Exploiting Short Application Lifetimes for Low Cost Hardware Encryption in Flexible Electronics

Nathaniel Bleier[a], M. Husnain Mubarik[a], Suman Balaji[b], Francisco Rodriguez[b], Antony Sou[b], Scott White[b], and Rakesh Kumar[a]

[a]University of Illinois at Urbana-Champaign
[b]Pragmatic Semiconductor

Abstract—**Many emerging flexible electronics [1] applications require hardware-based encryption, but it is unclear if practical hardware-based encryption is possible for flexible applications due to stringent power requirements of these applications and high area and power overheads of flexible technologies relative to silicon CMOS technologies. In this work, we observe that the lifetime of many flexible applications is so small that often one key suffices for the entire lifetime. This means that, instead of generating keys and round keys in hardware, we can generate the round keys offline, and instead store these round keys directly on the engine post fabrication in an on-chip programmable read-only memory. This eliminates the need for hardware for dynamic generation of round keys, which significantly reduces encryption overhead, while still allowing engines to have unique keys. This significant reduction in encryption overhead allows us to demonstrate the first practical flexible encryption engines. To prevent an adversary from reading out the stored round keys, we scramble the round keys before storing them in the ROM; camouflage cells are used to unscramble the keys before feeding them to logic. In spite of the unscrambling overhead, our encryption engines consume** 27.4% **lower power than the already heavily area and power-optimized baselines, while being** 21.9% **smaller on average.**

I. Introduction

Flexible electronics [1] promise conformality and dramatically lower costs and have, therefore, seen an increased recent interest [2], especially in the context of applications with ultra-low-cost and conformality requirements that silicon-based electronics cannot meet. Hardware-based encryption, including strong encryption, is critical for many emerging flexible applications (Table I), including applications in the health domain. However, it is unclear if practical hardware-based encryption is possible for flexible applications since a) these applications often have stringent power requirements [2] (e.g., many flexible applications may need to be self-powered, a small $120\,\mathrm{mm}^2$ flexible battery provides a capacity of only $10\,\mathrm{mA\,h}$ [4], etc.) and b) flexible technologies have much higher area and power overheads than silicon CMOS technologies [2]. The only previous work on hardware-based encryption for flexible applications [5] had large area and high power (an implementation of the simple 32 bit block size, 80 bit key KTANTAN32 [6] encryption scheme took up $331\,\mathrm{mm}^2$!), making it clear that implementing even simple encryption schemes will be challenging in these technologies.

We observe that many flexible applications have a short lifetime and generate relatively small amount of data during their lifetime [2]. This means that only a small number of keys may be needed during the lifetime of the chip (Section III) - often only one key suffices. This eliminates any need for dynamic key generation or key expansion (that 'expands' the key into a larger key schedule, or sequence of round keys). Instead the round keys can be generated directly offline and programmed into the engine. This simple optimization — eliminating key expansion logic and replacing it by a *key schedule ROM* — reduces gate count of encryption significantly since key expansion circuitry is a significant fraction of encryption overhead (Section IV). For us, key schedules are stored in laser programmable read-only memories (LPROMs) which are programmed individually after chips are fabricated. Thus each chip can be programmed with unique key schedules. Since hardware is designed without any knowledge of the key or the round keys, constant propagation or other logic synthesis optimizations requiring knowledge of the key schedule are inapplicable to this design. To the best of our knowledge, **no prior work on hardware encryption eliminates key schedule generation and instead directly stores the key schedule in the encryption engine**. Closest related work [6] stores the key in mask ROM, but still uses the expensive key schedule generation logic. Also, use of mask ROM has the disadvantage that different chips share the key - so, all chips are rendered insecure if their shared secret (the key) is exposed (through microscope-based visual inspection, for example). For us, any exposure compromises only the individual chip since the LPROMs are programmed individually and not via lithography like a mask ROM. Further, active n-type electronics made with indium galium zinc oxide (IGZO) semicundoctor material supports sub-cent chip pricing [2], meaning replacing a compromised chip is inexpensive (and likely much cheaper than compromising the chip in the first place).

For a set of encryption algorithms (AES [7], PRESENT [8], Simon [9]), we show that encryption overhead can be reduced by 26.6-78.6%, -5.6 to 46.2%, 19.9 to 47.0% respectively in terms of gate count, area, and power through programming key schedules (i.e., round keys) directly. These benefits are over the already heavily area and power-optimized baseline implementations. The proposed AES-128 design, for example, requires only 1020 logic gates to implement, which is the lowest gate-count AES-128 implementation to the best of our knowledge. We fabricate and test three chip prototypes to confirm the benefits from eliminating key schedule generation and demonstrate **the first practical flexible encryption engines**. When we scramble the key schedule before storing it in the ROM (to prevent an adversary from the reading the keys) and use camouflaged cells to unscramble the key schedule before feeding it to logic, our encryption engines consume 27.4% lower power than the already heavily area and power-optimized baselines, while being 21.9% smaller on average, in spite of the unscrambling overhead.

