Simontool Supplemental Information

Brian Degnan, and Gregory Durgin

***Abstract*—The Simon Cipher is a low complexity, symmetric cipher that was designed for pervasive computing applications, such as RFID and IoT. This document provides supplemental information regarding Simon Cipher implementation for the paper "Simontool, Simulation Support for the Simon Cipher." This document describes simontool version 1.0.5.**

I. Introduction

THE Simon Cipher is a symmetric cipher that was released by the NSA in 2013 as part of "the Simon and Speck Families of Lightweight Block Ciphers" [1]. The Simon Cipher is optimized for hardware implementations and the cipher description did not include an explicit hardware reference. In the paper, "Simontool, Simulation Support for the Simon Cipher," we present a hardware reference implementation and an introduction to the simontool software. This document expands on the Simon Cipher and simontool description through examples and details that were deemed too mundane to include in the original paper.

A. Motivation

We are interested in the Simon Cipher and its application over AES in RFID implementations because the total transistor count and system complexity is low. Although AES encrypted data is used in RFID implementations, AES is a byte-oriented cipher, and the Simon Cipher is a bit oriented. We do not explicitly compare these ciphers as the goal of the document was introduce the software, simontool. The bit oriented nature and simplicity of the Simon Cipher make it ideal for a RFID-tag implementation where resources are extremely constrained. Furthermore, the Simon Cipher does not require SRAM for S-boxes or the replacement of S-boxes with field arithmetic as AES does [2] [3].

B. Constrained Implementations

As an example of a constrained cipher implementation, the ISO/IEC 18000 part 3 standard describes a 13.56MHz carrier with a 100kHz data rate, which gives 135 carrier clock cycles between the slow data clocks. If a transparent encryption scheme is used, it either has to complete in those 135 cycles, implement an asynchronous architecture, or take multiple data clock cycles. In ISO/IEC 18000 part 3, there is a further constraint where requests must be complete in $320\mu s$, or 32 cycles of the slow clock, which gives you a total of 4320 clock cycles to respond [4] [5].

TABLE I
SIMONTOOL ARGUMENT TABLE FOR VERSION 1.0.5

argument	value	description
-a	–	output PWL files
-b	size[1]	configure block size in bits
-c	count	limit clock cycles to count*
-d	–	decryption
-e	–	encryption
-h	name	input filename to encrypt/decrypt
-i	–	help page
-k	size[2]	configure key size in bits
-l	name	logfile name
-r	count	override round count*
-s	ascii	key hex in ASCII
-t	ascii	text block hex in ASCII
-u	–	modifies key schedule and breaks specification[3]
-v	text	max voltage for the PWL file outputs
-x	name	export LaTeXoutputs in the file of name*

[1] valid block sizes are 32, 64, 96, 128
[2] valid key sizes are 64, 72, 96, 128, 96, 144, 192, 256
[3] add to hash exploration
* experimental features

Consider the "barely better than no encryption" case of SIMON32/64, where we have a 64-bit block word and 32-bit encrypted data. SIMON32/64 in a 1-bit-serial implementation can already meet the RFID timing requirement off the carrier clock because it requires only 1024 cycles to complete all 32 of the rounds. This completion only requires 10 cycles of slow clock. Consider the more secure SIMON128/128 that has 68 rounds, this implementation results in a total of 8704 clock cycles in a 1-bit-serial implementation. This cycle count is higher than the 4320 available; however, a 4-bit serial implementation would only take 2176 clocks.

