Performance of the SIMON and SPECK Families of Lightweight Block Ciphers

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

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

{rabeaul, djshors, jksmit3, sgtreat, beweeks, lrwinge}@tycho.ncsc.mil

29 May 2012

Abstract

In this paper we provide performance data for two new families of lightweight block ciphers, SIMON and SPECK, each of which comes in a variety of widths and key sizes. While many lightweight block ciphers exist, most were designed to perform well on a single platform and were not meant to provide high performance across a range of devices. The aim of SIMON and SPECK is to fill the need for secure, flexible, and analyzable lightweight block ciphers. Each offers excellent performance on hardware and software platforms, is flexible enough to admit a variety of implementations on a given platform, and is amenable to existing cryptanalytic techniques. While both perform exceptionally well across the full spectrum of lightweight applications, SIMON is tuned for optimal performance in hardware, and SPECK for optimal performance in software.

Performance of the SIMON and SPECK Families of Lightweight Block Ciphers

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

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

{rabeaul, djshors, jksmit3, sgtreat, beweeks, lrwinge}@tycho.ncsc.mil

29 May 2012

1. Introduction

Existing cryptographic algorithms were for the most part designed to meet the needs of the desktop computing era. Such cryptography tends not to be particularly well-suited to the emerging era of *pervasive computing*, in which many highly constrained hardware- and software-based devices will need to communicate wirelessly with one another. And security is important for many of these devices: a hacker should not be able to take control of your insulin pump or override the brakes in your car.

The field of *lightweight cryptography* addresses security issues for highly constrained devices, and many algorithms, block ciphers in particular, have been proposed for lightweight cryptographic applications—see the references at the end of this paper for a sample. The obvious first question is "Why not just use AES for lightweight applications?" Indeed AES [DR02] has been suggested for lightweight use, and given its stature, we believe it should be used whenever appropriate. However, for the most constrained environments, AES is not the

right choice: in hardware, for example, the emerging consensus in the academic literature is that area should not exceed 2000 gate equivalents (see [JW05]), while the smallest implementation of AES requires 2400* [MPL+11].

Among the block ciphers intended for use on constrained devices, some have been designed specifically to perform well on dedicated Application-Specific Integrated Circuits (ASICs), and thus can be realized by small circuits with minimal power requirements. Others are meant to perform well on low-cost microcontrollers with limited flash, SRAM, and/or power availability. Unfortunately, design choices meant to optimize performance on one platform often adversely affect performance on another.†

We have designed two families of highly-optimized block ciphers, SIMON and SPECK, that are flexible enough to provide excellent performance in *both* hardware and software environments. To the best of our knowledge, *each* of SIMON and SPECK outperforms both the best existing hardware algorithms (in terms of the area required to achieve a given throughput), and the best existing software algorithms (in terms of code size and memory usage). In addition, both families consist of algorithms having a range of block and key sizes, and admitting a variety of implementations, thereby allowing for a close match with application requirements and security needs.

SIMON has been optimized for performance on hardware devices, and SPECK for performance in software. But we emphasize that both families perform exceptionally well in *both* hardware and software, offering the flexibility across platforms that will be required by future applications.

Table 1.1 shows hardware and software performance figures for SIMON, SPECK, and some other popular block ciphers. This table (together with the expanded versions of it that appear later in the paper) is the primary content of the paper, and that is why we present it here at the outset. For readers with some background in the field, much of the table will make sense without further explanation, but more details can be found in Section 4.

*The algorithms we discuss in this paper can be implemented in hardware and software with roughly half the footprint of AES, and this greatly expands their range of application.

†for example, the reliance on software-unfriendly bit permutations or bit-serial computations, and not-especially-hardware-friendly S-boxes.