TABLE I: The number of keys needed for representative flexible applications for block sizes of 32, 48, 64, and 128 bits while attempting to keep probability of cipher text collision below $p = 0.001$. A single key with 48 bit block size is sufficient for approximately half of the applications, while a single key with 64 bit block size is sufficient for all but a single application. Several applications are one-time use, and thus a 32 bit block size suffices.

Applications	Lifetime	Sample Rate (Hz)	Precision (bit)	Throughput ($\mathrm{bit\,s^{-1}}$)	Keys 32 / 48 / 64 / 128 bit blocks
ECG sensor	12 - 48 Hours / Single Use	100	12	1200	2235 / 6 / 1 / 1
Blood Pressure Sensor	24 Hours / Single Use	60	8	480	447 / 2 / 1 / 1
Electronic Nose - body odor	Hours	-	-	-	- / - / -
Heart Rate Sensor	12 - 48 Hours / Single Use	0.8 - 3.2	1	2	4 / 1 / 1 / 1
Body Temp. Sensor	Single Use	0.1 - 1	8	4.4	1 / 1 / 1 / 1
Smart Bandage	days	1	10	10	47 / 1 / 1 / 1
Sensor for PD detection	13-16 hours per day 14-20 Days / Single Use	50 - 100	13	650	8069 / 21 / 2 / 1
Oral-Nasal Airflow	2-4 Hours	16 - 25	8	164	26 / 1 / 1 / 1
Perspiration Sensor	2 Hours / Single Use	25	8	200	16 / 1 / 1 / 1
Pedometer	1 - 4 hrs	20	13	260	41 / 1 / 1 / 1
Human activity recognition	1 - 4 hrs	30	8	240	38 / 1 / 1 / 1
EEG sensor	8 Hours / Single Use	100-200	8	1200	373 / 1 / 1 / 1
Temperature Sensor (IoT/Biomedical)	Days / Hours / Single Use	< 1	8	8	23 / 1 / 1 / 1
Humidity Sensor	Days / Hours / Single Use	< 1	8	8	23 / 1 / 1 / 1
Shock Sensor	Days / Hours / Single Use	30	8	240	671 / 2 / 1 / 1
Tilt sensor	Days / Hours / Single Use	1-100	8	400	1118 / 3 / 1 / 1
Vibration Sensor	Days / Hours / Single Use	30	8	240	671 / 2 / 1 / 1
Light Sensor	Days / Hours / Single Use	< 1	16	8	23 / 1 / 1 / 1
Accelerometer	Days / Hours / Single Use	30	8	240	671 / 2 / 1 / 1
Wine quality sensor	Days / Hours / Single Use	18.5	8	148	414 / 2 / 1 / 1
Gas identification	depends upon use case	25	16	200	559 / 2 / 1 / 1
Pressure Sensor	Days / Hours / Single Use	< 1	8	8	23 / 1 / 1 / 1
Trace Metal Sensor	Single Use	25	16	400	1 / 1 / 1 / 1

II. Motivation

To understand the cost of implementing hardware-based encryption in flexible technologies, we synthesized RTL implementations of low-area AES, PRESENT, and Simon block ciphers using Pragmatic's experimental 3 V 0.6 µm IGZO TFT technology library. A 0.8 µm version of this technology has been used previously for characterization and fabrication of several prior flexible processing engines [2], [3]. We extended the standard cell library introduced in [2] with and-or-inverter and or-and-inverter cells for our evaluations.

Figure 1 shows that area overhead of encryption is significantly greater than 1 mm^2 (over 4 mm^2 for strong 128 bit encryption schemes). All designs consume at least 1 mW, except for the very weak 64 bit key version of Simon. All 128 bit baseline encryption engines consume at least 2 mW. These overheads are concerning since commercial Blue Spark flexible batteries [4] that support 10 mA h (2 mA peak current) and 30 mA h (2 mA peak current) will support only a small fraction of the desired lifetime for many applications in Table I for a 2 mW encryption engine. One could choose a Molex [10] 90 mA h (20 mA peak current) flexible battery instead. However, this battery has a vastly larger area footprint (43.2 cm^2!) than the smaller battery - this footprint will be unacceptable for many applications.