II. Simontool Overview

The simontool software is a program that is designed to aid in the hardware implementation of the Simon Cipher through simulation support and verification. The simontool program is written in C and supports a variety of argument options as listed in Table I. A basic example of running the program using the encryption test vector given in [1] follows as

```
simontool -e -b 48 -k 96 \
          -s 1a19181211100a0908020100 \
          -t 72696320646e \
          -l log.e.48.96.txt -a
```

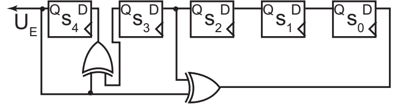
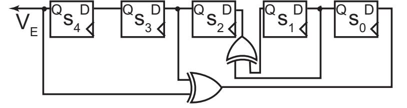
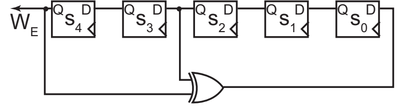
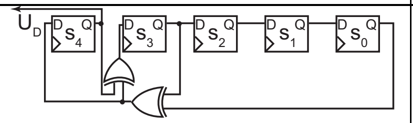
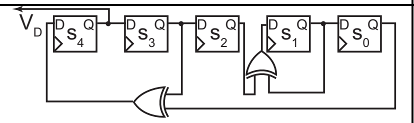
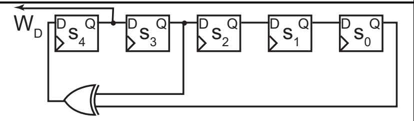
and this creates a log file (log.e.48.96.txt), the PWL files for SPICE simulation and outputs the result to stdout. The resulting log file has the format the state of z generation, the key and the cryptotext for each round. An example from the first 3 rounds of the log.e.48.96.txt file follows:

```
z[00] LFSR:---10000 toggle:------01 Z: 1
```

B. Degnan and G. Durgin are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.propagation.gatech.edu/).

Manuscript compiled September 25, 2017

TABLE II
THE TABLE DESCRIBES THE LFSR STATE FOR EACH ROUND IN THE MATHEMATICAL DESCRIPTION, AND THE EMULATED IMPLEMENTATION IN SIMONTOOL.

	U		V		W	
	$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$		$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$	
clock	encryption sequence u	decryption sequence u	encryption sequence v	decryption sequence v	encryption sequence w	decryption sequence w
0	10000	10000	10000	10000	10000	10000
1	10001	01000	00001	01000	00001	01000
2	10011	11100	00010	10100	00010	10100
3	10111	10110	01100	01100	00100	01010
4	11110	01011	11010	11010	01001	10101
5	01100	00101	11111	01010	10010	11010
6	11001	11010	10101	10111	00101	11101
7	00011	10101	01011	11001	01011	01110
8	00110	10010	11100	01100	10110	10111
9	01101	01001	11011	10110	01100	11011
10	11011	00100	11101	01001	11001	01101
11	00111	00010	11001	00100	10011	00110
12	01111	00001	10001	00010	00111	00011
13	11111	11000	00111	00011	01111	10001
14	01111	10100	01011	10011	11111	11000
15	11101	01010	10010	11011	11110	11100
16	01010	11101	00101	01111	11100	11110
17	10100	01111	01111	00101	11000	11111
18	11000	11111	11011	10010	10001	01111
19	00001	01111	10011	01011	00011	00111
20	00010	00111	00011	00111	00110	10011
21	00100	11011	00010	10001	01101	11001
22	01001	01101	00100	11001	11011	01100
23	10010	00110	01001	11100	10111	10110
24	10101	00011	10110	01011	01110	01011
25	11010	11001	01100	10101	11101	00101
26	00101	01100	11001	11111	11010	10010
27	01011	11110	10111	11010	10101	01001
28	10110	10111	01010	01100	01010	00100
29	11100	10011	10100	00010	10100	00010
30	01000	10001	01000	00001	01000	00001
31	10000	10000	10000	10000	10000	10000
Encrypt						
Decrypt						

```
k[00] 1a1918 121110 0a0908 020100
c[00] 726963 20646e
z[01] LFSR:---00001 toggle:------10 Z: 0
k[01] 7011c3 1a1918 121110 0a0908
c[01] 8b82a1 726963
z[02] LFSR:---00110 toggle:------01 Z: 0
k[02] b7ec48 7011c3 1a1918 121110
c[02] 546bee 8b82a1
```

In the listing above, the state of the LFSR, toggle bit, z bit, key register and crypto register are presented on a per-round basis. The most significant bit is the leftmost bit, and this is of note because the original Simon Cipher document is the reverse. The bit order is particularly relevant for the LFSR calculations in Table II. The toggle bit is implemented as a rotating register.