		hardware		software		
size	**name**	**area (GE)**	**throughput (kbps)**	**flash (bytes)**	**SRAM (bytes)**	**throughput (kbps)**
48/96	SIMON	782	15.0	176	0	560.6
	SPECK	882	12.5	134	0	933.2
	EPCBC	1008	12.1	[365]	0	[95.5]
64/80	PRESENT	1030	12.4	[487]	0	95.5
	Piccolo	1043	14.8	n/a	n/a	n/a
	KATAN	1054	25.1	272	18	14.2
	TWINE	1116	11.8	[396]	191	54.5
	KLEIN	1478	23.6	766	18	167.9
64/96	SIMON	854	19.0	242	0	535.3
	SPECK	984	15.7	180	0	900.6
	KLEIN	1528	19.1	[766]	[18]	[134.3]
64/128	SIMON	1011	18.1	246	0	521.9
	SPECK	1128	14.2	188	0	831.8
	Piccolo	1334	12.1	n/a	n/a	n/a
	PRESENT	1339	12.1	[487]	[0]	[95.5]
96/96	SIMON	985	12.5	398	0	470.0
	SPECK	1126	13.8	284	0	814.9
	EPCBC	1333	12.1	[730]	0	[95.5]
128/128	SIMON	1274	12.9	638	0	366.9
	SPECK	1501	21.6	438	0	644.6
	AES	2400	56.6	943	33	445.2

Table 1.1: Performance comparisons. Size is block size/key size; hardware refers to an ASIC implementation, and software to an implementation on an 8-bit microcontroller; clock speeds are 100 kHz (hardware) and 16 MHz (software). The best performance for a given size is indicated in red, the second best in blue. Numbers in brackets are our estimates.

No decision has yet been made regarding the public release of SIMON and SPECK. For this reason, this paper shows performance data but doesn't de-

scribe the algorithms themselves. Whether or not SIMON and SPECK are eventually published, we hope that this paper serves both to spur the field of lightweight cryptography and to encourage developers to incorporate cryptography in those lightweight applications where it is needed.

2. Lightweight block cipher design considerations

The term *lightweight* is used broadly to mean that an algorithm is suitable for use on *some* constrained platform. But the features that make an algorithm excel on an 8-bit microcontroller, say, do not necessarily imply that it can be realized by an extremely small circuit. Thus the definition of the term lightweight is really platform-dependent, and so some general discussion is in order regarding our goals.

First, we make no attempt to optimize for a specific application. We prefer to make application-independent design choices that ensure good performance on both ASICs and 8-bit microcontrollers, with the idea that good performance in these environments will carry over to other important platforms as well—FPGAs, 4- and 16-bit microcontrollers, 32-bit processors, and so on.

The principal aim is to provide algorithms that (1) have very small hardware implementations, and at the same time (2) have low-power software implementations on small microcontrollers, with minimal flash and SRAM usage.

It is important to note that for many (but not all) lightweight applications, throughput is not the top priority. For instance, a throughput of around 12 kilobits per second (kbps) at 100 kHz is adequate for many hardware-based RFID applications (and for this reason the hardware performance values in Table 1.1 are for the smallest implementation exceeding this threshold); other applications, e.g., access control, can tolerate even lower throughput. Our desire for low-area hardware designs means that we favor simple, low-complexity round functions, even if that means many rounds are required.

For a lightweight algorithm to be as useful as possible, it should be *flexible* enough not just to be implemented efficiently on a variety of platforms, but also to allow for a variety of implementations on a single platform. For hardware applications, this means that it should be possible to take advantage of the

available real estate. For extremely constrained hardware environments, very low-area implementations should be achievable, but if constraints are not so tight, one should be able to take advantage of this fact with larger-area, higher-throughput implementations. For software applications, very small flash and SRAM usage should be attainable, but high-throughput, low-power implementations should be achievable as well.

Existing lightweight algorithms are not always as flexible as they could be. One important consideration is the extent to which an algorithm can be *serialized* in hardware. An implementation that updates a single bit at a time is said to be fully serialized, or *bit-serial*, while one that updates the entire block during each cycle is said to be unserialized, or *iterated*. Some algorithms are inherently bit-serial, making for very small implementations in hardware but preventing the possibility of higher-throughput, iterated implementations. On the other hand, many algorithms are S-box based, precluding the possibility of efficient serialization at a level below the width of the S-box. This is reflected in the AES row of Table 1.1, where we see a throughput value much higher than the requisite 12 kbps, since AES is built from 8-bit S-boxes.* Algorithms that can be efficiently serialized at any level dividing the word size provide better optimization opportunities.