We also provide a breakdown of power and area between 'encryption' and 'key schedule'. The encryption portion consists of the logic needed to compute the round function, including the data block register, the round function logic, and the engine controller. The 'key schedule' portion consists of logic to generate the key schedule 'on the fly' (i.e., while running, rather than computing and storing the full key schedule before beginning encryption). We see that key schedule computation is the majority of engine area and power consumption for Simon and PRESENT, and a significant portion for AES.

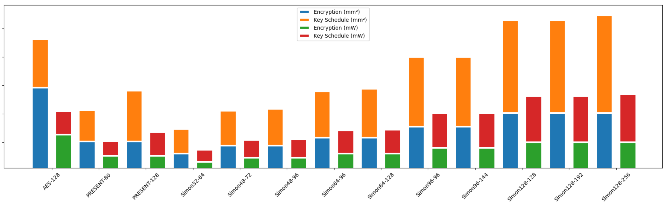


Fig. 1: Area and power consumption of baseline lightweight implementations of several block ciphers. AES-128 is a FIPS-197 compliant implementation of the Rijndael cipher using 128 bit blocks and keys. PRESENT-80 and PRESENT-128 are implementations of the PRESENT cipher (ISO/IEC 2167-11:2014) with 64 bit blocks and 80 bit and 128 bit keys, respectively. SimonX-Y are implementations of the Simon cipher (ISO/29167-21) with X bit blocks and Y bit keys.

We focus on block ciphers rather than stream ciphers due to their lower overhead. The internal state of a block cipher is equal to the block size, while the internal state of stream ciphers is often very large (e.g., 8288 bits for ISAAC, 19 968 bit for CryptMT, 65 536 bit for HC-256, 1216 bit for MUGI, 576 bit for SNOW 2.0, etc). Even 'lightweight' stream ciphers have significantly more internal state than the block ciphers implemented in this paper. E.g., Trivium has 288 bit, Salsa20 has 512 bit, etc. In comparison, the proposed AES-128 design has 140 bit of total state, including 12 bit used for its controller. Although some stream ciphers with small internal states have been used, these have been shown to be severely insecure (e.g., A5/1, A5/2, Crypto-1).

III. Flexible Applications Require Few Keys

The number of keys required by an application depends on the amount of data it needs to encrypt during its lifetime and is determined by the minimum number required to prevent a *birthday attack* with sufficient likelihood. A birthday attack is an attack on a block cipher which is enabled by 'cipher text collisions' — when two different plain text blocks are encrypted to the same cipher text block. For example, in the basic electronic code book (ECB) mode, a collision reveals that the related plain texts are equivalent, since $c_i = E_k(p_i)$ and thus $c_i = c_j$ implies $p_i = p_j$. Likewise, in cipher block chaining (CBC) mode, where $c_i = E_k(c_{i-1} \oplus p_i)$, $c_i = c_j$ implies $c_{i-1} \oplus p_i = c_{j-1} \oplus p_j$ and thus $p_i \oplus p_j = c_{i-1} \oplus c_{j-1}$. If all of the cipher texts are stored, an adversary can thus extract the XOR of the two plaintexts. Further, if using the counter (CTR) mode of operation, where $c_i = E_k(C_i \oplus p_i)$ where C_i is an incrementing counter concatenated with a secret nonce, cipher text collisions enable *distinguishing attacks* which enable an adversary to distinguish the cipher text from uniformly random strings.

Figure 2 shows how much data can be transmitted using a single key before the probability of a collision exceeds a threshold as a function of block size. This is calculated by the function $d(p,b) = \sqrt{2 \cdot 2^b \log \frac{1}{1-p}}$ where p is the collision threshold probability, b is the block size in bits, and d is the number of blocks encrypted before the probability of collision exceeds p. This formula is derived from a commonly used exponential estimate of the probability of collision [11]:

$$p(d,b) \approx 1 - e^{-d(d-1)\left(2 \cdot 2^{b}\right)} \approx 1 - e^{-d^{2}/\left(2 \cdot 2^{b}\right)}.$$

The figure shows that the number of keys needed is strongly dependent on both the block size and the application's resilience to cipher text collisions. Table I shows the number of keys needed for each application for various block sizes under the strong assumption that probability of collision should be limited to $p = 0.001$.

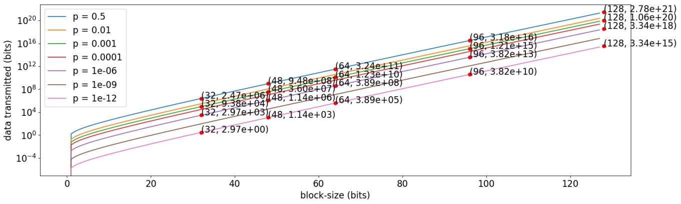


Fig. 2: Number of bits of data encrypted before the probability of block collision exceeds p as a function of block size. Curves are based on exponential approximation of collision likelihood.

