SIMON and SPECK: Block Ciphers for the Internet of Things*

Ray Beaulieu Douglas Shors Jason Smith
Stefan Treatman-Clark Bryan Weeks Louis Wingers

National Security Agency
9800 Savage Road, Fort Meade, MD, 20755, USA

rayb@ccrwest.org, {djshors, jksmit3, sgtreat, beweeks, lrwinge}@tycho.ncsc.mil

12 June 2015

Abstract

The U.S. National Security Agency (NSA) developed the SIMON and SPECK families of lightweight block ciphers as an aid for securing applications in very constrained environments where AES may not be suitable. This paper summarizes the algorithms, their design rationale, along with current cryptanalysis and implementation results.

1 Introduction

Biologists make a distinction between specialist species, which occupy narrow ecological niches, and generalists, which can survive in a broader variety of environmental conditions. Specialists include Kirtland's warbler, a bird that only nests in 5–20 year-old jack pine forests, and the koala, which feeds (almost) exclusively on eucalyptus leaves. Generalists such as the American crow and the coyote are able to adapt to a variety of different environments. In a stable world, it's a good strategy to specialize, but when conditions change rapidly, specialists don't always fare so well.

The new age of pervasive computing is nothing if not rapidly changing. And yet, in the world of lightweight cryptography, specialists abound. Of course there are important research challenges associated with optimizing performance on particular platforms, and the direction taken by many in the field has been to take on such challenges, generally quite successfully. This can involve optimizing with respect to the instruction set for a certain microcontroller, or designing algorithms for a particular ASIC application (e.g., with hard-wired key or for IC printing), or designing specifically for low-latency applications, and so on.

We would argue that what's needed in the Internet of Things (IoT) era is not more Kirtland's warblers and koalas, as wonderful as such animals may be, but crows and coyotes. An animal that eats only eucalyptus leaves, even if it outcompetes the koala, will never become widely distributed. Similarly, a block cipher highly optimized for performance on a particular microcontroller will likely be outcompeted on other platforms, and could be of very limited utility in 15 years when its target platform is obsolete.

Of course it's hard to get a handle on block cipher performance on devices that don't yet exist. But what we can do is strive for *simplicity*, by designing algorithms around very basic operations that are certain to be supported by any future device capable of computation. SIMON and SPECK aim to be the sort of *generalist* block ciphers that we think will be required for future applications in the IoT era.

It would be unsatisfactory if we had to defer any discussion of performance because we're waiting for the arrival of future devices. But we can measure performance on current platforms, and in this paper we demonstrate the sort of performance that is achieved by SIMON and SPECK on a broad range of existing software and hardware platforms. We emphasize, however, that the main point is not the performance of SIMON and SPECK with respect to other algorithms on any particular platform. Rather, it's that by limiting the operations we rely on to a small list that works well in hardware and software, we

*This paper was accepted for the NIST Lightweight Cryptography Workshop, 20-21 July 2015.

obtain algorithms that are likely to perform well just about anywhere.

2 AES and Lightweight Cryptography

Before focusing our discussion on Simon and Speck, we'd like to better establish the state of play. In particular, we note that quite a lot of effort has gone into reshaping the current go-to block cipher, AES, into a solution for lightweight applications. Indeed, great strides have been made in this direction in the past 15 years or so. ASIC implementations of AES-128 have been developed with an area of just 2400 gate equivalents (GE) [38] and fast software implementations are available for 8-bit [41] and 16-bit [20] microcontrollers.

However, there are limits as to how far these types of adaptations can be pushed. They tend to fall short of what is required for today's most constrained environments, and surely won't meet tomorrow's needs. For example, the consensus has long been that a budget of 2000 GE is all the chip area that might reasonably be allocated for security on the most constrained RFID tags [34], and this is well out of reach for AES implementations. On microcontrollers, AES implementations can be very fast but they also tend to be large and complex. Implementations that decrease size or complexity certainly exist, but small implementations tend to be complex (and slow), while simple implementations tend to be large (and slow).

One further point about AES: not every application requires the same high level of security that AES is designed to provide. When resources are scarce, it doesn't always make sense to lavish them on an algorithm providing 128 (or 192 or 256) bits of security when 96 might suffice. In addition, the AES block size of 128 bits is not always optimal. An RFID authentication protocol may only ask that 64-bit quantities be encrypted, and demanding 128 bits of state when only 64 are necessary can amount to a significant waste of chip area.

These are the principal reasons for the development of new lightweight block ciphers, and many new algorithms have been proposed. Since the limitations of AES are more apparent in hardware than in software, most of the best efforts to date have focused on this aspect of the problem. This work has produced designs including PRESENT [17], KATAN [21], and Piccolo [49], each of which has a very small hardware footprint. But none was meant to provide high performance on constrained software-based devices, e.g., 8- and 16-bit microcontrollers. The designers of LED [33] and TWINE [54] are more intent on supporting software implementations, but these algorithms retain a bias toward hardware performance.

We believe a lightweight block cipher should be "light" on a wide range of hardware- and software-based devices, including ASICs, FPGAs, and 4-, 8-, 16-, and 32-bit microcontrollers. Moreover, as noted in [11], many of these devices will interact with a backend server, so a lightweight block cipher should also perform well on 64-bit processors.

It seems clear to us that there is a need for *flexible* secure block ciphers, i.e., ones which can perform well on *all* of these platforms. Our aim, with the design of Simon and Speck, is to make this sort of block cipher available for future use.

3 The Simon and Speck Block Ciphers

In 2011, prompted by potential U.S. government requirements for lightweight ciphers (e.g., SCADA and logistics applications) and the concerns with existing cryptographic solutions which we've noted above, we began work on the Simon and Speck block cipher families on behalf of the Research Directorate of the U.S. National Security Agency (NSA).

Because our customers will rely on commercial devices, we determined that the only realistic way to make the algorithms available would be to put them in the public domain. Furthermore, because cost will be such an important driver in this area—a fraction of a penny per device may make the difference between whether a cryptographic solution is viable or not—we were motivated to make Simon and Speck as simple, flexible, and lightweight as we could. Our hope was that their availability would make it possible to raise the security bar for future IoT devices.

The development process culminated in the publication of the algorithm specifics in June 2013 [9]. Prior to this, Simon and Speck were analyzed by NSA cryptanalysts and found to have security commensurate with their key lengths; i.e., no weaknesses were found. Perhaps more importantly, the algorithms have been pretty heavily scrutinized by the international cryptographic community for the last two years (see, e.g., [2], [3], [5], [4], [1], [6], [15], [16], [19], [25], [27], [35], [44], [48], [50], [53], [56], [59], [57],

size	alg	rounds		ref
		total	attacked	
32/64	Simon	32	21 (66%)	[59, 57, 58]
	Speck	22	14 (64%)	[27]
48/72	Simon	36	23 (64%)	[58]
	Speck	22	14 (64%)	[27]
48/96	Simon	36	24 (67%)	[58]
	Speck	23	15 (65%)	[27]
64/96	Simon	42	28 (67%)	[57, 58]
	Speck	26	18 (69%)	[27]
64/128	Simon	44	29 (66%)	[1, 58]
	Speck	27	19 (70%)	[27]
96/96	Simon	52	37 (71%)	[58]
	Speck	28	16 (57%)	[27]
96/144	Simon	54	37 (69%)	[58]
	Speck	29	17 (59%)	[27]
128/128	Simon	68	49 (72%)	[58]
	Speck	32	17 (53%)	[27]
128/192	Simon	69	49 (71%)	[58]
	Speck	33	18 (55%)	[3, 27]
128/256	Simon	72	50 (69%)	[1, 58]
	Speck	34	19 (56%)	[27]

Table 1: Security of Simon and Speck.

