb{F}_{2^8}$, $a^{255} = 1$ because the multiplicative group in $\mathbb{F}_{2^8}$ has 255 elements. Thus, $a^{-1} = a^{254}$. For any non-zero polynomial $p(x)$, then the multiplicative inverse of p is p^{254}, where $p^2 = p(x) * p(x)$, $p^3 = p(x)*p(x)*p(x)$, and so on. This is why procedure 5 correctly computes the inverse.

Procedure 5. *Compute the Inverse of $p(x)$ if $p(x) \neq 0$*

```
function mult_inverse(p)
   if (p == [0,0,0,0,0,0,0,0])
     return p
   end

   p4 = multiply(p2, p2)
   p5 = multiply(p4, p)
   p7 = multiply(p5, p2)
  p10 = multiply(p5, p5)
  p20 = multiply(p10, p10)
  p40 = multiply(p20, p20)
  p80 = multiply(p40, p40)
  p120 = multiply(p80, p40)
  p127 = multiply(p120, p7)
  p254 = multiply(p127, p127)

  return p254
end
```

For $a = (a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1)$ in $\mathbb{F}_{2^8}$, define affine map $A : \mathbb{F}_{2^8} \to \mathbb{F}_{2^8}$ as $A(a) =$

$$\begin{array}{ll} (a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_5, & a_2 \oplus a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus 1 \\ a_3 \oplus a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus 1, & a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus a_8 \\ a_1 \oplus a_5 \oplus a_6 \oplus a_7 \oplus a_8, & a_1 \oplus a_2 \oplus a_6 \oplus a_7 \oplus a_8 \\ a_1 \oplus a_2 \oplus a_3 \oplus a_7 \oplus a_8 \oplus 1, & a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_8 \oplus 1). \end{array}$$

Procedure 6 shows an implementation of A.

Procedure 6. *Affine Map A Applied to Inverse a*

```
function affine(a)
   b1 = a1 ⊕ a2 ⊕ a3 ⊕ a4 ⊕ a5 ⊕ 0
   b2 = a2 ⊕ a3 ⊕ a4 ⊕ a5 ⊕ a6 ⊕ 1
   b3 = a3 ⊕ a4 ⊕ a5 ⊕ a6 ⊕ a7 ⊕ 1
   b4 = a4 ⊕ a5 ⊕ a6 ⊕ a7 ⊕ a8 ⊕ 0
   b5 = a1 ⊕ a5 ⊕ a6 ⊕ a7 ⊕ a8 ⊕ 0
   b6 = a1 ⊕ a2 ⊕ a6 ⊕ a7 ⊕ a8 ⊕ 0
   b7 = a1 ⊕ a2 ⊕ a3 ⊕ a7 ⊕ a8 ⊕ 1
   b8 = a1 ⊕ a2 ⊕ a3 ⊕ a4 ⊕ a8 ⊕ 1
  return [b1, b2, b3, b4, b5, b6, b7, b8]
end
```

In procedure 6, argument a is the inverse element of p in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$ when $p \neq 0$. If $p = 0$, then $a = 0$ is passed into function `affine`. Thus, the S-box computation on p in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$ is computed as `affine( mult_inverse(`p`) )`.

Lastly, before designing reversible circuits that compute procedures 1-6, Julia source code was written to implement all six procedures. One test applies procedure 5 to all 255 non-zero elements in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$. For every element, procedure 5 computes the correct inverse. A second test combines procedures 5 and 6 and correctly computes the AES S-box table (NIST, 2001, page 16).

7.2. σ_3 Gate S-box Implementation

Procedures 1-6 use the XOR function. In Fig. 7, a circuit with three σ_3 gates computes $A \oplus B = (A \vee B) \wedge \neg(A \wedge B)$. Fig. 8 shows an XOR circuit, composed of three σ_3 gates in Fig. 7, that helps compute the S-box.