Flexibility extends in another direction as well: since applications and devices vary, a variety of block and key sizes is useful. For instance, block sizes of 64 and 128 bits are prevalent in the world of desktop computing, but atypical block sizes of 48 or 96 bits are optimal for electronic product code (EPC) applications. Key sizes, on the other hand, are related to the desired level of security: a very low-cost device may achieve adequate security using just 64 bits of key, while more sensitive applications (running on suitably higher-cost devices) may require as many as 256 bits of key.

This is our first mention of security, which is of course the primary goal of cryptography. In addition to meeting performance objectives, it is important that a cryptographic algorithm have the advertised level of security. Since confidence in the security of an algorithm increases as it is analyzed, a designer should strive to create algorithms that are amenable to current cryptanalytic

*And SIMON 128/128 can achieve a throughput of 51.4 kbps, comparable with this implementation of AES, at an area of 1503 GE—just 63% the area of AES. See Table 5.2.

techniques. Designs that are difficult to analyze may well be secure but hardly inspire confidence. SIMON and SPECK have simple round functions that lend themselves well to analysis. Assuming the eventual public release of these algorithms, we trust that the simplicity of the designs will entice the cryptographic community to expend some effort analyzing them.

While our intent is that SIMON and SPECK provide the advertised level of security, a large security margin is a luxury that we can't always afford when resources are scarce. Hence we have built in what we believe is a sufficient security margin, but not an excessive one, and one which perhaps is a bit tighter than would be supplied in a more traditional setting.

This brings up an important issue, and one we would like to see discussed further: What sorts of *cryptanalytic* adversaries should be considered in the world of lightweight cryptography? Does it make sense to allow access to the complete set of matched inputs and outputs for an algorithm with a 128-bit block size? After all, the amount of data that we expect a single lightweight device to encrypt during its functional lifetime is tiny, and data to which an adversary has access will likely remain small when this tiny quantity is summed over all devices using a common key. In addition, for devices that can't be secured physically, practical (side-channel, reverse engineering) attacks will likely take precedence over cryptanalytic ones. The point is that there is a price to be paid (with every encryption) for blocking purely theoretical weaknesses, and it makes sense to think about what price is justified.

3. SIMON AND SPECK PARAMETERS

SIMON and SPECK are block cipher *families*. Each supports block sizes of 32, 48, 64, 96, and 128 bits, with up to three key sizes to go along with each block size. Each family provides ten algorithms in all. Table 3.1 lists the different block and key sizes, in bits, for SIMON and SPECK.

The 32-bit block size versions are intended for the most highly constrained applications, where only a minimal level of security is needed. The 48- and 96-bit variants are primarily intended for EPC applications. The 64-bit versions are meant for a variety of lightweight applications, and the 128-bit instantiations

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

Table 3.1: SIMON and SPECK parameters

are for applications where the highest security is required but where AES is unsuitable due to hardware or software constraints.

SIMON with a block size of k bits and a key size of r bits is denoted SIMON k/r. For example, SIMON 64/128 refers to the SIMON variant that uses a block size of 64 bits and a key size of 128 bits. Similar notation is used to denote the various SPECK block ciphers.

4. Performance comparisons

In this section we further discuss the performance of SIMON and SPECK, and fill in the details regarding the comparisons made in Table 1.1 with AES [DR02], EPCBC [YKPH11], KATAN [CDK09], KLEIN [GNL11], Piccolo [SIH+11], PRESENT [BKL+07], and TWINE [SMMK]. Note that while our algorithms are being considered for release, they are still under analysis, and so the numbers we present are subject to change: it is possible for them to improve slightly as better implementations are found, or perhaps get a bit worse if it is determined that some additional stepping is required for security.

It is important to note the difficulties inherent in the sort of comparison we're doing. Different authors implement their algorithms under differing assumptions: various cell libraries are used for hardware implementations, and a variety of assumptions are made for software implementations. In addition, it's not always clear what a particular author means, for example, by code size (is the decryption algorithm implemented or not?) or gate count (is the key schedule

included?). All of this can make attempts at a fair comparison problematic. That said, we believe the relatively large performance gap between our algorithms and others cannot fully be explained by these discrepancies.

In this paper we strive to make equitable comparisons, and to provide all the relevant details about our performance metrics and our implementation platforms. We begin by discussing the platforms a bit further.