[28], [7], [23], [39]). Table 1 summarizes the cryptanalytic results as of this writing that attack the most rounds of Simon and Speck. (We note that the recent paper [7] claims 24, 23, and 24 round attacks on Simon 32/64, Simon 48/72, and Simon 48/96, respectively. The author informs us that this paper is currently under revision, and we have therefore not included those results in Table 1.) The content of the table is simple: there are no attacks on any member of the Simon or Speck families, and each block cipher maintains a healthy security margin.

As we see in the table, Simon and Speck are not simply block ciphers, but are block cipher *families*, each family comprising ten distinct block ciphers with differing block and key sizes to closely fit application requirements.

We will write Simon $2n/mn$ to mean the Simon block cipher with a $2n$-bit block and m-word (mn-bit) key. We will sometimes suppress mention of the key and just write Simon 128, for example, to refer to a version of Simon with a 128-bit block. The analogous notation is used for Speck.

The block and key sizes we support are shown in Table 2. The range here goes from tiny to large: a 32-bit block with a 64-bit key at the low end, to a 128-bit block with a 256-bit key at the high end.

We note that key lengths below 80 bits or so do not provide an especially high level of security, but they may still be useful for certain highly constrained applications where nothing better is possible.

block size	key sizes
32	64
48	72, 96
64	96, 128
96	96, 144
128	128, 192, 256

Table 2: Simon and Speck parameters.

The desire for flexibility through simplicity motivated us to limit the operations used within Simon and Speck to the following short list:

- modular addition and subtraction, + and −,
- bitwise XOR, ⊕,
- bitwise AND, &,
- left circular shift, S^j, by j bits, and
- right circular shift, S^{-j}, by j bits.

Speck gets its nonlinearity from the modular addition operation, which slightly favors software performance over hardware. Simon's nonlinear function is a bitwise AND operation, which tends to favor hardware over software. But modular addition can be computed efficiently in hardware, and similarly, bitwise AND is easy and natural in software.

The round functions for Simon $2n$ and Speck $2n$ each take as input an n-bit *round key* k, together with two n-bit *intermediate ciphertext* words. For Simon, the round function is the 2-stage Feistel map

$$R_k(x, y) = (y \oplus f(x) \oplus k,\ x),$$

where $f(x) = (Sx \,\&\, S^8 x) \oplus S^2 x$ and k is the round key. For Speck, the round function is the (Feistel-based) map

$$R_k(x, y) = ((S^{-\alpha}x + y) \oplus k,\ S^{\beta} y \oplus (S^{-\alpha}x + y) \oplus k),$$

with rotation amounts $\alpha = 7$ and $\beta = 2$ if $n = 16$ (block size = 32) and $\alpha = 8$ and $\beta = 3$ otherwise.

The round functions are composed some number of times which depends on the block and key size. See Table 1.

Each algorithm also requires a *key schedule* to turn a key into a sequence of round keys. We briefly describe the key schedules, but refer the reader to [9] for complete details.

For Simon, if we let the key value be $k_0, \ldots, k_{m-1}$ ($m \in \{2, 3, 4\}$ is the number of key words), the sequence of round keys is $k_0, k_1, k_2, \ldots$, where

$$\begin{aligned} k_{i+2} &= k_i \oplus (I \oplus S^{-1}) S^{-3} k_{i+1} \oplus C_i, \\ k_{i+3} &= k_i \oplus (I \oplus S^{-1}) S^{-3} k_{i+2} \oplus D_i, \\ k_{i+4} &= k_i \oplus (I \oplus S^{-1}) (S^{-3} k_{i+3} \oplus k_{i+1}) \oplus E_i, \end{aligned}$$

depending on whether m is 2, 3, or 4, respectively. The values C_i, D_i, and E_i are round constants which serve to eliminate slide properties; we omit discussion of them here. I is the $n \times n$ identity matrix.

Like Simon, Speck has 2-, 3-, and 4-word key schedules. Speck's key schedules are based on its round function, as follows. We let m be the number of words of key, and we write the key as $(\ell_{m-2}, \ldots, \ell_0, k_0)$. We then generate two sequences k_i and ℓ_i by

$$\begin{aligned} \ell_{i+m-1} &= (k_i + S^{-\alpha} \ell_i) \oplus i \quad \text{and} \\ k_{i+1} &= S^{\beta} k_i \oplus \ell_{i+m-1}. \end{aligned}$$

The value k_i is the ith round key, for $i \geq 0$. Note the round counter i here which serves to eliminate slide properties.

4 Design Notes

Efficiency and security are competing goals in cryptographic design, and understanding how to strike the right balance is the primary challenge faced by a designer. If security is not important, efficiency is easy: do nothing! Conversely, if efficiency doesn't matter, then it makes sense to build a round function using the most secure cryptographic components available, and then iterate an absurdly large number of times. But in the real world both of these things matter, and we'd like to design algorithms that are maximally efficient, while still providing the advertised level of security, as determined by the key size.

There is an important intellectual challenge associated with understanding optimally secure cryptographic components such as 8-bit S-boxes. However, we would argue that the way to design efficient cryptography, particularly cryptography for constrained platforms, is to forgo them in favor of very simple components, iterating an appropriate number of times to obtain a secure algorithm. Such simple components are by their nature cryptographically weak, making them unappealing to some designers. But simplicity enables compact implementations, and deciding on appropriate numbers of rounds is possible with analysis.

The question is whether there is something inherently wrong with this approach. It seems clear to us that there isn't: After all, a complex round function can always be factored into a composition of simple functions (transpositions, even), and so *every* block cipher is a composition of simple functions. It's just that in general the decomposition into simple functions is not useful to an implementer, because the factors tend to be unrelated, and so there is no associated efficient implementation of the algorithm. Viewed this way, we could imagine that Simon and Speck are based on complex round functions—a "round" in this sense may in fact mean eight of the usual rounds—but we've worked to make those complex round functions factor into identical functions, at least up to the translations by round key.

We now discuss in a bit more detail the thinking that went into the design of Simon and Speck.

Nonlinear and Linear Components

Most designers of lightweight block ciphers employ S-boxes to provide nonlinearity; a notable feature of Simon and Speck is their lack of dependence on S-boxes. The appeal of S-boxes is that, when used as a part of a substitution-permutation network (SPN), they allow for relatively easy security arguments, at least with respect to standard attacks. But for efficiency on constrained platforms, we believe that these sorts of designs are not optimal. We prefer to increase the one-time work necessary to do the cryptanalysis, in order to reduce the every-time work of encryption and decryption.