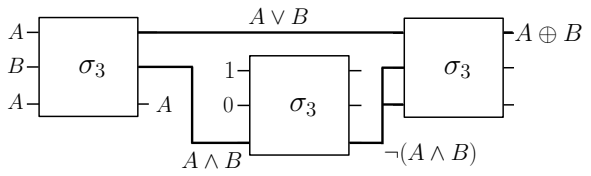


Fig. 7. Three σ_3 Gates Compute $A \oplus B$

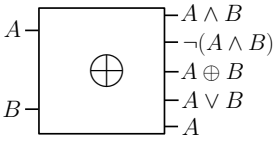


Fig. 8. Reversible XOR Circuit

Fig. 9 shows a reversible circuit,[6] built from three copies of the XOR circuit shown in Fig. 8. This circuit computes $x * p(x)$ in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$, by computing procedures 2 and 3 when $n = 1$.

[6] In Fig. 9, each XOR circuit is rotated 90 degrees clockwise relative to the circuit in Fig. 8.

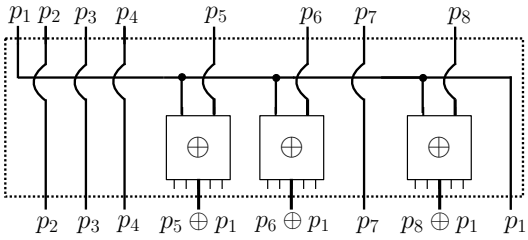


Fig. 9. Reversible Circuit Computes $xp(x)$

Fig. 10 shows a reversible circuit that computes $q(x) = x^2 * p(x)$ in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$, by computing procedure 3 when $n = 2$.

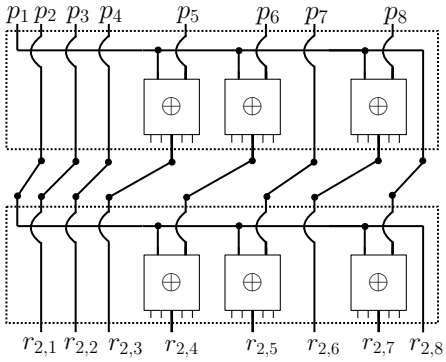


Fig. 10. Reversible Circuit $r(x) = x^2p(x)$

The circuit in Fig. 11 computes procedure 3 for $n = 1, 2, 3, 4, 5, 6, 7$. After the first layer, it computes $q(x) = xp(x)$ in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$. After the second layer, it computes $q(x) = x^2p(x)$ in $\mathbb{F}_2[x] \,/\, \big(m(x)\big)$. After the seventh layer, it computes $q(x) = x^7p(x)$. In binary, $[r_{7,1}, r_{7,2}, r_{7,3}, r_{7,4}, r_{7,5}, r_{7,6}, r_{7,7}, r_{7,8}]$ represents the computation of $x^7p(x)$.

In Fig. 12, a reversible circuit computes procedure `add_polynomials(r, w)`. In order to build procedure `multiply(p, q)`, with $q = [q_1, q_2, \ldots, q_8]$, the condition `if` $(q_j == 1)$ can be implemented with a reversible binary `AND` function.

In the loop of procedure `multiply(p, q)`, when $n = 0$, $j = 8$; when $n = 1$, $j = 7$; and so on up to $n = 7$ and $j = 1$. Observe that $n + j = 8$ in the loop. When $n = 0$, the procedure computes $1 * p(x)$.

Set $s_0 = [p_1 \wedge q_8, p_2 \wedge q_8, \ldots, p_n \wedge q_8]$. The test $(q_8 == 1)$ is satisfied by executing `add_polynomial(s0, w)`, where initially $w = [0,0,0,0,\ 0,0,0,0]$. If q_8 is 0, then $s_0 = [0,0,0,0,\ 0,0,0,0]$ and adding s_0 to w remains at 0. If q_8 is 1, then $s_0 = [p_1, p_2, \ldots, p_8]$ and `add_polynomial(s0, w)` returns $[p_1, p_2, \ldots, p_8]$.