The key hardware resources are circuit area and power. Area is measured in gate equivalents; a gate equivalent (GE), which depends on a particular cell library, is the physical area required for the smallest available two-input `NAND` gate.

Our results were generated using an ARM standard cell library for the IBM 8RF (0.13 micron) ASIC process. The areas of some basic gates in this library are as follows: `NOT` 0.75, `NAND` 1.00, `AND` 1.25, `OR` 1.25, `XOR` 2.00, `XNOR` 2.00, 2-1 `MUX` 2.25, D flip-flop 4.25, 1-bit full adder 5.75, scan flip-flop 6.25. (The existence of a smaller D flip-flop in this library than that found in libraries used by some authors—4.25 vs. 4.67—improves our numbers somewhat, but this does not fully explain our area advantages. We note also that our scan flip-flop is larger—6.25 vs. 6.0.)

Areas given for our algorithms are for *complete* implementations: this includes flip-flops to store the state and key, logic to implement the encryption algorithm and key schedule, control logic to manage the encryption, and logic to allow the plaintext to be loaded and ciphertext to be read out.[*]

In this paper, we report only area figures, and forgo detailed information about power. The reasons for this are (1) power consumption is strongly tied to the feature size, clock speed, etc., and this makes comparisons especially difficult, and (2) once these parameters are fixed, power scales roughly linearly with area, and so from area figures alone we can estimate power consumption if we know the scaling factor.[†]

Before we discuss Table 1.1 in detail, we point out the one slight mismatch in key sizes in the data presented there: neither SIMON nor SPECK is equipped with

[*]We have not included an implementation of the decryption algorithm in our area figures. This is consistent with other authors' work: for extremely lightweight applications one would want to use a block cipher in an encrypt-only mode.

[†]See Section 5, where we give a formula that correctly computes power as a function of area to within 25% for our simulations at 0.13 microns and 100 kHz.

a variant that uses an 80-bit key. In an attempt to draw the fairest comparison, we have lumped algorithms of size 64/96 together in the table with those of size 64/80.

Columns 3 and 4 of Table 1.1 compare SIMON and SPECK with some of the best performing block ciphers available for lightweight hardware applications. The hardware data for PRESENT and EPCBC is found in [YKPH11], for Piccolo in [SIH+11], for KATAN in [CDK09], for TWINE in [SMMK], for KLEIN in [GNL11], and for AES in [MPL+11].

Turning to software applications now, lightweight algorithms are typically implemented on inexpensive microcontrollers with very limited memory resources. ATMEL's ATtiny45 8-bit microcontroller, for instance, has just 4 kB of flash and 256 bytes of SRAM. Any cryptography on such a device competes with the application for scarce resources.

In addition, such microcontrollers usually run on battery power. Cryptographic components must limit power usage in the interest of extending battery life.

All of our software implementations were coded in assembly on an ATMEL ATmega128 8-bit microcontroller running at 16 MHz. Distinct implementations were done to minimize power consumption, flash usage, and SRAM usage. Results are presented in Section 6.

Table 1.1 compares *balanced*[*] implementations of SIMON, SPECK, and various well-known algorithms. For the latter, we used the best such implementation that we could find in the literature. The software data for PRESENT is derived from [EKP+07]; for KLEIN, KATAN, and AES from [EGG+12]; and for TWINE from [SMMK]. We note that the implementations given in these papers include the code for the encryption and decryption algorithms. Since our SIMON and SPECK code provides encrypt capability only (which is reasonable for lightweight applications), we have subtracted the size of the decryption code[†] from the numbers reported by those authors to obtain the numbers in our table.

[*]Balanced here means maximizing the ratio $\frac{\text{throughput}}{\text{flash} + 16 \cdot \text{SRAM}}$, amongst the implementations we considered.

[†]This was determined by examining assembly code whenever available: see [EGG+]. When the code was not available, we cut the specified flash usage in half as a reasonable estimate.

Alternatively, we could have included the decryption algorithms in our implementations. Naive implementations would have required another 100 bytes of flash or so, but by exploiting the similarity between the encryption and decryption algorithms we could reduce this number significantly.