Lightweight block ciphers often use bit permutations as part of an SPN. The role of these bit permutations is to spread bits around in some optimal manner, and therefore allow SPN-style security arguments. If the target platform is an ASIC this is a perfectly reasonable thing to do, as such permutations are essentially free. But if we care about software implementations at all, then extreme care must be taken to ensure that the bit permutation can be done efficiently on a microprocessor. The bit permutations we use are all circular shifts, which are easy to effect on just about any platform. While we lose something in diffusion rates as compared with more general bit permutations, we are able to achieve significant improvements in software performance, even when increased round numbers are factored in.

One might argue that arbitrary bit permutations are fine in software, because efficient *bit-sliced* implementations are possible. However, it doesn't seem wise to rely on these, as they have drawbacks—including relatively expensive data transpose operations on the plaintext and ciphertext, and the inability to efficiently encrypt single plaintext blocks (and single encryptions will be necessary for many lightweight communication and authentication protocols). In addition, the code size and the RAM requirements tend to be quite large, making such implementations unsuitable for some lightweight applications.

Parameters

Both Simon and Speck are equipped with a single set of rotation parameters for all variants (with the exception of the smallest version of Speck, which has its own set of parameters). Besides allowing a succinct description of the family, this *uniformity* helps reduce the risk of coding errors whereby a programmer might mistakenly use the Simon 64/128 parameters, say, for Simon 128/128.

Many microcontrollers only support shifts by a single bit; the result is that a rotation by two bits is twice as expensive as a rotation by one bit. On the other hand, 8-bit rotations tend to be easy on 8-bit microcontrollers, as they correspond to simple relabelings of registers, and well supported through byte-swap or byte-shuffle operations on machines with larger word sizes. So for efficiency on a variety of software platforms, it's best to keep rotation amounts as close to multiples of eight as possible.

The Simon and Speck rotation amounts were carefully chosen with this consideration in mind. Both algorithms employ 8-bit rotations, and the other rotations used are as close to multiples of 8 as we could make them, without sacrificing security.

In-place Operations in Software

Speck's superior performance in software is due in part to the fact that it's possible to implement it entirely with in-place operations, and so moves are unnecessary. This can be seen in the following pseudocode for a round of Speck:

```
$x = \mathrm{RCS}(x, \alpha)$
$x = x + y$
$x = x \oplus k$
$y = \mathrm{LCS}(y, \beta)$
$y = y \oplus x$
```

Simon requires some moves, because multiple operations are done on a single word of intermediate ciphertext, and copies need to be made. This fact (combined with the fact that Simon uses a weaker nonlinear function than Speck, and so more rounds are required), makes Speck outperform Simon in software.

Encrypt/Decrypt Symmetry

To enable compact joint implementations of the encryption and decryption algorithms, it's best to make encryption look like decryption. Simon decryption can be accomplished by swapping ciphertext words, reading round keys in reverse order, and then swapping the resulting plaintext words.

We note that Simon beats Speck in this regard (Speck decryption requires modular subtraction, and the rotations are reversed), *because* its Feistel stepping performs all operations on one word, which is precisely why its software implementations required moves.

Key Schedule Considerations

Speck's reuse of the round function for key scheduling allows for reductions in code size and improves performance for software implementations requiring on-the-fly round key generation.

Because Simon was optimized for hardware, it does not take advantage of this software-oriented optimization. Instead, it uses a key schedule which was designed to be a little lighter than the round function.

Of course it is possible to have key schedules even simpler than the ones we have used for Simon and Speck; for example, one can produce round keys simply by cycling through key words. This leads to the possibility of "hard-wiring" the key in an ASIC implementation, thereby saving considerably on area by eliminating any flip-flops needed for holding the key. But such an approach, when used together with very simple round functions, can lead to related-key issues, and we therefore avoided it.

We believe the ability to use hard-wired key is of limited utility, and it runs counter to our flexibility goal by optimizing for a particular sort of use, perhaps to the detriment of other uses in the form of increased numbers of rounds or cryptanalytic weaknesses. Our key schedules do the minimal mixing that we thought would eliminate the threat of related-key attacks.

Both block ciphers include round constants, which serve to eliminate slide issues. Speck, where design choices were made to favor software over hardware, uses one-up counters. Simon achieves a small savings in hardware (at a small cost in software) by using a sequence of 1-bit constants generated by a 5-bit linear register.

As a final point, we omit plaintext and ciphertext key whitening operations, as such operations would increase circuit and code sizes. This means that the first and last rounds of the algorithms do nothing cryptographically, beyond introducing the first and last round keys.

We conclude this section by pointing to some work that we think helps to validate our approach to the design of Simon and Speck. Designing an algorithm to perform well on a particular platform is a straightforward proposition; we believe the real test is performance on *unintended* platforms, in particular platforms which may not even exist today.

As we've noted, it's hard to get a handle on an issue like this, but we have one data point that's interesting: Because of its simplicity (more precisely, its low multiplicative depth), Simon has been picked up by more than one team [36], [22] for use in the decidedly non-lightweight world of homomorphic encryption.

5 Implementations on Constrained Platforms

In this section, we quickly summarize implementation results for Simon and Speck on constrained platforms, beginning with ASICs and FPGAs, and then moving on to microcontrollers.

ASICs

Until recently, designers of lightweight cryptography primarily took aim at ASIC performance. As a result there are a number of excellent ASIC designs (see Table 3), all of which can be implemented with substantially less area than the 2400 GE required by AES. Much of this improvement is possible because of the hardware complexity of AES components, in particular its S-box. But a significant gain comes from the recognition that a 128-bit block size is not always required for constrained applications, and there is a considerable area savings to be had by reducing to a 64-bit block.

As we've noted, care must be taken with an ASIC design, or else software performance can suffer. Software performance is indeed a weakness of a number of existing algorithms. Simon and Speck have improved on the state of the art for hardware implementation, while also offering leading software performance.

Simon has ASIC implementations with the smallest areas achieved to date, when compared with block ciphers with the same block and key size and with flexible key. This is because the logic required for a bit-serial implementation (meaning that only one bit of the round function is computed per clock cycle) is minimal: computing a bit of the round function requires just one AND and three XORs, and so there isn't much room for further improvement. There is of course additional logic required for control (which we've also worked to minimize), and a few XORs are needed in the key schedule, etc., but for the smallest implementations, almost all the area is used by the flip-flops required to store the state.

Because the logic required to compute a bit of the round function is so small, implementations of Simon scale nicely: two bits or more can be updated in one clock cycle with minimal impact on area.

Speck is not far behind Simon with respect to small ASIC implementations. The primary differences are that Simon's AND gets replaced with a full adder, and some additional multiplexing is required because of how the state updates. Its area also scales well, but not quite as well as Simon's.

In the remainder of this section, we provide area and throughput data to illustrate the ASIC performance of Simon and Speck.

Our ASIC implementations were done in VHDL and synthesized using Synopsys Design Compiler

11.09-SP4 to target the ARM SAGE-X v2.0 standard cell library for IBM's 8RF 130 nm (CMR8SF-LPVT) process. Worst-case operating conditions were assumed. We did not proceed to place and route: in an actual chip there will be interconnect delays that haven't been accounted for, and these delays will likely significantly affect clock speeds. But we note that most work in this field—in particular the work cited in this paper—uses this approach, similarly ignoring interconnect delays, so this shouldn't bias our comparisons.