For each j such that $1 \le j \le 8$, set $s_j = [r_{j,1} \wedge q_{8-j}, r_{j,2} \wedge q_{8-j}, r_{j,3} \wedge q_{8-j}, r_{j,4} \wedge q_{8-j}, r_{j,5} \wedge q_{8-j}, r_{j,6} \wedge q_{8-j}, r_{j,7} \wedge q_{8-j}, r_{j,8} \wedge q_{8-j}]$. Hence, the circuit in Fig. 12 adds s_j to the current value of w, and computes the correct value at the jth layer of the circuit in Fig 11.

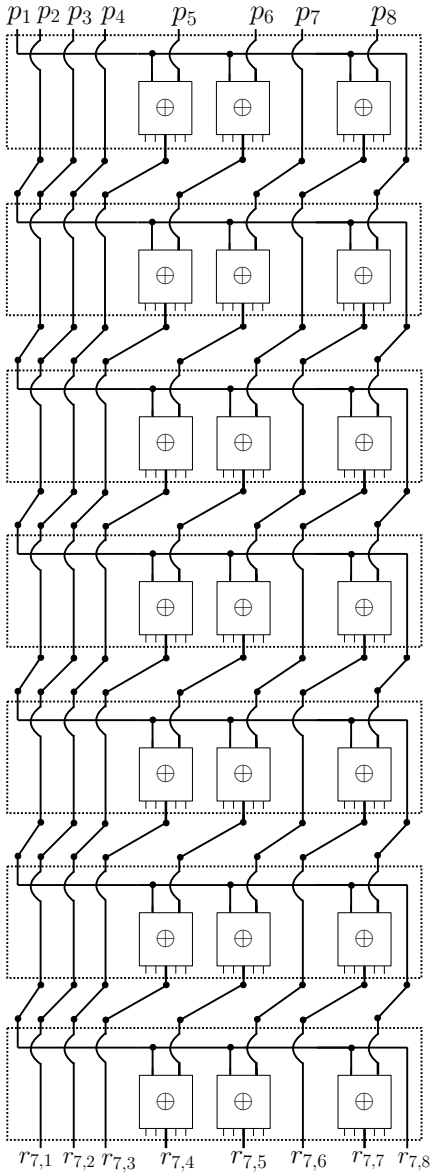


Fig. 11. Reversible Circuit $r(x) = x^7p(x)$

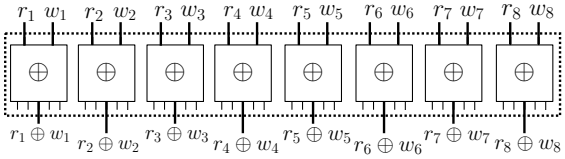
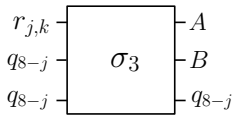


Fig. 12. Reversible Circuit Computes $r(x) + w(x)$

Output line 1 (labelled A) of the circuit in Fig. 13



$A = r_{j,k} \wedge q_{8-j}. \quad B = r_{j,k} \vee q_{8-j}.$

$r_{j,k}$	q_{8-j}	q_{8-j}	A	B	q_{8-j}
0	0	0	0	0	0
0	1	1	0	1	1
1	0	0	0	1	0
1	1	1	1	1	1

Fig. 13. Gate σ_3 with $S = q_{8-j}$

computes $r_{j,k} \wedge q_{8-j}$. Eight of these circuits handle the condition $q_j == 1$ in the loop.

Executing this computation in a reversible circuit, all the outputs after the jth layer $[r_{j,1}, r_{j,2}, r_{j,3}, r_{j,4}, r_{j,5}, r_{j,6}, r_{j,7}, r_{j,8}]$ are each sent to the top line of eight different σ_3 gates as shown in Fig. 14. q_{8-j} is sent to the second input line and input swap line of all eight σ_3 gates as shown in Fig 14.