We see that our example flexible applications need no more than a single key for a 128 bit block size. All but one applications need one key even for 64 bit block block size. More keys are needed for smaller block sizes. The fact that only a small number of keys are needed for flexible applications can be leveraged to significantly reduce encryption costs.

IV. Exploiting Small Number of Keys for Reduced Encryption Overhead

Since only a small number of keys are needed (no more than one for block sizes of 64 and 128), there exists a surprisingly simple, but effective opportunity. It is no longer essential to generate keys (or round keys) online using key expansion logic that today's encryption engines have. Instead round keys can be generated offline and directly programmed into an on-chip 'key schedule ROM' post fabrication. Since key expansion overhead is high (Figure 1), replacing this key schedule generating hardware by a 'key schedule ROM', which stores the entire key schedule could lead to significant reduction in encryption overheads, while still allowing individual chips to have unique keys.

Figure 3 shows the relative area and power of encryption engines with key schedule stored in a laser-programmed ROM (LPROM) (charactersitics in Table II) vs generated online using dedicated key schedule generation hardware.

TABLE II: Measured characteristics of IGZO-TFT LPROMs. The power and area for the peripheral logic and LPROM are based on a 512 bit LPROM with 64× 8 bit words.

	Area (μm^2)	Power (μW)
Bit Cell	328.91	0.33
Read Power bit^{-1}	-	2.81
Peripheral Logic	162130	225
LPROM (512 bit)	330531	394

We see significant area and power benefits from this simple optimization ($> 25\%$ area reduction and $> 30\%$ power reduction, on average). Recall that these benefits are reported over the already heavily area and power-optimized baseline implementations. An LFSR can be used to throttle transmissions in the event that the device lifetime exceeds the expected application lifetime.

We note that using a small, fixed number of keys increases susceptibility to other cryptanalytic attacks which require collecting large amounts of cipher text data generated by a single key. However, such attacks typically require collecting more data than the birthday bound itself. For example, the best published attack on AES-128 has time complexity of $2^{126.18}$ and space complexity 2^{88} [12]. Thus, storing sufficient number of keys to ameliorate birthday attacks is sufficient to prevent many other cryptanalytic attacks.

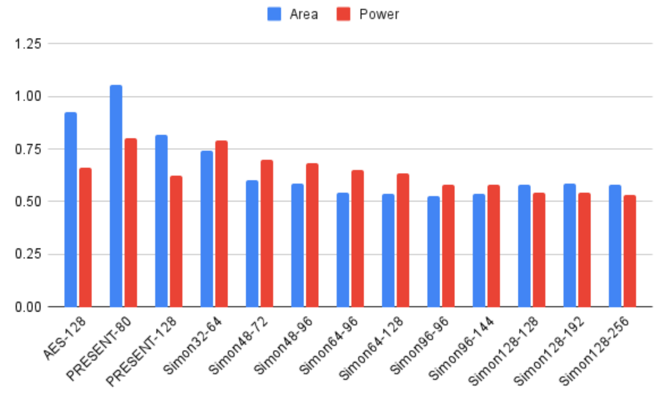


Fig. 3: Relative area and power of encryption engines with key schedule implemented in LPROM vs with dedicated key expansion hardware.

V. Architectures

Architectures for the proposed AES-128, PRESENT, and Simon encryption engines (with their key expansion logic replaced by a key schedule ROM) are shown in Figure 4.

Figure 4a depicts Simon's architecture, which implements a single instance of Simon's round function. As a Feistel cipher, Simon splits its data block into 'L'eft and 'R'ight halves. Each cycle, a single round is executed which shifts the R register into the L register, and loads the R register with a simple function of the L register and the key schedule.

Figure 4b depicts PRESENT's datapath. In order to minimize cell count, PRESENT is serialized to 4 bit, the maximum serialization allowed by its substitution function (SBox). Each round consists of 16-cycles in which the four most significant bits of the block register, D, are summed with the key schedule (XOR), then substituted through the SBox, and then shifted back into the least significant bits of D. After these 16 cycles, a single cycle is used to perform PRESENT's permutation, which simply permutes all of the 64 bits in the data block.

Figure 4c depicts AES-128's datapath. Like with PRESENT, the datapath is maximally serial, with most operations acting on a single octet. The data block is depicted broken into four columns of four octets. Depending on the stage of AES-128 execution (i.e., AddRoundKey, SubBytes, ShiftRows, MixColumns) one of several dataflows are used. During AddRoundKey, data from octet $a_{0,3}$ is added with key schedule data, and the sum is routed into $a_{3,3}$, while the rest of column three is vertically rotated. After an entire column has been summed with the round key, the black and *dashed* routes (to bypass MixColumns) are used. A similar dataflow occurs for SubBytes. MixColumns acts on an entire column at once, and its dataflow is shown with the solid black lines. ShiftRows uses the solid and dashed black routes.