The PWL file output assumes microsecond resolution; however, the voltage can be set via the command line with the -v option. An example of PWL data for SPICE follows,

```
0.0e-6 3.3
0.99e-6 3.3
1.0e-6 0
1.99e-6 0
2.0e-6 0
2.99e-6 0
3.0e-6 0
```

where, the first term is the time and the second term is the voltage.

A. z_x generation

The Simon Cipher key schedule is complex and uses a stream of constant values to eliminate slide properties in the key expansion. The value of z_x is a function of the selection of block and key bit size, and this in turn, selects the logic of the Linear Feedback Shift Register (LFSR). Mathematically, the bitstream of the LFSR can be created in multiple ways; however, in hardware there exists only a single simple method and this method described in Table II. It is of note that circuit implementation is the mirror of the implementation described in the original Simon Cipher document. This is due to the fact that most significant bit is the leftmost bit in the tools that are being used to make the Simon Cipher circuits.

III. Test Vectors

The original Simon Cipher document listed a series of test vectors for verification of implementations [1]. The simontool software source tree includes script called *simontest.sh* that was used to verify the logical implementation of the Simon Cipher [6]. The simontool program has the ability to export LaTeXsource for generating bit fields using the TiKZ package by passing simontool the -x option. Examples of cipher verification for SIMON32/64 and SIMON128/128 are presented in the following sections.

A. SIMON32/64

The following listing generated the encryption bit fields for the block data as Figure 1 and the key expansion as Figure 2.

```
simontool -e -b 32 -k 64 \
          -s 1918111009080100 -t 65656877 \
          -l log.e.32.64.txt -x simon-e-32-64
```

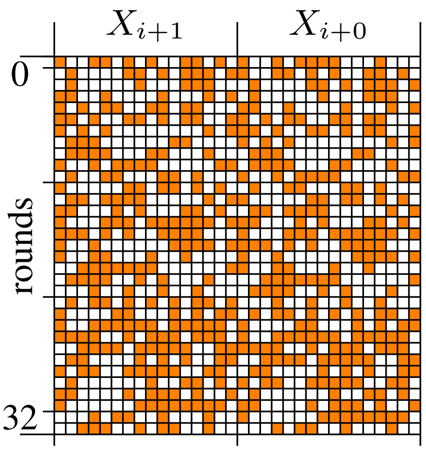


Fig. 1. The bit field with the encryption of the cryptotext 65656877 for SIMON32/64 where a white square is a 1 and a filled square is a 0.

B. SIMON128/128

The following listing generated the decryption bit fields for the block data as Figure 3 and the key expansion as Figure 4.

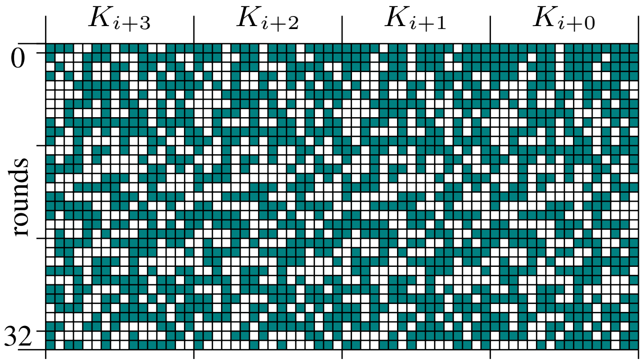


Fig. 2. The bit field for the encryption key expansion starting from 1918111009080100 for SIMON32/64 where a white square is a 1 and a filled square is a 0.

```
simontool -d -b 128 -k 128 \
          -s 6413494fda72360d1cb8547cd58c4df9 \
          -t 49681b1e1e54fe3f65aa832af84e0bbc \
          -l log.d.128.128.txt -x simon-d-128-128
```