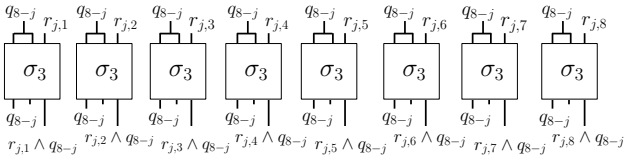


Fig. 14. If $(q_{8-j} == 1)$ Circuit

8. Quantum Randomness is Preferred

In section 9, a QRNG is integrated into self-modifiable circuits. Other sources of unpredictability can be used. However, quantum randomness is preferred because unpredictability is maximized based on a fundamental physical principle: *information cannot be captured before it exists.* Quantum theory states that outcomes are unpredictable until measurement occurs (Heisenberg, 1927; Colbeck & Renner 2011).[7] Empirically, QRNGs using photon detection (Stipčević & Rogina, 2007) have been rigorously tested on Diehard (Marsaglia, 1995) and NIST (Rukhin et al., 2010).

A pseudorandom number generator (PRNG) executes an algorithm with an input seed. Since the algorithm already exists, a PRNG violates the physical principle. If the obfuscation techniques in section 9 use a PRNG and Eve captures the seed, Eve can find the locations of the bit computations. A PRNG also begs the question: *How is the seed generated?*

Stipčević & Koç (2014) explain that other physical systems which are non-deterministic ultimately come from quantum effects, but randomness in these systems is difficult or impossible to prove. Non-deterministic random number generators can have pitfalls. For example, an attacker can predict "random" bits of a ring oscillator, using a frequency injection attack into the power supply (Markettos & Moore, 2009). The attack reduced the keyspace of a secure microcontroller from 2^{64} to 3300, and successfully compromised a chip and pin ATM payment card.

9. Moving Locations of Bit Computations

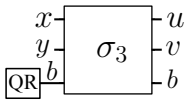


Fig. 15. Random Swap Circuit

Fig. 15 shows a random swap circuit that can unpredictably move the output bit of a computation. In Fig. 15, the rectangle labelled QR is a QRNG that generates random bits. If random bit $b = 1$, then $u = x$ and $v = y$. If random bit $b = 0$, then $u = y$ and $v = x$. Output u is sent to a different part of the S-box circuit than output v. If Eve listens to the electromagnetic noise emitted, this physical change can help hinder her eavesdropping.

The next circuit, shown in Fig. 16, is fundamental to physically moving the location of the relevant operations and also obfuscating the computation. This circuit combines a random swap circuit in Fig. 15 and an XOR circuit in Fig. 8.

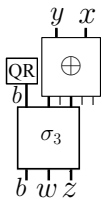


Fig. 16. Random XOR Circuit

The logic of the random XOR circuit operates as follows. If QR generates random bit $b = 1$, no swap occurs and outputs $w = x$ and $z = x \oplus y$. If random bit $b = 0$, then $w = x \oplus y$ and $z = x$. The random XOR circuit can augment the circuit in Fig. 11.

Fig. 17 shows a circuit that computes $x * p(x)$. This can be seen by observing in Fig. 8 that the 5th output line is A when the top line input is A. Hence, the output

[7] Heisenberg (1927) demonstrates fundamental measurement limitations for conjugate variables such as position and momentum. Colbeck & Renner (2011) show that no extension of standard quantum theory can predict measurement outcomes before they occur, nor improve predictive power.

lines of the circuit in Fig. 17 compute the same values as the corresponding output lines of the circuit in Fig. 9.

There are two additional points to consider. In Fig. 9, the outputs $p_5 \oplus p_1$, $p_6 \oplus p_1$, and $p_8 \oplus p_1$ are likely to take more time to stabilize than the outputs p_1, p_2, p_3, p_4, and p_7.

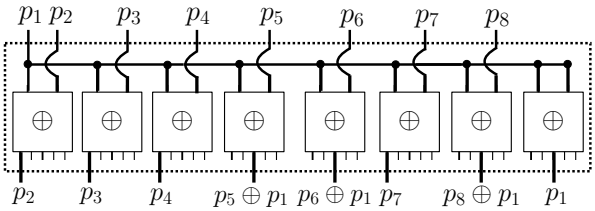


Fig. 17. Reversible Circuit Computes $x * p(x)$

In Fig. 17, the inputs $p_1, p_2 \ldots, p_8$ all are fed into the same circuit that is shown in Fig. 8. Due to symmetry, the outputs of the circuit in Fig. 17 should stabilize at the same time. This should make it more difficult for Eve to identify a voltage / time signature that can help Eve gain information about the computation. Second, the circuit in Fig. 17 is ready to augment with randomness in order to move the locations of the bit computations.