The smallest flip-flop available to us had an area of 4.25 GE. For a block cipher with a 64-bit block and 128-bit key, this means at least $4.25 \cdot 192 = 816$ GE are required for flip-flops. Our bit-serial implementations of Simon 64/128 and Speck 64/128 have areas of 958 GE and 996 GE, respectively. This means that they require (at most) $958 - 816 = 142$ GE and $996 - 816 = 180$ GE, respectively, for all the logic required to compute the round function, key schedule, and do the control, which includes loading the plaintext and reading out ciphertext. And of the 142 GE not devoted to storing the cipher and key for Simon 64/128, $11 \cdot 4.25 = 46.75$ GE, or about a third, are flip-flops needed to count rounds in order to signal the end of encryption.

Table 3 compares size-optimized ASIC implementations of Simon, Speck, and some other prominent block ciphers, listing the area and throughput at a fixed 100 kHz clock rate. Note that we show our absolute smallest implementations of Simon and Speck, with correspondingly low throughputs. Throughputs can be doubled, quadrupled, etc., for small area increases. See [9] for data regarding additional implementations. For example, quadrupling the throughput for Simon 128/128 and Speck 128/128 increases the area by just 29 GE and 116 GE, respectively.

An important caveat is that these comparisons consider implementations done by different authors, with perhaps different levels of effort, and using different cell libraries, so it's hard to make really meaningful inferences regarding small differences in the table.

Large differences, on the other hand, are meaningful, and comparing Simon and Speck with AES shows the dramatic savings possible with a lightweight block cipher. At the same security level, Simon and Speck nearly halve AES's 2400 GE area to 1234 and 1280 GE, respectively. Keeping the same 128-bit key size and reducing the block size to 64 bits further drops the areas to 958 and 996 GE. Using smaller block or key sizes results in even greater area reductions.

size	**algorithm**	**area (GE)**	**tput (kbps)**	**ref**
48/96	Simon	739	5.0	[9]
	Speck	794	4.0	[9]
64/80	TWINE	1011	16.2	[54]
	PRESENT	1030	12.4	[62]
	Piccolo	1043	14.8	[49]
	Katan	1054	25.1	[21]
	KLEIN	1478	23.6	[31]
64/96	Simon	809	4.4	[9]
	Speck	860	3.6	[9]
	KLEIN	1528	19.1	[31]
64/128	Simon	958	4.2	[9]
	Speck	996	3.6	[9]
	Piccolo	1334	12.1	[49]
	PRESENT	1339	12.1	[62]
96/96	Simon	955	3.7	[9]
	Speck	1012	3.4	[9]
128/128	Simon	1234	2.9	[9]
	Speck	1280	3.0	[9]
	AES	2400	56.6	[38]

Table 3: ASIC performance comparisons at a 100 kHz clock speed optimized for size.

Some applications won't require areas to be minimized; rather it may be important to maximize *efficiency* (throughput divided by area, in kbps/GE). The implementations in Table 3 have low efficiency, but efficiency can easily be raised by doing additional computation during each clock cycle, in effect to begin to amortize away the fixed cost of storing the state. The flexibility of Simon and Speck mean that many sorts of implementations are possible. See Section 6 for data regarding efficient implementations; in particular implementations which compute a full round per clock cycle, and implementations which fully unroll the algorithms.

We conclude this section by discussing *latency*, i.e., the time required to encrypt *one* plaintext block. Low-latency implementations of lightweight block ciphers have recently been much discussed; the leading voices have been the authors of [39]. The al-

algorithm	area (GE)	latency (ns)	clock (MHz)
PRINCE	8679	28.17	35.5
Simon 64/128	6576	27.56	399.2
	5072	31.90	344.9
Speck 64/128	6377	52.36	191.0

Table 4: Low-latency implementations of PRINCE, Simon, and Speck at 130 nm. The Simon and Speck implementations count 64 + 128 flip flops; the PRINCE implementation doesn't.

gorithm they propose, PRINCE, is a clever design which can encrypt in one clock cycle at the impressively small area of 8679 GE.[1]

It would appear that Simon and Speck are not low-latency designs, because they require many rounds. However, because of their simplicity, it's possible to compute multiple rounds per clock cycle, while maintaining reasonably good clock speeds. Indeed (at the same feature size) the comparably-sized Simon 64/128 can match PRINCE's latency at a significantly smaller area by implementing the combinational logic for 4 rounds, and encrypting in $\frac{44}{4} = 11$ cycles. In spite of its need to compute carry chains, Speck can get within a factor of two in latency, again at a smaller area.[2] (This implementation computes 3 rounds per clock cycle for a total of $\frac{27}{3}+1 = 10$ cycles—because of the way round keys are generated Speck requires a load cycle that Simon and PRINCE don't.) Of course these are not single-cycle implementations, but we don't see a compelling case that such implementations—at what seem to be artificially-constrained clock speeds—are necessary. See Table 4, where one Speck and two Simon implementations are shown; other latency/area trade-offs are possible but are omitted here.

FPGAs

We've shown that it's possible to realize considerable reductions in ASIC area by using Simon or Speck instead of an algorithm such as AES. The advantages of Simon and Speck become even more pronounced on FPGA platforms.

size	algorithm	area (slices)	tput (Mbit/s)	ref
64/128	Simon	24	9.6	†
	Simon	138	512	†
	Speck	34	7.0	†
	Speck	153	416	†
	PRESENT	117	28.4	[61]
	PRESENT	202	508	[43]
128/128	Simon	28	5.7	†
	Simon	36	3.6	[8]
	Simon (DPA)	87	3.0	[46]
	Simon	197	567	†
	Simon	375	867	†
	Speck	36	5.0	†
	Speck	232	455	†
	Speck	401	920	†
	AES	184	36.5	[24]

Table 5: FPGA performance comparisons on low-cost Xilinx Spartan FPGAs. All implementations are on the Spartan-3. Results marked with a † are our work. The Simon implementation labeled (DPA) is resistant to first-order DPA.

In this section we briefly discuss implementations of the algorithms on the Spartan-3, a low-end FPGA which is often used by cryptographers for comparisons. Table 5 presents some of these results for AES and PRESENT, alongside our results for Simon and Speck.

On this platform, the smallest reported implementation of AES-128 requires 184 slices [24]. Remarkably, Simon 128/128 can be implemented in just 28 slices (15% of the size of AES), and Speck 128/128 can be done in 36 slices (20% of AES's size). Comparisons with PRESENT also show dramatic area reductions: PRESENT-128 requires 117 slices; the comparable Simon 64/128 and Speck 64/128 algorithms require 24 and 34 slices—21% and 30% of the area—respectively.

If higher throughputs are required, area reductions are still possible, as can be seen in Table 5.

Other authors have reported Simon implementation results [13, 8, 32, 46] which are in line with our results, and extend them. In [32], it is shown that a joint implementation of all 10 versions of Simon can be done using 90 slices on the Spartan-3, which is about half the size of a single AES-128 implementa-

[1] We note that registers were not counted in this total, and any real system would need to register the data, thus increasing the area by about 10% to around 9500 GE.