VI. Prototyping Results

To both confirm the benefits from eliminating key expansion logic and to demonstrate the first practical encryption engines in a flexible technology, we designed and fabricated prototypes of AES-128 (128 bit block and key sizes), PRESENT (64 bit block size, 80 bit or 128 bit key sizes), and Simon (48 bit block size, 96 bit key size) on 9 mm^2 dies using Pragmatic's 0.6 µm IGZO-TFT technology. We only had a 512 bit LPROM macro available during fabrication, so multiple LPROMs were needed to store the key schedules.

Table III shows the area of logic and LPROM of the prototyped encryption engines. Both of the lightweight block ciphers (Simon, PRESENT) use fewer than 500 cells and take area $< 1.25\,\text{mm}^2$. AES-128 is significantly larger, at 1715 cells and $> 3.7\,\text{mm}^2$. However, the control and datapath components, excluding the SBox which can be encoded into LPROM, uses only 1020 cells and 2.65 mm^2 area.

TABLE III: Cell count and area of prototypes after SP&R.

		Area (mm^2)		
Engine	NAND2 Equiv.	Logic	LPROM	Total
AES-128	2576	3.71	0.93	4.64
AES-128 (no SBox)	1840	2.65	2.17	4.82
PRESENT	868	1.25	1.24	2.49
Simon48-64	854	1.23	0.62	1.85

The prototype dies were all manufactured on a single 200 mm polyimide wafer. After fabrication, chips were tested on a semi-automated wafer probe station MPI TS2000 while still attached to a glass carrier. Yield is not significantly affected when chips are removed from the carrier. Test patterns derived from SystemVerilog simulation were translated to input signals generated by a NI PXIe-6570 Digital Pattern Instrument. Output signals, captured by the same instrument, were compared against the SystemVerilog simulation outputs. The chips were tested with LPROMs storing key schedules from published test vectors for the ciphers. The tests consist of encrypting several plain text blocks (including published test vectors) several times. We consider a chip to yield if there is an average of less than one error per block encrypted. A small number of errors, n, can be handled by encoding a checksum or nonce into each plain text block. If the decrypted cipher text does not contain that checksum or nonce, then the decrypter can decrypt a small number of related cipher texts whose Hamming distance from the error-prone cipher text is $\leq n$. Thus, small number of errors does not prevent the encryption engines from being used practically. The chips were tested at 3 V and 4.5 V. Simon yields 70% and 71% at 3 V and 4.5 V respectively, with all working chips having 0 errors. PRESENT yields 38% and 51% at 3 V and 4.5 V, with several dies being error-free. AES-128 yields 47% at both 3 V and 4.5 V with all working chips being error-free. It is expected that yields will be improved following optimization of the 0.6 µm technology, which is an experimental process node at time of wafer production.

At 3 V, Simon current draw is under 1 mA and is on average 0.6 mA, increasing to an average of 1 mA at 4.5 V. Average current draw for PRESENT is 0.7 mA and 1.1 mA at 3 V and 4.5 V respectively. Thus, at 3 V, the prototyped Simon48-96 engines draw only 1.8 mW on average, while PRESENT draws only 2.1 mW. The AES-128 prototype draws 4.5 mW on average at 3 V. Even these small numbers are, in fact, artificially high in the prototypes. Since the LPROMs in Simon and AES-128 (PRESENT) is currently broken between three (four) macros, each macro duplicates expensive peripheral logic which accounts for the majority of the LPROM's power consumption. Storing the key schedules in a single larger LPROM would thus lead to significant power savings.

Our PRESENT, Simon, and AES-128 prototypes are the first demonstration of practical encryption engines in a flexible technology, and can be used with numerous block cipher modes of operation, including ECB, CBC, PCBC (for encryption only), and GCM, CTR, CFB, and OFB (for encryption and decryption).

VII. Key Schedule Pre-scrambling

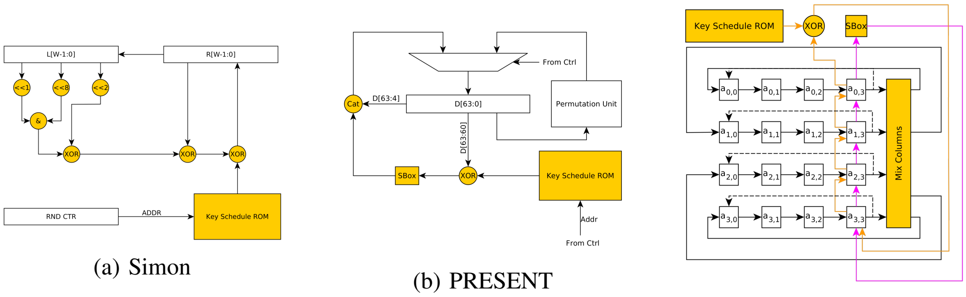
Flexible technologies have coarse feature sizes (0.6 µm for our evaluations) which may enable an adversary to read the stored key schedules using a microscope (Figure 6), thereby compromising security.