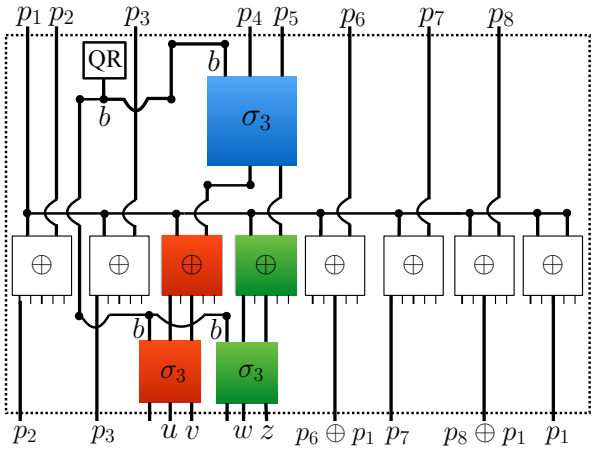


Fig. 18. Circuit $x * p(x)$ Swaps p_4 and p_5.

Fig. 18 shows a circuit that swaps inputs p_4 and p_5 when QR generates random bit $b = 0$. In Fig. 18, the circuit behavior is analyzed below in two cases: $b = 1$ and $b = 0$.

CASE $b = 1$. p_4 and p_5 are not swapped by the blue σ_3 gate. Hence, p_4 is sent into the right line of the red XOR circuit and $p_4 \oplus p_1$ is sent into the first line of the red σ_3 gate. Since $b = 1$ is sent to the swap input line of the red σ_3 gate, the output lines of Fig. 8 imply that $u = p_4$ and $v = p_4 \oplus p_1$.

Also p_5 is sent into the right line of the green XOR circuit and $p_5 \oplus p_1$ is sent into the first line of the green σ_3 gate. Since $b = 1$ is sent to the swap input line of the green σ_3 gate, the output lines of Fig. 8 imply that $w = p_5$ and $z = p_5 \oplus p_1$.

CASE $b = 0$. p_4 and p_5 are swapped by the blue σ_3 gate. Hence, p_5 is sent into the right line of the red XOR circuit and $p_5 \oplus p_1$ is sent into the first line of the red σ_3 gate. Since $b = 0$ is sent to the swap input line of the red σ_3 gate, the two inputs to the red σ_3 gate are swapped, so the output lines of Fig. 8 imply that $u = p_5 \oplus p_1$ and $v = p_5$.

Also p_4 is sent into the right line of the green XOR circuit and $p_4 \oplus p_1$ is sent into the first line of the green σ_3 gate. Since $b = 0$ is sent to the swap input line of the green σ_3 gate, the two inputs to the green σ_3 are swapped, so the output lines of Fig. 8 imply that $w = p_4 \oplus p_1$ and $z = p_4$

In both cases $b = 0$ and $b = 1$, the correct two bits of $x\,p(x)$ occur at the u and z output lines. The only difference is that when $b = 0$, the bits at u and z are in the wrong order. To deobfuscate the bits of $x\,p(x)$, then u and z must be swapped. Another σ_3 gate is added, where b is sent to the swap input line; z is sent to the first line of the σ_3 gate; and w is sent to the second line. This deobfuscation circuit is shown in Fig. 19.

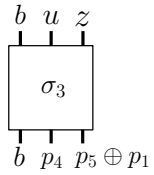


Fig. 19. Deobfuscating Circuit with u and z

Fig. 20 shows an alternative method of deobfuscation. In Fig. 18, the circuit's output lines v and w compute $v \oplus p_1$ and $w \oplus p_1$. If $b = 1$, then $v \oplus p_1 = p_4 \oplus p_1 \oplus p_1 = p_4 \oplus 0 = p_4$. Similarly, if $b = 1$ $w \oplus p_1 = p_5 \oplus p_1$. For the case $b = 1$, $v \oplus p_1$ equals u and $w \oplus p_1$ equals z.