[2] The PRINCE implementation supports decryption, while the Simon and Speck implementations that we consider here don't. But for applications requiring extremely low latency, it's likely that encrypt-only modes would be preferred.

		AVR			MSP430		
size	algorithm	ROM (bytes)	RAM (bytes)	cost (cyc/byte)	ROM (bytes)	RAM (bytes)	cost (cyc/byte)
			efficient implementations				
64/80	PRESENT [29]	936	0	1340	-	-	-
64/128	Speck	218	0	154	204	0	98
	Simon	290	0	253	280	0	177
	TWINE [37]	1208	23	326	-	-	-
128/128	Speck	460	0	171	438	0	105
	AES-128 [10]	970	18	146	-	-	-
	Simon	760	0	379	754	0	389
			fast implementations				
64/128	Speck	628	108	122	556	0	89
	Simon	436	176	221	324	0	153
128/128	AES-128 [40, 20]	1912	432	125	3147	176	132
	Speck	452	256	143	602	0	101
	Simon	510	544	337	1108	0	379

Table 6: Assembly implementations on the 8-bit AVR ATmega128 and 16-bit MSP430 microcontrollers.

tion. The 87-slice implementation of Simon 128/128 described in [46] provides resistance to first-order differential power analysis, again at about half the area of an *unprotected* AES-128 implementation.

Microcontrollers

We turn now to software implementations on 8-bit, 16-bit, and low-end 32-bit microcontrollers. Table 6 shows ROM and RAM usage and encryption cost (in cycles/byte) for assembly implementations of Simon, Speck, and a few other algorithms [40, 41]. The first half of the table shows implementations optimized for *efficiency*[3] and the second half implementations optimized for speed.

The data for PRESENT exemplifies the potential difficulty of adapting hardware-oriented algorithms to software; this algorithm is unable to match the performance of AES, and is easily beaten by Simon and Speck in both throughput and code size.[4]

For high-speed applications on the 8-bit AVR microcontroller, AES-128 is the fastest 128-bit block cipher we know of, beating Speck 128/128 by about 17%. However, because of its low memory usage, Speck 128/128 has higher efficiency than AES-128. And as key sizes increase, Speck overtakes AES in throughput because of how round numbers scale. Moreover, Speck 64/128, which has the same key size as AES-128, but a smaller block, is both smaller and slightly faster than AES-128.

On the 16-bit MSP430, Speck is the highest in efficiency and throughput. It is 23% faster than AES, uses no RAM and 81% less ROM. In [20] this performance advantage resulted in a 35% lower energy consumption compared to AES. Speck 64/128 consumes even fewer resources for the many applications where a smaller block size is acceptable.

Others' work supports our conclusions. In [26], C implementations of AES, Simon 64/96, Speck 64/96, and ten other lightweight algorithms are compared on the 8-bit AVR, 16-bit MSP430, and 32-bit ARM Cortex-M3 microcontrollers. Algorithms were ranked in two usage scenarios using a *figure of merit* balancing performance, RAM, and code size across the three platforms. Speck and Simon place first and fourth in a large data scenario and first and second in a scenario involving encryption of a single block.

On the 32-bit ARM processor, the authors of this paper find Speck and Simon to be simultaneously the smallest and fastest block ciphers for both of the scenarios they consider. We point out, however,

[3] We define efficiency to be encryption throughput in bytes per cycle, divided by ROM + 2 · RAM. See [10].

[4] We note that there is a faster bit-sliced implementation of PRESENT [42], which encrypts at 370.875 cycles per byte, plus about 40 cycles per byte for data transposition operations. But it's much larger, requiring 3816 bytes of ROM and 256 bytes of RAM.

size	algorithm	area (GE)	throughput (Mbps)	efficiency (kbps/GE)	clock (MHz)	implementation
64/128	Simon	1751	870	497	625	iterative
		44322	34243	773	535	key-agile pipeline
		35948	45070	1254	704	non-key-agile pipeline
	Speck	2014	634	315	307	iterative
		48056	23908	498	374	key-agile pipeline
		39992	29722	743	464	non-key-agile pipeline
128/128	Simon	2342	1145	489	626	iterative
		146287	106961	731	836	key-agile pipeline
		104790	87798	838	686	non-key-agile pipeline
	Speck	3290	880	268	234	iterative
		98003	41531	424	324	key-agile pipeline
		86976	52162	600	408	non-key-agile pipeline
128/256	Simon	3419	1081	316	625	iterative
		233204	100078	429	782	key-agile pipeline
		110875	87193	786	681	non-key-agile pipeline
	Speck	5159	1287	249	382	iterative
		163770	51705	316	404	key-agile pipeline
		97432	52056	534	407	non-key-agile pipeline

Table 7: Efficient, high-throughput 130 nm ASIC implementations of Simon and Speck

that their C implementations of AES are faster than those of Speck on the 8-bit and 16-bit platforms by about a factor of two, presumably due to the GNU C compiler's poor handling of rotations. Implementing the rotations in assembly should lead to greatly improved performance for our rotation-dependent designs.

It is our opinion that for lightweight applications on microcontrollers, if high performance is important, then Simon and Speck should be coded in assembly: because of the simplicity of the algorithms, these implementations are pretty straightforward, and they can improve performance by up to a factor of five over C implementations. Details on such implementations on the AVR microcontroller can be found in [10].

6 Implementations on Higher-end Platforms

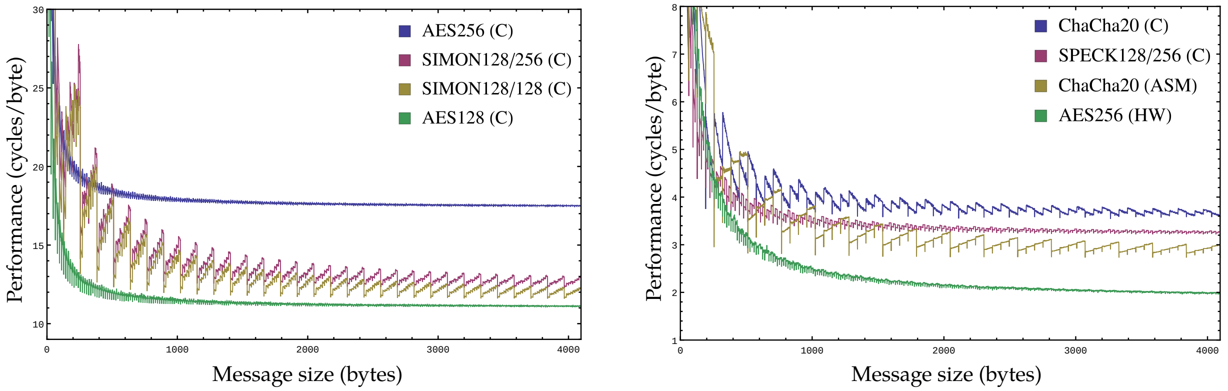
Constrained devices will need to communicate with other, similar devices, but will also need to communicate with higher-end systems. These systems may perform functions such as aggregating sensor or inventory data. To facilitate these sorts of interactions, and in particular to support efficient communication with large numbers of constrained devices, lightweight algorithms will need to perform well on both lightweight and "heavyweight" platforms.