To deal with the above vulnerability, we pre-scramble the key schedule before storing in on-chip LPROM. On-chip unscrambling logic built using camouflage cells [13] is used to unscramble the key schedule before it is used by the encryption pipeline. In addition, key schedule can be chosen randomly (e.g., from a uniform distribution) to maximize security.

Figure 7 shows the unscrambling hardware for a hypothetical system with 32 bit key schedule and an 8 bit datapath. Scrambled key schedule data is read from the LPROM ($L_7,\ldots,L_0$) and broadcast to four 8 bit arrays of camouflaged cells. The camouflaged cells, which are functionally either XOR2 or XNOR2, have one input tied to VDD and the other to the appropriate broadcast signal. Hence the camouflaged cells are wired to act as either buffers or inverters, and the output of each array is unknown to an adversary, even when the contents of the LPROM are known. The four camouflaged buses are then multiplexed by the subround key mux, whose output is the unscrambled 8 bit subround key.

Figure 8 gives the power and area overheads for key schedule unscrambling hardware for combinations of key size and datapath width (we estimate 1.5x overhead for camouflage cells). Table IV shows the power, area, and energy per encrypted block for different block ciphers when both replacing key expansion logic and key unscrambling hardware is supported. In spite of the unscrambling overhead, our encryption engines consume 27.4% lower power than the power and area-optimized baselines and are 21.9% smaller.

Physical unclonable functions (PUFs) are an alternate approach to securely storing keys on IoT devices, but often require high overhead error correction coding (ECC) due to

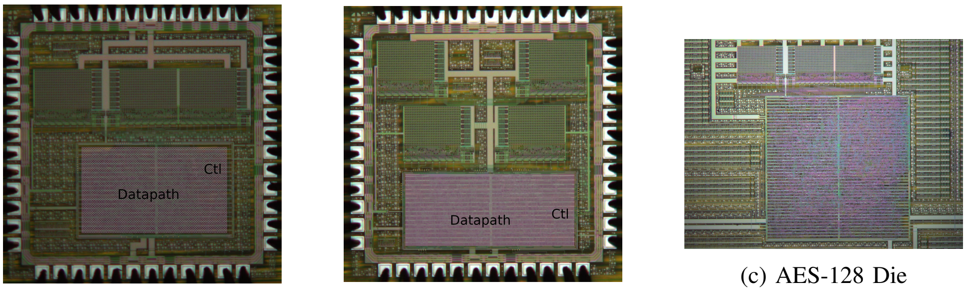


(a) Simon

(b) PRESENT

(c) AES-128

Fig. 4: Architectures for Simon, PRESENT, and AES-128.



(a) Simon48-96 Die

(b) PRESENT Die

(c) AES-128 Die

Fig. 5: Die photos of the three prototyped encryption engines.

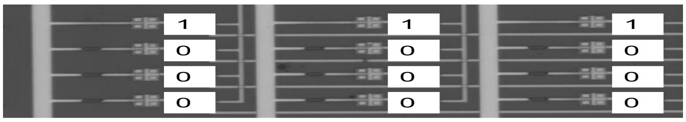


Fig. 6: Our in-house optical inspection revealed the data stored in LPROM.

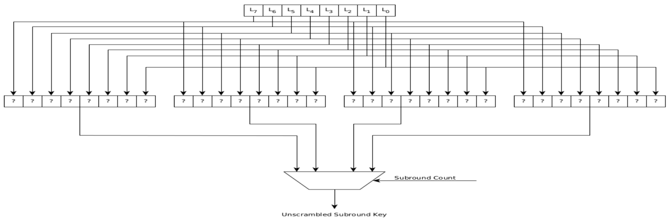


Fig. 7: Key Schedule Unscrambling Hardware

variability in PUF output (high variability exists for flexible electronics [17]) - securely storing a 128 bit key with high reliability may require thousands of stored bits due to the ECC overhead [16]. This overhead would be unacceptable for flexible applications. In fact, actual overhead for us would be much higher since we directly store not key, but key schedule, which is typically much larger.