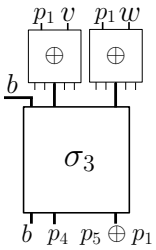


Fig. 20. Deobfuscating Circuit with v and w

If $b = 0$, then $v \oplus p_1 = p_5 \oplus p_1$. Hence, if $b = 0$, then $v \oplus p_1$ equals u. Similarly, if $b = 0$, then $w \oplus p_1 = p_4 \oplus p_1 \oplus p_1 = p_4$. Hence, $w \oplus p_1 = z$.

Fig. 21 shows the deobfuscation circuit in Fig. 19 combined with the circuit in Fig. 18. Fig. 22 shows the deobfuscation circuit in Fig. 20 combined with the circuit in Fig. 18. The computational equivalence of the circuits in Fig. 21 and Fig. 22 illustrates a design principle, expressed in Corollary 1. This principle is useful for designing reversible circuit obfuscation.

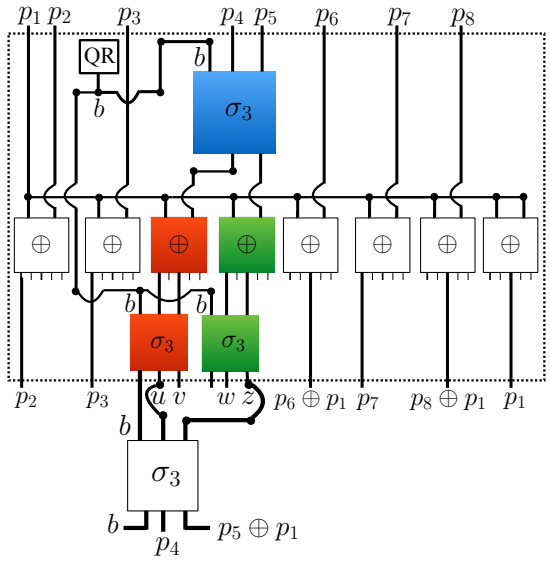


Fig. 21. Deobfuscated x * p(x) Circuit with u and z

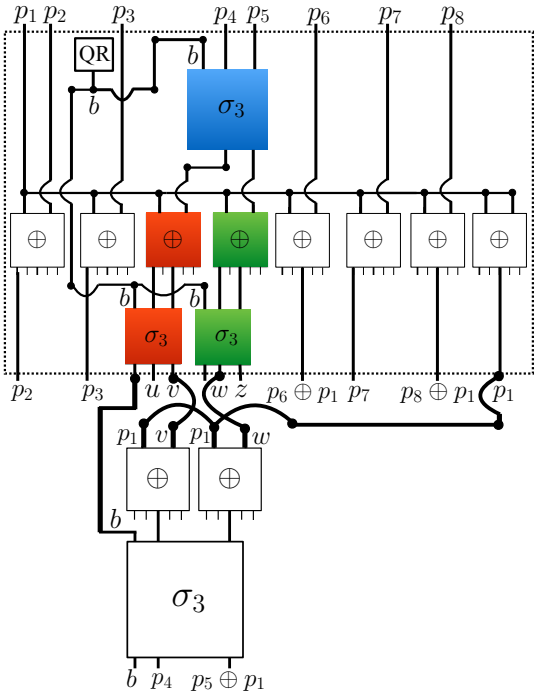


Fig. 22. Deobfuscated $x * p(x)$ Circuit with v and w

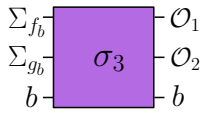


Fig. 23. Random Bit b: Outputs $\mathcal{O}_1 = \mathcal{O}_2$

Consider circuits Σ_{f_b} and Σ_{g_b} such that when $b = 1$, circuit Σ_{f_b} computes Boolean function $f_1 : \{0,1\}^n \to \{0,1\}$, and circuit Σ_{g_b} computes $g_1 : \{0,1\}^n \to \{0,1\}$; also, when $b = 0$, circuit Σ_{f_b} computes $f_0 : \{0,1\}^n \to \{0,1\}$ and circuit Σ_{g_b} computes $g_0 : \{0,1\}^n \to \{0,1\}$.