We note that the best balanced implementations of our algorithms were in fact the SRAM-minimizing implementations. For these, code implementing the encryption algorithm is stored in flash, and the key is pre-expanded and also stored in flash; this obviates the need to include code for the key schedule and allows for high-throughput/low-power encryption. This is in contrast to the way in which the other algorithms handle the key: generally they include code for the key schedule, and generate round keys on the fly.

We conclude by highlighting a couple of comparisons from Table 1.1 between our algorithms and some other prominent algorithms.

- PRESENT-80 is a leading hardware-oriented lightweight block cipher, with an implementation requiring just 1030 GE and achieving throughput of 12.4 kilobits per second at 100 kHz. SIMON 64/96 and SPECK 64/96 (which provide 16 added bits of security) achieve even higher throughput at areas of just 854 and 984 GE, respectively. More importantly, our algorithms also have excellent software performance, and this is something that PRESENT was not designed to offer: SIMON 64/96 and SPECK 64/96 provide about five and nine times the throughput of PRESENT-80, respectively, for less than half the code size.

- AES is one of the best block ciphers for 8-bit microcontrollers. A good implementation of AES-128 on an ATmega 8-bit microcontroller requires 943 bytes of flash, 33 bytes of SRAM, and achieves a throughput of 445 kbps. SPECK 128/128 can achieve nearly 50% greater throughput while using less than 50% of the memory: SIMON 128/128 can come close to the throughput of AES, but at two-thirds the memory. More importantly, SPECK 128/128 and SIMON 128/128 can be realized in hardware for 63% and 53%, respectively, of the area required for AES-128.

  Perhaps most significantly, lightweight applications typically do not require a 128-bit block cipher: a 64-bit block cipher is perfectly adequate.

When SIMON 64/128 and SPECK 64/128 are compared with AES (which offers no 64-bit size), the difference is more compelling. The memory for our algorithms is about a quarter of that required by AES, and the throughput is 17% greater (SIMON) and 87% greater (SPECK). The hardware area drops significantly as well, to under half that required by AES.

5. Hardware data

This section presents detailed information on our ASIC implementations of SIMON and SPECK. To demonstrate the full flexibility of our algorithms in hardware, we have determined the area required for circuits at all levels of serialization for which efficient implementations are possible, together with the corresponding throughput values.

size	name	area (GE)	throughput (kbps)
32/64	SIMON	523	5.5
	SPECK	582	4.6
48/96	SIMON	738	5.0
	SPECK	794	4.2
64/96	SIMON	801	4.8
	SPECK	860	3.9
64/128	SIMON	939	4.5
	SPECK	997	3.6
96/96	SIMON	946	4.2
	SPECK	1012	3.4
128/128	SIMON	1218	3.2
	SPECK	1292	2.7

Table 5.1: Hardware performance: area-minimizing implementations

As we have pointed out, SIMON and SPECK admit bit-serialized hardware implementations. Table 5.1 shows that these have throughputs below the 12 kbps

threshold required for inclusion in Table 1.1. However, they have smaller circuit areas, which represent lower limits for the areas of hardware instantiations of our algorithms.

Table 5.2 presents all of our hardware data for SIMON and SPECK. For our implementations, at 0.13 microns and at a clock speed of 100 kHz, leakage power and switching power are negligible, each accounting for less than 2% of total power. Essentially all the power here is *cell internal power*, which is basically just the power required to keep the chip on. Empirically, the power in nanowatts (nW) for these implementations turns out to be approximately equal to 0.56 times the area in gate equivalents; this formula is accurate in all cases to within 25% (and in most cases to within 5%). For example, for SIMON 64/128 serialized at 4 bits per cycle, the cell internal power is 537 nW, the switching power is 10 nW, and the leakage power is 8 nW, for a total of 555 nW. The area is 1011 GE, and $0.56 \cdot 1011 \approx 566$, a prediction about 2% higher than the actual power consumption.