High-throughput ASIC Implementations

Table 7 shows a sample of higher-throughput implementations on the same 130 nm ASIC process used to generate the Simon and Speck data in Table 3. Decryption is not supported in these implementations, but for Simon, in particular, it could be added at low cost due to the similarity of the encryption and decryption algorithms.

For each algorithm and block/key size an iterative and two fully-pipelined encryption implementations are presented. In the iterative case, a single copy of the round function is used to loop over the data for a number of cycles equal to the total number of rounds.

In the fully-pipelined case, a number of copies of the round function equal to the number of rounds is implemented, with registers in between. This allows a complete block of ciphertext to be output every clock cycle, once the pipeline is full. One of the fully-pipelined implementations is key-agile, meaning



(a) AES (without AES-NI) and Simon

(b) AES, ChaCha20, and Speck

Figure 1: Intel Xeon E5640 throughput in cycles/byte (smaller is better) for messages from 1–4096 bytes.

that every plaintext block to be encrypted can have its own associated key. The second fully-pipelined implementation is not key-agile: it saves area by requiring that all blocks in the pipeline use the same key, so that only one instance of the key schedule is necessary, rather than one for each level of the pipeline. Changing key for this second sort of implementation requires the new round keys to be loaded and the pipeline to be flushed.

The flexibility of Simon and Speck enables all sorts of implementations in between these performance extremes (e.g., iterated versions computing multiple rounds per clock cycle, and pipelined implementations with multiple rounds between stages), but we do not have the space to include those results here.

Simon and Speck have compelling advantages for high-throughput ASIC applications. This seems clear, even in view of the difficulties inherent in comparing implementations using different technologies and libraries. As a point of comparison, we consider the CLEFIA block cipher.[5] The designers of that algorithm report on a joint implementation [52] of the encryption and decryption algorithms[6] which has an efficiency of 401, using a 90 nm technology (9339 GE, 3.74 Gbit/s at 572 MHz). This is excellent performance relative to other block ciphers; indeed CLEFIA realizes the "world's highest hardware gate efficiency" [51].

We did ASIC implementations of Simon and Speck at this same 90 nm feature size. (Note that these results are not reported in Table 7, where the feature size is 130 nm.) Speck has a 8089 GE (encrypt-only) implementation, running at 1.404 GHz, for a throughput of 10.6 Gbit/s and an efficiency of 1307. Simon is even better: for 8011 GE, an encrypt-only version runs at 3.066 GHz, for a throughput of 17.1 Gbit/s and an efficiency of 2130. There may be differences in cell libraries, etc. (and we note again that interconnect delays are not considered in our work or in the CLEFIA work), but a factor of $\frac{2130}{401} > 5$ improvement is surely significant.

x86 and ARM Implementations

We have recently studied implementations of Simon and Speck as stream ciphers in counter mode on several higher-end 32-bit and 64-bit processors. These processors are likely to be used in systems such as smartphones, tablets, and servers communicating with constrained devices. We considered the the 32-bit Samsung Exynos 5 Dual (which includes NEON SIMD instructions), based on an ARM Cortex-A15, and two 64-bit Intel processors: the Xeon E5640 and Core i7-4770, representing the Westmere and Haswell architectures, respectively. Performance was benchmarked using SUPERCOP [12], making for fair comparison with the performance of highly-optimized implementations of AES and ChaCha20, in particular. The Simon and Speck code, all written in C, is available on GitHub [60]. Figure 1 illustrates the detailed data produced by SUPERCOP.

[5] CLEFIA is a lightweight ISO standard which supports high-throughput ASIC implementations.

[6] CLEFIA's symmetry means that there is little overhead in providing decryption functionality. On the other hand, the area won't go down by much for an encrypt-only version.

The overall results are similar on the ARM and the x86 platforms. The C implementations of Simon have better overall performance than the C implementations of AES for 256-bit keys and slightly worse performance for 128-bit keys. The C implementations of Speck 128/256 have better overall performance than the best C implementations of ChaCha20, a stream cipher especially noted for its speed.

Finally, we note that extremely high-performance instantiations of AES are possible on certain processors, for example using Intel's hardware AES-NI instructions. Despite this, Speck in software can come close to matching this high performance: on the Haswell architecture our C implementation of Speck 128/256 is only 33% slower than the AES-NI version of AES-256.

7 Side-Channel Mitigations

The most secure algorithm can become vulnerable to attack if it is implemented in a way that leaks information because power usage or execution time (or something else) is correlated to secret key values. Understanding these sorts of *side-channels* and how to eliminate them is an important line of research, and it's particularly relevant for constrained devices, which tend to lack physical countermeasures.

We very briefly discuss side-channel attacks and mitigations, and note some work in this area involving Simon and Speck.

One sort of side-channel attack exploits key-dependent variations in encryption times to recover secret information. Algorithms which are implemented using lookup tables, e.g., AES, on processors with cache memory can be particularly vulnerable to these *cache-timing attacks* [18]. Since Simon and Speck have no look-up tables, they are naturally immune to this type of attack.

Perhaps the most important type of side-channel attack uses key-dependent power emanations. Implementations of block ciphers typically are susceptible to such *differential power analysis (DPA)* attacks unless countermeasures are taken. Because of Simon's low-degree round function, *masking* countermeasures are especially efficient; see [47, 46]. In particular, the second of these papers demonstrate a threshold implementation of Simon 128/128 which provides resistance to first-order DPA for 87 slices on a Spartan-3 FPGA. This makes it less than half the size of the smallest reported unprotected Spartan-3 implementation of AES, and 25% smaller than unprotected implementations of PRESENT-128. (And PRESENT-128 is not exactly a comparable algorithm, since it has a block size of 64 bits, and the version of Simon they consider has a block size of 128 bits.)

We are not aware of similar work to protect Speck, but there are other countermeasures that apply equally to both Simon and Speck. One such measure aims to confound DPA by partially unrolling an algorithm [14]. We've done such implementations of Simon and Speck, but don't have the space in this paper to discuss them. Briefly, for the 64-bit block and 128-bit key size, there is an ASIC implementation of Simon that computes four full rounds per clock cycle and requires 3290 GE. A similar implementation of Speck computes three rounds per clock cycle and has an area of 3120 GE. We have not done side channel analysis for these implementations.

Another mitigation uses frequent key updating [55]. The tiny hardware implementations of Simon and Speck in Tables 3 and 5 are key agile, meaning the key can be changed with each run without incurring a significant performance penalty, and so they would be good candidates for use with this strategy.

8 Conclusion

We have sought in this paper to demonstrate the sort of performance that Simon and Speck can achieve. Most importantly, Simon and Speck have an edge over other algorithms *not* in terms of head-to-head comparisons on particular platforms (although it appears that on most platforms one of Simon or Speck is the best existing algorithm, and the other is not far behind), but by virtue of their flexibility. This flexibility is a consequence of the simplicity of the designs, and means the algorithms admit small ASIC, FPGA, microcontroller, and microprocessor implementations, but can also achieve very high throughput on all of these platforms. Their flexibility makes Simon and Speck ideal for use with heterogeneous networks, where algorithms optimized for particular platforms or usages will not be appropriate.