Circuits Σ_{f_b} and Σ_{g_b} are said to be *equivalent independent* of b if $f_0(a) = g_0(a)$ for all a in $\{0,1\}^n$ and $f_1(a) = g_1(a)$ for all a in $\{0,1\}^n$. This definition leads to corollary 1, which builds upon theorem 1.

Corollary 1. Suppose circuits Σ_{f_b} and Σ_{g_b} are equivalent independent of b. Then for $b = 0$ and $b = 1$, $\mathcal{O}_1 = \mathcal{O}_2$ in the circuit shown in Fig. 23. ■

Fig. 21 and Fig. 22 can represent special cases of corollary 1, by sending appropriate outputs from each circuit into additional σ_3 gates.

In Fig. 11, each circuit layer can be obfuscated with one or more swaps. Any permutation of a sequence of 8 bits requires at most 7 swaps. Procedure 7 generates a random permutation ρ on a sequence of n bits. For the AES S-box, $n = 8$. Each σ_3 gate swaps the first and second input lines when $S = 0$, so procedure 7 can be implemented in a circuit with σ_3 gates.

Procedure 7. *Random Permutation ρ*

```
Input: n
set ρ(1) := 1   set ρ(2) := 2   ...   set ρ(n) := n
set k := n
while k ≥ 2   {
    a QRNG helps randomly choose positive
    integer r in the set {1,...,k}
    set t := ρ(r)
    set ρ(r) := ρ(k)
    set ρ(k) := t
    decrement k by 1
}
Output: Permutation ρ on {1,...,n}
```

10. Summary

Self-modifiable reversible circuits were introduced. The AES S-box was implemented so that the part of the reversible circuit that computes the S-box changes its location without physically changing the hardware lines of the circuit. Theorem 1 and corollary 1 imply that the execution of any algorithm can be obfuscated with methods similar to the methods used in section 9. A hardware implementation is feasible, since the binary logic of a universal σ_3 gate can be built from transistors.

11. Future Research

Near term research should further develop obfuscation with random, reversible circuits. A hardware prototype should be developed and tested. To test the obfuscation's effectiveness, the electromagnetic signal emitted by the circuit should be measured to see if a voltage as a function of time signature exists while executing cryptographic operations.

Building on self-modifiable dynamical systems (Fiske, 2023; Fiske, 2024), long-term research should explore how reversible circuits can repair themselves either due to sabotage or design errors. Similar to error correcting codes (Hamming, 1950; Gallager, 1960), repair will require spare σ_3 gates in the original circuit; meta variables that detect errors; and meta operators that reconfigure gate connections. The self-modifiability of reversible circuits — where σ_3 gates implement meta computation that heals — provides a mechanism for dynamic hardware repair without replacing physical components. Overall, long-term research should strive to develop mathematical and physical systems whose complexity can increase over time.

Acknowledgments

Many thanks to the peer reviewers for their helpful comments, and to Mario Stipčević for generating data from his quantum random flip-flop (Stipčević, 2016) and providing his expertise on QRNGs. I am deeply grateful to Obi for inspiring me while writing this paper.

References

Boaz Barak, et. al. (2012, May). "On the (Im)possibility of Obfuscating Programs." *Journal of the ACM*. 59(2), 1–48.

Steven M. Bellovin & Randy Bush. (2002, August). "Security Through Obscurity Considered Dangerous." *IETF*. 1–4.

Charles H. Bennett. (1973, November). "Logical Reversibility of Computation." *IBM Journal of Research*, 525–532.

Joseph Bonneau & Ilya Mironov. (2006). "Cache-Collision Timing Attacks Against AES." *Cryptographic Hardware and Embedded Systems.* LNCS 4249. Springer, 201–215.

Keith Clay & Paula X. Kover. (1996, September). "The Red Queen Hypothesis and plant/pathogen interactions." *Annual Review of Phytopathology.* 34, 29–50.