Table 5.2: Hardware performance for SIMON and SPECK

algorithm	**area (GE)**	**throughput (kbps)**	**algorithm**	**area (GE)**	**throughput (kbps)**
SIMON 32/64	523	5.5	SPECK 32/64	582	4.6
	535	11.1		640	9.0
	587	22.1		714	18.0
	666	43.8		824	35.9
	722	86.5		849	128.0
SIMON 48/72	630	5.5	SPECK 48/72	693	4.6
	663	11.1		752	9.3
	665	16.6		772	13.9
	701	22.1		818	18.6
	716	33.1		855	27.8
	764	44.0		950	37.0
	830	65.8		1036	55.2
	912	129.7		1151	200.0

Continued on next page

algorithm	area	throughput	algorithm	area	throughput
SIMON 48/96	738	5.0	SPECK 48/96	794	4.2
	777	10.0		856	8.3
	782	15.0		882	12.5
	824	19.9		931	16.6
	850	29.8		971	24.9
	908	39.7		1068	33.1
	993	59.3		1164	49.5
	1061	117.1		1254	177.8
SIMON 64/96	801	4.8	SPECK 64/96	860	3.9
	817	9.5		918	7.8
	854	19.0		984	15.7
	936	37.9		1090	31.2
	1084	75.3		1328	62.2
	1204	148.8		1521	228.6
SIMON 64/128	939	4.5	SPECK 64/128	997	3.6
	966	9.1		1057	7.1
	1011	18.1		1128	14.2
	1109	36.2		1245	24.5
	1288	71.9		1499	56.6
	1401	142.2		1658	206.5
SIMON 96/96	946	4.2	SPECK 96/96	1012	3.4
	983	8.3		1063	6.9
	985	12.5		1085	10.3
	1025	16.6		1126	13.8
	1037	24.9		1156	20.6
	1113	33.2		1254	27.5
	1143	49.7		1331	41.2
	1236	66.2		1512	54.9
	1356	99.0		1666	82.1
	1579	195.9		2057	309.7

Continued on next page

algorithm	area	throughput
SIMON 96/144	1147	3.9
	1184	7.8
	1187	11.8
	1226	15.7
	1239	23.5
	1315	31.3
	1345	46.8
	1438	62.3
	1557	93.2
	1781	184.6
SIMON 128/128	1218	3.2
	1235	6.4
	1274	12.9
	1352	25.8
	1503	51.4
	1751	102.4
	2094	203.2
SIMON 128/192	1493	3.0
	1513	6.1
	1550	12.1
	1629	24.2
	1780	48.3
	2032	96.2
	2373	191.0
SIMON 128/256	1776	2.9
	1800	5.9
	1846	11.8
	1943	23.5
	2122	46.9
	2408	93.4
	2770	185.5

algorithm	area	throughput
SPECK 96/144	1219	3.2
	1270	6.3
	1294	9.5
	1335	12.7
	1370	19.0
	1469	25.4
	1557	37.9
	1741	50.6
	1921	75.6
	2262	282.3
SPECK 128/128	1292	2.7
	1341	5.4
	1405	10.8
	1501	21.6
	1727	43.1
	2172	85.9
	2739	328.2
SPECK 128/192	1566	2.6
	1618	5.0
	1683	10.1
	1790	20.2
	2033	40.4
	2524	80.5
	3012	304.8
SPECK 128/256	1838	2.4
	1892	4.8
	1961	9.5
	2079	19.0
	2338	38.0
	2862	75.8
	3282	284.5

6. Software data: 8-bit microcontrollers

Typical microcontrollers have 4-, 8-, 16-, or 32-bit word sizes, and low-cost microcontrollers tend to have severe restrictions on available memory (flash and SRAM); power availability tends also to be constrained. While the software values presented in Table 1.1 represent balanced implementations, in this section we display our full data for minimizing SRAM usage, flash usage, and maximizing throughput.*

For the SRAM-minimizing implementations, code for the encryption algorithm is stored in flash. We do not implement the key schedule, instead storing pre-expanded round keys in flash. The encryption cost includes the time to load the key from flash into registers as needed.

For implementations maximizing throughput and for those minimizing flash, we again store the encryption code in flash. Here, however, we implement the key schedule and store the corresponding code in flash as well.† We then expand the key and store it in SRAM. The encryption cost counts only the cycles required for encryption, i.e., it does not include the cycles required for the generation or storage of expanded key in SRAM. This approach makes sense for many high-speed applications, where a large amount of data may need to be encrypted. We note that the key generation for SIMON or SPECK requires about as many cycles as a single encryption.