Even disregarding ECC overheads, PUF overheads may be high for securely storing key schedule. Since delay-based PUFs outputs only a single bit, either a large number of PUFs would be needed, or the block ciphers would need to be bit-serialized, presenting a significant trade-off between area and performance/energy. This is especially significant in Simon, since Simon's round-function is designed to support inexpensive single-cycle execution. Thus, Simon would require $\frac{\textit{block size}}{2}$ PUFs to maintain its performance characteristics. We estimate that using 24× arbiter PUFs would add $> 80\%$ logic area overhead to the fabricated Simon48-96 design.

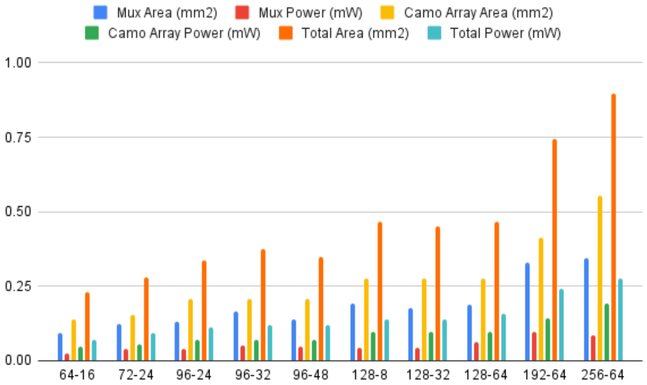


Fig. 8: Area and power overhead of key schedule unscrambling hardware. Columns for `X-Y` refer to encryption engines with `X` bit keys and `Y` bit datapaths (e.g., AES-128 implementation is `128-8`).

VIII. System-level Impact

To estimate system level (Figure 10) benefit of replacing key expansion logic with key schedule ROM, we consider three different flexible system configurations, System01-System03. System01 adds a flexible encryption engine to a flexible RFID chip [14]. System02 adds a flexible gas sensor [15] to system01. System03 adds a flexible microprocessor [2] to System02. The power consumption of different flexible components of the systems is presented in Table V. Power consumption of the encryption engines with and without optimizations are in Figure 1 and Table IV respectively. Once we have the total power consumption of the baseline and the optimized systems, we calculate the time for which a system can be powered by a given battery assuming that all the components of the system are always active. Figure 9 shows that, on average, the lifetime is increased by 27.05%, 26.07% and 18.94% for system01,

TABLE IV: Power, area, and latency of AES-128, Simon, and PRESENT block cipher implementations with KS ROM and Key Schedule Pre-scrambling applied. Percentage difference is against baselines with neither KS ROM nor Key Schedule Pre-scrambling.

Latency (Cycles)		Area		Power		Energy	
	Cipher	mm^2	% Δ	mW	% Δ	μJ	% Δ
512	AES-128	4.31	-6.7	1.67	-20.4	85.3	-20.4
527	PRESENT-80	2.63	23.1	.95	-8.2	50.14	-8.2
527	PRESENT-128	2.71	-3.5	.97	-28.3	51.13	-28.3
32	Simon32-64	1.31	-10.3	.64	-11.3	2.06	-11.3
36	Simon48-72	1.54	-26.5	.84	-21.5	3.03	-21.5
36	Simon48-96	1.6	-26.1	.86	-21.3	3.1	-21.3
42	Simon64-96	1.88	-32.4	1.03	-26.5	4.35	-26.5
44	Simon64-128	2.	-30.6	1.05	-26.7	4.64	-26.7
52	Simon96-96	2.45	-38.8	1.29	-36.4	6.7	-36.4
54	Simon96-144	2.71	-32.2	1.35	-33.2	7.3	-33.2
68	Simon128-128	3.52	-33.4	1.58	-39.8	10.75	-39.8
69	Simon128-192	3.09	-41.6	1.42	-45.7	9.83	-45.7
72	Simon128-256	4.08	-25.3	1.7	-36.7	12.25	-36.7

TABLE V: Power consumption of the non-encryption components of the systems under study.

Component	Description	Power uW
RFID [14]	An a-IGZO TFT-based RFID chip	20
FlexiCore4 [2]	A low gate count flexible microprocessor	700
Sensor [15]	A gas detection sensor	65

system02 and system03 respectively through the use of our optimization.

IX. Summary and Conclusions

Many emerging flexible electronics applications require hardware-based encryption. However, it is unclear if hardware encryption engines can be developed at an acceptable overhead for flexible technologies due to high area and power overhead of such technologies. In this work, we presented a simple, but extremely effective optimization - eliminating key schedule generation logic and replacing it by a laser programmable key schedule ROM storing a small set of round keys, often a singleton set - enabled by short lifetime and low data emission rate of flexible applications. This optimization significantly reduces encryption overhead while allowing individual chips to have programmable unique keys, and allows us to demonstrate the first practical flexible encryption engine chip prototypes. To hide the stored key schedule from adversaries, we scramble the key schedule offline before being stored in the ROM, while camouflage cells are used to unscramble the key schedule before feeding it to logic. Our key schedule ROM-based encryption engines consume 27.4% lower power than the power and area-optimized baselines and are 21.9% smaller in spite of the unscrambling overhead.