Roger Colbeck & Renato Renner. (2011, August). "No extension of quantum theory can have improved predictive power." *Nature Communications*. 2:411, 1–5.

Miguel Herrero-Collantes, & Juan C. Garcia-Escartin. (2017, February 22). "Quantum random number generators." *Reviews of Modern Physics*. 89(1), 015004, APS.

Michael S. Fiske. (2020, January). "Toward a Mathematical Understanding of the Malware Problem." *Proc. of the 53rd Hawaii Int. Conf. on System Sciences*. 6705–6714.

Michael S. Fiske. (2023, January). "Dynamical Systems that Heal." *Proc. of the 56th Hawaii Int. Conf. on System Sciences*. 6685–6694.

Michael S. Fiske. (2024, January). "Stabilizing D Flip-Flop Orbits with Self-Modifiable Differential Equations." *Proc. of the 57th Hawaii Int. Conf. on System Sciences*. 7408–7417.

Edward Fredkin & Tommaso Toffoli. (1982, April). "Conservative Logic." *Int. J. of Theor. Physics*. 21(3-4), 219–253.

Robert Gallager. (1960, September). *Low Density Parity Check Codes* (Doctoral dissertation). MIT.

Daniel Genkin, Adi Shamir, & Eran Tromer. (2014, December). "RSA Key Extraction via Low Bandwidth Acoustic Cryptanalysis." *CRYPTO*, Springer, 444–461.

Richard W. Hamming. (1950, April). "Error Detecting and Error Correcting Codes." *Bell Technical J*. 29(2), 147–160.

Werner Heisenberg. (1927, März). "Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik." *Zeitschrift für Physik*. 43(3-4), 172–198.

John Hennessy & David Patterson. (2012). *Computer Architecture*. 5th Edition, Morgan Kaufmann.

Auguste Kerckhoff. (1883, Janvier). "La Cryptographie Militaire." *Journal des Sciences militaires*. Vol. IX, 5–38.

Kayla C. King, Jukka Jokela, & Curtis M. Lively. (2011, April). "Trematode parasites infect or die in snail hosts." *Biology Letters*. 7(2), 265–268.

Paul Kocher, Joshua Jaffe, & Benjamin Jun. (1999, December). "Differential Power Analysis." *Advances in Cryptology*. LNCS 1666. Springer, 388–397.

Rolf Landauer. (1961, July). "Irreversibility and Heat Generation in the Computing Process." *IBM Journal of Research*. 5(3), 183–191.

R. Brian Langerhans. (2008). "Coevolution." *Encyclopedia of Ecology*. Elsevier, 32–36.

A. Theodore Markettos & Simon W. Moore. (2009). "The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators." *Cryptographic Hardware and Embedded Systems*. LNCS 5747. Springer, 317–331.

George Marsaglia. (1995). "The Marsaglia Diehard Battery of Tests of Randomness." *Florida State University*.

NIST. (2001, November). "Advanced Encryption Standard."

Andrew Rukhin, et. al. (2010, April). "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications." NIST.

Jean-Jacques Quisquater & David Samyde. (2001, September). "Electromagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards." *Smart Card Programming and Security.* LNCS 2140. Springer, 200–210.

Mario Stipčević & B. Medved Rogina. (2007, April). "Quantum random number generator based on photonic emission in semiconductors." *Review of Scientific Instruments*. 78, 1–7.

Mario Stipčević & Çetin Kaya Koç. (2014, November). "True Random Number Generators." *Open Problems in Mathematics and Computational Science.* Springer, 275–315.

Mario Stipčević. (2016, March 16). "Quantum random flip-flop and its applications in random frequency synthesis and true random number generation." *Review of Scientific Instruments* 87, 035113, 1–9.

Luis Valbuena (2020, December). *Software Execution under Extreme Electromagnetic Interference* (Doctoral dissertation). University of New Mexico.

Leigh Van Valen. (1973). "An Evolutionary Law." 1–30.

Andrew Chi-Chih Yao. (1982, November). "Protocols for Secure Computations." *23rd Symposium on Foundations of Computer Science*. 160–164.