Tables 6.1, 6.2, and 6.3 show implementations that minimize SRAM usage, minimize flash usage, and maximize throughput, respectively. In each table, throughput is not shown directly, as it was in Table 1.1. Instead, we show the *encryption cost*, which is the number of cycles required for each encrypted byte, as this is the common metric used in the literature for software implementations.

*Throughput on a microcontroller is inversely proportional to energy, i.e., power integrated over time. So we effectively maximize battery life by minimizing the number of cycles required for an encryption, and therefore maximizing throughput.

†This convention does not always produce minimal flash implementations—for smaller versions of the algorithms, with fewer round keys, it can require less memory to store all the pre-expanded round keys in flash than it does to implement the key schedule. For consistency, however, we use the same approach for all implementations in a given table.

size	name	flash (bytes)	SRAM (bytes)	enc. cost (cycles/byte)
32/64	SIMON	114	**0**	217
	SPECK	92	**0**	137
48/72	SIMON	167	**0**	209
	SPECK	128	**0**	125
48/96	SIMON	176	**0**	228
	SPECK	134	**0**	137
64/96	SIMON	242	**0**	239
	SPECK	180	**0**	142
64/128	SIMON	246	**0**	245
	SPECK	188	**0**	154
96/96	SIMON	398	**0**	272
	SPECK	284	**0**	157
96/144	SIMON	410	**0**	284
	SPECK	296	**0**	168
128/128	SIMON	638	**0**	349
	SPECK	438	**0**	199
128/192	SIMON	662	**0**	366
	SPECK	454	**0**	210
128/256	SIMON	670	**0**	372
	SPECK	470	**0**	221

Table 6.1: Software implementations minimizing SRAM usage

Regarding the data in Table 6.3, we emphasize that the implementations were optimized for throughput without any regard for the other resources. For a *very slight* decrease in throughput, there are implementations of SIMON and SPECK that use significantly less flash. For instance, there is an implementation of SIMON 64/96 that requires around 404 bytes of flash, 156 bytes of SRAM, and has an encryption cost of 208 cycles/byte. That is, increasing the encryption

size	name	flash (bytes)	SRAM (bytes)	enc. cost (cycles/byte)
32/64	SIMON	**182**	64	208
	SPECK	**114**	40	162
48/72	SIMON	**180**	99	198
	SPECK	**130**	60	168
48/96	SIMON	**200**	108	216
	SPECK	**136**	66	185
64/96	SIMON	**198**	156	225
	SPECK	**156**	96	178
64/128	SIMON	**218**	160	230
	SPECK	**164**	104	193
96/96	SIMON	**214**	276	253
	SPECK	**196**	168	180
96/144	SIMON	**234**	288	264
	SPECK	**208**	180	193
128/128	SIMON	**250**	480	323
	SPECK	**244**	288	217
128/192	SIMON	**270**	504	339
	SPECK	**260**	304	229
128/256	SIMON	**290**	512	344
	SPECK	**310**	320	241

Table 6.2: Software implementations minimizing flash usage

cost by 1% lowers the flash usage by 87%. SPECK 64/96 has an implementation using just 366 bytes of flash (a decrease of 87%), 96 bytes of SRAM, and having an encryption cost of 108 cycles/byte (an increase of 2%). Many other trade-offs are possible.

Finally, none of the flash values we report include any *wrappers* necessary for an actual application to interface with the external world. In particular, we have

omitted the instructions and cycle counts associated with reading in data from the ports.

size	name	flash (bytes)	SRAM (bytes)	enc. cost (cycles/byte)
32/64	SIMON	1418	64	**177**
	SPECK	884	40	**112**
48/72	SIMON	2032	99	**176**
	SPECK	1078	60	**93**
48/96	SIMON	2226	108	**192**
	SPECK	1172	66	**101**
64/96	SIMON	3082	156	**205**
	SPECK	1612	96	**106**
64/128	SIMON	3178	160	**210**
	SPECK	1764	104	**116**
96/96	SIMON	5250	276	**238**
	SPECK	2732	168	**122**
96/144	SIMON	5494	288	**248**
	SPECK	2868	180	**128**
128/128	SIMON	8978	480	**308**
	SPECK	4516	288	**154**
128/192	SIMON	9442	504	**323**
	SPECK	4804	304	**163**
128/256	SIMON	9610	512	**328**
	SPECK	4980	320	**168**