The simplicity of Simon and Speck has additional benefits. First, they are very easy to implement, and efficient implementations can be had for minimal work; this is in marked contrast to the situation for algorithms such as AES, where a decade of research was required to find near-optimal implementations. Coding errors are much easier to avoid for simple algorithms. In addition, simplicity enables relatively cheap side-channel mitigations, and makes the algorithms attractive for unanticipated uses (such as homomorphic encryption). Last, but not least, simplicity makes the algorithms attractive targets for cryptanalysis. Complexity in this regard presents a barrier to entry, and this tends to limit the amount of scrutiny that an algorithm receives. Because of their simplicity (and perhaps because of their source!), Simon and Speck have been quite thoroughly vetted by the cryptographic community in the two years since their publication.

Simon and Speck are also unique among existing lightweight block ciphers in their support for a broad range of block and key sizes, allowing the cryptography to be precisely tuned to a particular application.

We are hopeful that the approach we have taken to the design of Simon and Speck means they will continue to offer high performance on tomorrow's IoT devices.

Bibliography

[1] M. A. Abdelraheem, J. Alizadeh, H. A. Alkhzaimi, M. R. Aref, N. Bagheri, P. Gauravaram, and M. M. Lauridsen. Improved Linear Cryptanalysis of Reduced-round SIMON. Cryptology ePrint Archive, Report 2014/681, 2014. http://eprint.iacr.org/2014/681.pdf.

[2] F. Abed, E. List, S. Lucks, and J. Wenzel. Differential and Linear Cryptanalysis of Reduced-Round Simon. Cryptology ePrint Archive, Report 2013/526, 2013. http://eprint.iacr.org/2013/526.pdf.

[3] F. Abed, E. List, S. Lucks, and J. Wenzel. Differential Cryptanalysis of Round-reduced Simon and Speck. In *Fast Software Encryption, FSE 2014*, LNCS. Springer, 2014.

[4] J. Alizadeh, H. AlKhzaimi, M. R. Aref, N. Bagheri, P. Gauravaram, A. Kumar, M. M. Lauridsen, and S. K. Sanadhya. Cryptanalysis of SIMON Variants with Connections. In Saxena and Sadeghi [45], pages 90–107.

[5] J. Alizadeh, N. Bagheri, P. Gauravaram, A. Kumar, and S. K. Sanadhya. Linear Cryptanalysis of Round Reduced Simon. Cryptology ePrint Archive, Report 2013/663, 2013. http://eprint.iacr.org/2013/663.pdf.

[6] H. A. Alkhzaimi and M. M. Lauridsen. Cryptanalysis of the SIMON Family of Block Ciphers. Cryptology ePrint Archive, Report 2013/543, 2013. http://eprint.iacr.org/2013/543.pdf.

[7] T. Ashur. Improved Linear Trails for the Block Cipher Simon. Cryptology ePrint Archive, Report 2015/285, 2015. http://eprint.iacr.org/2015/285.pdf.

[8] A. Aysu, E. Gulcan, and P. Schaumont. SIMON Says, Break the Area Records for Symmetric Key Block Ciphers on FPGAs. Cryptology ePrint Archive, Report 2014/237, 2014. http://eprint.iacr.org/2014/237.pdf.

[9] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK Families of Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.pdf.

[10] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK Block Ciphers on AVR 8-bit Microcontrollers. In Eisenbarth and Öztürk [30].

[11] R. Benadjila, J. Guo, V. Lomné, and T. Peyrin. Implementing Lightweight Block Ciphers on x86 Architectures. Cryptology ePrint Archive, Report 2013/445, 2013. http://eprint.iacr.org/2013/445.pdf.

[12] D. J. Bernstein and T. Lange. eBACS: ECRYPT Benchmarking of Cryptographic Systems. http://bench.cr.yp.to.

[13] S. Bhasin, T. Graba, J. Danger, and Z. Najm. A look into SIMON from a side-channel perspective. In *Hardware-Oriented Security and Trust, HOST 2014*, pages 56–59. IEEE Computer Society, 2014.

[14] S. Bhasin, S. Guilley, L. Sauvage, and J.-L. Danger. Unrolling Cryptographic Circuits: A Simple Countermeasure Against Side-Channel Attacks. In J. Pieprzyk, editor, *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *LNCS*, pages 195–207. Springer, 2010.

[15] A. Biryukov, A. Roy, and V. Velichkov. Differential Analysis of Block Ciphers SIMON and SPECK. In *Fast Software Encryption, FSE 2014*, LNCS. Springer, 2014.

[16] A. Biryukov and V. Velichkov. Automatic Search for Differential Trails in ARX Ciphers. In J. Benaloh, editor, *Topics in Cryptology - CT-RSA 2014*, volume 8366 of *LNCS*, pages 227–250. Springer, 2014.

[17] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Blockcipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, 2007.

[18] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In *Cryptographic Hardware and*

Embedded Systems - CHES 2006, volume 4249 of *LNCS*, pages 201–215. Springer, 2006.

[19] C. Boura, M. Naya-Plasencia, and V. Suder. Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon (Full Version). Cryptology ePrint Archive, Report 2014/699, 2014. `http://eprint.iacr.org/2014/699.pdf`.

[20] B. Buhrow, P. Riemer, M. Shea, B. Gilbert, and E. Daniel. Block Cipher Speed and Energy Efficiency Records on the MSP430: System Design Trade-Offs for 16-bit Embedded Applications. Cryptology ePrint Archive, Report 2015/011, 2015. `http://eprint.iacr.org/2015/011.pdf`.

[21] C. D. Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009.

[22] B. Carmer and D. W. Archer. Block Ciphers, Homomorphically. Galois, Inc. Blog, December 2014. `http://galois.com/blog/2014/12/block-ciphers-homomorphically/`.

[23] Z. Chen, N. Wang, and X. Wang. Impossible Differential Cryptanalysis of Reduced Round SIMON. Cryptology ePrint Archive, Report 2015/286, 2015. `http://eprint.iacr.org/2015/286.pdf`.

[24] J. Chu and M. Benaissa. Low area memory-free FPGA implementation of the AES algorithm. In D. Koch, S. Singh, and J. Tørrensen, editors, *Field Programmable Logic and Applications (FPL) 2012*, pages 623–626. IEEE, 2012.

[25] N. Courtois, T. Mourouzis, G. Song, P. Sepehrdad, and P. Susil. Combined Algebraic and Truncated Differential Cryptanalysis on Reduced-round Simon. In M. S. Obaidat, A. Holzinger, and P. Samarati, editors, *SECRYPT 2014*, pages 399–404. SciTePress, 2014.

[26] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov. Triathlon of Lightweight Block Ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/209, 2015. `http://eprint.iacr.org/2015/209.pdf`.

[27] I. Dinur. Improved Differential Cryptanalysis of Round-Reduced Speck. In A. Joux and A. M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014*, volume 8781 of *LNCS*, pages 147–164. Springer, 2014.

[28] I. Dinur, O. Dunkelman, M. Gutman, and A. Shamir. Improved Top-Down Techniques in Differential Cryptanalysis. Cryptology ePrint Archive, Report 2015/268, 2015. `http://eprint.iacr.org/2015/268.pdf`.