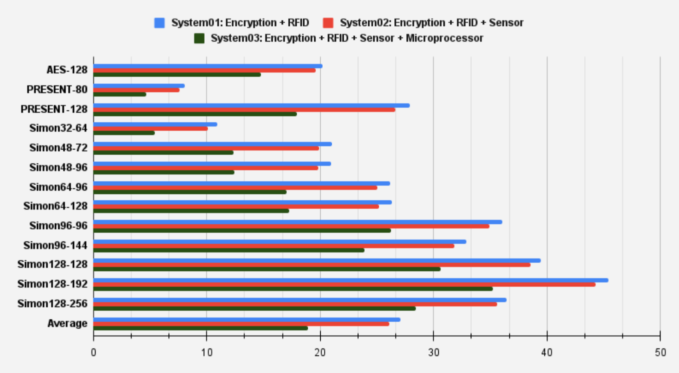


Fig. 9: Percentage increase in the lifetime of different systems due to the proposed encryption engine optimizations.

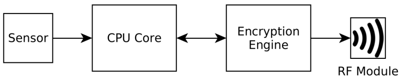


Fig. 10: An example flexible system encrypts processed biomedical data and then transmits the encrypted data.

X. Acknowledgments

We thank Emre Ozer from Pragmatic Semiconductor for help with AES-128 prototyping results, reviewers for their feedback, and NSF (CCF-2006763) for their partial support.

References

[1] W. Wong and A. Salleo, “Flexible Electronics: Materials and Applications”, Springer Science & Business Media, New York, NY, USA, 2009.

[2] N. Bleier et al., “FlexiCores: low footprint, high yield, field reprogrammable flexible microprocessors”, Proceedings of the 49th Annual International Symposium on Computer Architecture, pp. 831–846, June 2022.

[3] J. Biggs et al., “A natively flexible 32-bit ARM microprocessor”, Nature 595.7868 (2021), pp. 532-536.

[4] Blue Spark Technologies, “Thin-Film Battery”, 2015.

[5] N. Mentens et al., “Security on Plastics: Fake or Real?”, IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 1–16, August 2019.

[6] C. Cannière and O. Dunkelman and M. Knežević, “KATAN and KTANTAN—a family of small and efficient hardware-oriented block ciphers”, International Workshop on Cryptographic Hardware and Embedded Systems, pp. 272–288, September 2009.

[7] M. Dworkin et al., “Advanced Encryption Standard (AES)”, Federal Inf. Process. Stds. (NIST FIPS), National Institue of Standards and Technology, November, 2001.

[8] A. Bogdanov et al., “PRESENT: An ultra-lightweight block cipher” International Workshop on Cryptographic Hardware and Embedded Systems, pp. 450–466, September 2007.

[9] R. Beaulieu et al., “The SIMON and SPECK Lightweight Block Ciphers”, Proceedings of the 52nd Annual Design Automation Conference, pp. 1–6, June 2015.

[10] Molex, “Molex - Thin-Film Battery”,

[11] M. Sayrafiezadeh, “The Birthday Problem Revisited”, Mathematics Magazine, pp. 220–223, June 1994.

[12] A. Bogdanov and D. Khorvratovich and C. Rechberger, “Biclique cryptanalysis of the full AES”, International Conference on the Theory and Applicatoins of Cryptology, pp. 344-371, December, 2011.

[13] R. Cocchi and J. Baukus and L. Chow and B. Wang, “Circuit camouflage integration for hardware IP protection”, 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–5, June 2014.

[14] H. Ozaki and T. Kawamura and H. Wakana and T. Yamazoe and H. Uchiyama, “20 μW operation of an a-IGZO TFT-based RFID chip using purely NMOS active load logic gates with ultra low consumption power”, 2011 Symposium on VLSI Circuits - Digest of Technical Papers, pp. 54–55 2011.

[15] V. Misra et al., “Ultra-low power sensing platform for personal health and personal environmental monitoring”, 2015 IEEE International Electron Device Meeting, pp. 13.1.1–13.1.4, December, 2015.

[16] C. Bösch, and J. Guajardo and A. Sadeghi and J. Shokrollahiand P. Tuyls, “Efficient helper data key extractor on FPGAs”, International workshop on cryptographic hardware and embedded systems, pp. 181–197, September 2008.

[17] A. Erozan et al., “Inkjet-printed EGFET-based physical unclonable function—Design, evaluation, and fabrication”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 2935–2946, 2018.