Table 6.3: Software implementations minimizing encryption cost (power consumption)

References

[BKL+07] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, Lecture Notes in Computer Science, No. 4727, pages 450–66. Springer-Verlag, 2007. 7

[CDK09] C. D. Cannière, O. Dunkelman, and M. Knežević. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES 2009*, Lecture Notes in Computer Science, No. 5747, pages 272–88. Springer-Verlag, 2009. 7, 9

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, Berlin, 2002. 1, 7

[EGG+] T. Eisenbarth, Z. Gong, T. Guneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, F. X. Standaert, and L. van Oldeneel tot Oldenzeel. Implementatons of low cost block ciphers in Atmel AVR devices. http://perso.uclouvain.be/fstandae/lightweight_ciphers/. 9

[EGG+12] T. Eisenbarth, Z. Gong, T. Guneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, F. X. Standaert, and L. van Oldeneel tot Oldenzeel. Compact Implementation and Performance Evaluation of Block Ciphers in ATtiny Devices. In *Africacrypt 2012*, Lecture Notes in Compter Science. Springer-Verlag, 2012. 9

[EKP+07] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A Survey of Lightweight-Cryptography Implementations. In *IEEE Design & Test*, Volume 24, Issue 6, pages 522–33, 2007. 9

[GNL11] Z. Gong, S. Nikova, and Y.W. Law. KLEIN: A New Family of Lightweight Block Ciphers. In *RFIDsec '11 Workshop Proceedings*, Cryptology and Information Security Series, No. 6, pages 1–18. IOS Press, 2011. 7, 9

[JW05] A. Juels and S.A. Weis. Authenticating Pervasive Devices with Human Protocols. In *Advances in Cryptology–CRYPTO '05*, Lecture Notes in Computer Science, No. 3126, pages 293–308. Springer-Verlag, 2005. 2

[MPL+11] 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*, Lecture Notes in Computer Science, No. 6632, pages 69–88. Springer-Verlag, 2011. 2, 9

[SIH+11] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *CHES 2011*, Lecture Notes in Computer Science, No. 6917, pages 342–57. Springer-Verlag, 2011. 7, 9

[SMMK] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE: A Lightweight, Versatile Block Cipher. www.nec.co.jp/rd/media/code/research/images/twine_LC11.pdf. 7, 9

[YKPH11] H. Yap, K. Khoo, A. Poschmann, and M. Henricksen. EPCBC - A Block Cipher Suitable for Electronic Product Code Encryption. In *CANS 2011*, Lecture Notes in Computer Science, No. 7092, pages 76–97. Springer-Verlag, 2011. 7, 9

A. SIMON and SPECK on 64-bit Processors

As it turns out, and although this was not our primary aim, SIMON and SPECK have exceptional performance on 64-bit processors. Table A.1 reports the encryption cost for two different C implementations on a single core of a 2.93 GHz Intel Core i7-870 processor: a straightforward reference implementation with nothing done in parallel, and a high-speed SSE implementation. Both use a pre-expanded key, and so incur no cost for the key expansion.

The SSE versions of SPECK 64 and SPECK 128 carried out sixteen and eight parallel encryptions, respectively. The SSE versions of SIMON 64 and SIMON 128 were bit-sliced, each performing 128 parallel encryptions. The cost to transpose data for SIMON was not counted.

In each case our code was compiled using GCC version 4.6.1 with the -O3 flag set (except for the SSE versions of SIMON 128, which performed significantly better when compiled with GCC version 4.1.2 and the -O1 flag).

size	name	enc. cost (cycles/byte)	SSE enc. cost (cycles/byte)
64/96	SIMON	27.0	4.1
	SPECK	15.6	2.0
64/128	SIMON	27.0	4.2
	SPECK	16.2	2.2
128/128	SIMON	20.2	5.9
	SPECK	11.6	2.8
128/192	SIMON	20.7	6.2
	SPECK	12.3	2.9
128/256	SIMON	21.3	7.1
	SPECK	12.9	3.1

Table A.1: Encryption costs on a 64-bit processor