[29] T. Eisenbarth, S. S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A Survey of Lightweight-Cryptography Implementations. *IEEE Design & Test of Computers*, 24(6):522–533, 2007.

[30] T. Eisenbarth and E. Öztürk, editors. *Lightweight Cryptography for Security and Privacy - LightSec 2014*, volume 8898 of *LNCS*. Springer, 2014.

[31] Z. Gong, S. Nikova, and Y. W. Law. KLEIN: A New Family of Lightweight Block Ciphers. In A. Juels and C. Paar, editors, *RFID Security and Privacy - RFIDSec 2011*, volume 70555 of *LNCS*, pages 1–18. Springer, 2011.

[32] E. Gulcan, A. Aysu, and P. Schaumont. A Flexible and Compact Hardware Architecture for the SIMON Block Cipher. In Eisenbarth and Öztürk [30].

[33] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED Block Cipher. In *Cryptographic and Embedded Systems - CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, 2011.

[34] A. Juels and S. A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 293–308. Springer, 2005.

[35] S. Kölbl, G. Leander, and T. Tiessen. Observations on the SIMON block cipher family. Cryptology ePrint Archive, Report 2015/145, 2015. `http://eprint.iacr.org/2015/145.pdf`.

[36] T. Lepoint and M. Naehrig. A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In D. Pointcheval and D. Vergnaud, editors, *AFRICACRYPT 2014*, volume 8469 of *LNCS*, pages 318–335. Springer, 2014.

[37] K. Minematsu. TWINE Block Cipher. Personal communication regarding results from [54], July 2014.

[38] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *LNCS*, page 69. Springer, 2011.

[39] T. Mourouzis, G. Song, N. Courtois, and M. Christofii. Advanced Differential Cryptanalysis of Reduced-Round SIMON64/128 Using Large-Round Statistical Distinguishers. Cryptology ePrint Archive, Report 2015/481, 2015. `http://eprint.iacr.org/2015/481.pdf`.

[40] D. A. Osvik. Fast Implementations of AES on Various Platforms. Personal communication regarding results from [41], June 2014.

[41] D. A. Osvik, J. W. Bos, D. Stefan, and D. Canright. Fast Software AES Encryption. In *Fast Software Encryption, FSE 2010*, volume 6147 of *LNCS*, pages 75–93. Springer, 2010.

[42] K. Papagiannopoulos. High throughput in slices: the case of PRESENT, PRINCE and KATAN64 ciphers. In Saxena and Sadeghi [45], pages 137–155.

[43] A. Y. Poschmann. *Lightweight Cryptography: Cryptographic Engineering for a Pervasive World*. PhD thesis, Ruhr University Bochum, 2009.

[44] R. Rabbaninejad, Z. Ahmadian, M. Salmasizadeh, and M. R. Aref. Cube and Dynamic Cube Attacks on SIMON32/64. In *ISCISC 2014*, pages 98–103, 2014.

[45] N. Saxena and A. Sadeghi, editors. *Radio Frequency Identification: Security and Privacy Issues - RFIDSec 2014*, volume 8651 of *LNCS*. Springer, 2014.

[46] A. Shahverdi, M. Taha, and T. Eisenbarth. Silent Simon: A Threshold Implementation under 100 Slices. Cryptology ePrint Archive, Report 2015/172, 2015. `http://eprint.iacr.org/2015/172.pdf`.

[47] D. Shanmugam, R. Selvam, and S. Annadurai. Differential Power Analysis Attack on SIMON and LED Block Ciphers. In R. S. Chakraborty, V. Matyas, and P. Schaumont, editors, *Security, Privacy, and Applied Cryptography Engineering, SPACE 2014*, volume 8804 of *LNCS*, pages 110–125. Springer, 2014.

[48] D. Shi, L. Hu, S. Sun, L. Song, K. Qiao, and X. Ma. Improved Linear (hull) Cryptanalysis of Round-reduced Versions of SIMON. Cryptology ePrint Archive, Report 2014/973, 2014. `http://eprint.iacr.org/2014/973.pdf`.

[49] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *Cryptographic and Embedded Systems - CHES 2011*, volume 6917 of *LNCS*, pages 342–357. Springer, 2011.

[50] L. Song, L. Hu, B. Ma, and D. Shi. Match Box Meet-in-the-Middle Attacks on the SIMON Family of Block Ciphers. In Eisenbarth and Öztürk [30].

[51] Sony Corporation. CLEFIA: The 128-bit Blockcipher. `http://www.sony.net/Products/cryptography/clefia/`.

[52] T. Sugawara, N. Aoki, and A. Satoh. High-performance ASIC implementations of the 128-bit block cipher CLEFIA. In *International Symposium on Circuits and Systems (ISCAS) 2008*, pages 2925–2928. IEEE, 2008.

[53] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-oriented Block Ciphers. In *Advances in Cryptology - ASIACRYPT 2014*, volume 8874 of *LNCS*, pages 158–178. Springer, 2014.

[54] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayahi. Twine: A Lightweight Block Cipher for Multiple Platforms. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography, SAC 2012*, volume 7707 of *LNCS*, pages 339–354. Springer, 2012.

[55] M. M. I. Taha and P. Schaumont. Key Updating for Leakage Resiliency With Application to AES Modes of Operation. *IEEE Transactions on Information Forensics and Security*, 10(3):519–528, 2015.

[56] Y. Todo. Structural Evaluation by Generalized Integral Property. Cryptology ePrint Archive, Report 2015/090, 2015. `http://eprint.iacr.org/2015/090.pdf`.

[57] N. Wang, X. Wang, K. Jia, and J. Zhao. Improved Differential Attacks on Reduced SIMON Versions. Cryptology ePrint Archive, Report 2014/448, June 2014. `http://eprint.iacr.org/eprint-bin/versions.pl?entry=2014/448`.

[58] N. Wang, X. Wang, K. Jia, and J. Zhao. Differential Attacks on Reduced SIMON Versions with Dynamic Key-guessing Techniques. Cryptology ePrint Archive, Report 2014/448, February 2015. `http://eprint.iacr.org/2014/448.pdf`.

[59] Q. Wang, Z. Liu, K. Varici, Y. Sasaki, V. Rijmen, and Y. Todo. Cryptanalysis of Reduced-round SIMON32 and SIMON48. Cryptology ePrint Archive, Report 2014/761, 2014. `http://eprint.iacr.org/2014/761.pdf`.

[60] L. Wingers. Software for SUPERCOP Benchmarking of SIMON and SPECK. Available at `http://github.com/lrwinge/simon_speck_supercop`.

[61] P. Yalla and J.-P. Kaps. Lightweight Cryptography for FPGAs. In *Reconfigurable Computing and FPGAs, ReConFig '09*, pages 225–230, December 2009.

[62] H. Yap, K. Khoo, A. Poschmann, and M. Henricksen. EPCBC — A Block Cipher Suitable for Electronic Product Code Encryption. In D. Lin, G. Tsudik, and X. Wang, editors, *Cryptology and Network Security, CANS 2011*, volume 7092 of *LNCS*, pages 76–97. Springer, 2011.