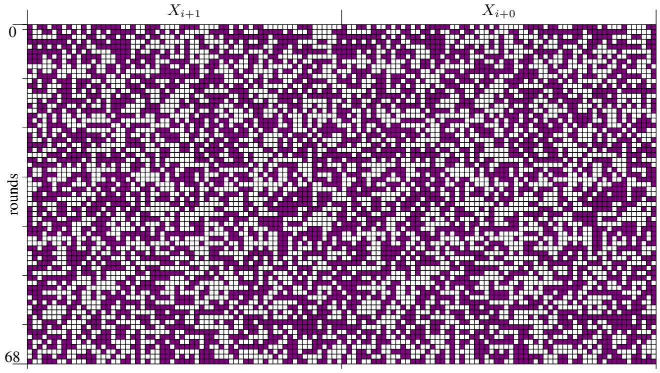


Fig. 3. The bit field with the decryption cryptotext of 49681b1e1e54fe3f65aa832af84e0bbc for SIMON128/128 where a white square is a 1 and a filled square is a 0.

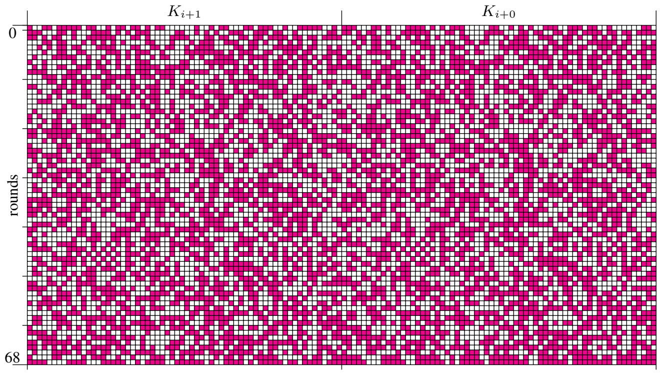


Fig. 4. The bit field for the decryption key expansion of 6413494fda72360d1cb8547cd58c4df9 for SIMON128/128 where a white square is a 1 and a filled square is a 0.

Appendix A
Field Methods

The weights for the LFSR were calculated by the following method,

$$\begin{bmatrix} s_4 \\ s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = C \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix}, \tag{1}$$

where C is the $GF(2^5)$ field that is U,V, or W, with the initial condition of $[00001]$, where the 1 is in s_4. An example of U for the initial condition follows as

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{2}$$

The equivalent MATLAB code for generating the U sequences shown in Table II is as follows:

```
function print_invert(p)
fprintf('%i   %i%i%i%i\n',p(5),...
  p(5),p(4),p(3),p(2),p(1));
end
U=[0 1 0 0 0;  %U matrix
   0 0 1 0 0;
   1 0 0 1 0;
   0 0 0 0 1;
   1 0 0 0 1 ];
A_o=[0;0;0;0;1]; %initial state
for i=1:31
  A=transpose(A_o)*U^(i-1);
  A=mod(A,2);
  print_invert(A);
end
```

Acknowledgment

This research was supported by an appointment to the Intelligence Community Postdoctoral Research Fellowship Program at the Georgia Institute of Technology, administered by Oak Ridge Institute for Science and Education through an interagency agreement between the U.S. Department of Energy and the Office of the Director of National Intelligence.

References

[1] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK families of Lightweight Block Ciphers." *IACR Cryptology ePrint Archive*, vol. 2013, p. 404, 2013.

[2] N. F. Pub, "197: Advanced Encryption Standard (AES)," *Federal Information Processing Standards Publication*, vol. 197, pp. 441–0311, 2001.

[3] C. Paar and J. Pelzl, *Understanding Cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[4] ISO, "Information technology – Radio frequency identification for item management – Part 3: Parameters for air interface communications at 13,56 MHz," International Organization for Standardization, Geneva, CH, Standard, Nov. 2010.

[5] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong authentication for RFID systems using the AES algorithm," in *Cryptographic Hardware and Embedded Systems-CHES 2004*. Springer, 2004, pp. 357–370.

[6] B. Degnan, "simontool," 2016, [Online; accessed 16-August-2016]. [Online]. Available: https://github.com/bpdegnan/simontool