Efficient Implementation of Cryptographic Primitives on the GA144 Multi-core Architecture

Tobias Schneider, Ingo von Maurich, and Tim Güneysu
Horst Görtz Institute for IT-Security
Ruhr University Bochum
Germany
{tobias.schneider-a7a,ingo.vonmaurich,tim.gueneysu}@rub.de

***Abstract*—With myriads of small and pervasive devices in our digital age, the availability of low-power and energy-efficient processing technology has become absolutely essential. Most of these constrained devices need to incorporate security services for confidentiality and privacy in addition to their primary tasks – typically involving computationally expensive cryptography. In the last years, many researchers have worked on novel lightweight cryptographic constructions to minimize the computational burden on the constrained devices. However, most of those alternative constructions sacrificed security for simplicity, potentially enabling just as simple attacks. In this work, we aim for another approach and implement standardized and well-established cryptography on a special but very lightweight platform, namely an asynchronous GA144 ultra-low-powered multi-core processor with 144 simplistic cores. For the first time, we demonstrate that symmetric and asymmetric cryptography such as AES and RSA is even feasible on such a low-end and unclocked device. With energy consumption being as low as 0.63 μJ and 22.3 mJ, this platform achieves a performance of 38 μs and 462.9 ms per AES and RSA operation, respectively. Both energy consumption as well as computation time are significantly lower than many lightweight implementations reported so far.**

I. Introduction

The advent of ubiquitous computing has raised the challenge to integrate very complex computational services on (nearly) invisible and small devices. Moreover, not all these ubiquitous devices have a static power-supply so that the energy-efficiency of computations on such battery-buffered systems are of utmost importance. This particularly applies to computationally expensive security services which are often required to ensure confidentiality and privacy of user data.

In the context of public-key cryptography, for example with thousands and millions of multiplications involved, it has become a major research branch to adapt and optimize advanced cryptosystems to small systems, such as RFID devices. Many works focused on the lightweight adaption of cryptographic algorithms to clocked hardware, often sacrificing security on this level or using very low clock frequencies.

In this work, we highlight a different approach and map standardized and established cryptography to a very lightweight asynchronous processor, namely the GA144 by GreenArrays, Inc. Our core interest for this research is determined by the open question if there are more efficient processor platforms for cryptographic operations than the ones recently developed and distributed by companies such as Intel, AMD, ARM, AVR, and others.

A. Contribution

In this paper we present the first implementation of the standardized symmetric AES block cipher and the asymmetric RSA cryptosystem on a GA144 device. The GA144 is an asynchronous multi-core processor with 144 simplistic cores and a different computational concept to maximize energy-efficiency.

We demonstrate that even with very few capabilities of the given cores (e.g., a maximum of 64 words of program memory per core) it is possible to operate even complex cryptosystems at moderate speeds with very restricted energy requirements.

More precisely, our results show that symmetric and asymmetric cryptography such as AES and RSA is not just feasible on such a low-end and unclocked device – but with energy consumption being as low as 0.63 μJ and 22.3 mJ, this platform achieves a decent performance of 38 μs and 462.9 ms per AES and RSA operation, respectively. Both energy consumption as well as computation time are significantly lower than many lightweight implementations reported so far.

B. Related Work

The GA144 is a very atypical and new processor. Therefore, hardly any other implementations are available for this platform. GreenArrays published their own implementation of the hash function MD5 and announced a pipelined implementation of the SHA-256 hash-function which is about to be released. Therefore, we can only compare our results against other processor platforms and ASIC implementations running AES and RSA as presented in the results section.

C. Organization

This work is organized as follows: we first describe the novel architecture of the GA144 and explain its programming paradigm in Section II. Next, we present both our implementation of AES and RSA in Sections III and IV, respectively. Finally, we present our results in Section V, before we outline future work and draw conclusions in Sections VI and VII.

II. Processor Architecture of the GA144

In the following we give a brief description of the internal structure of GreenArrays GA144 multi-core processor. We start by introducing the F18A computing node as the foundation of the processor, explain how 144 of these cores are interconnected to form the GA144 and give a short intuition on how to program these chips.

A. The F18A Computing Node

When designing the F18A, GreenArrays aimed to create a small and inexpensive computing node which is both fast and energy-efficient. The processor operates on a word size of 18 bit and is equipped with 64 words of RAM and ROM. Furthermore, every F18A possesses stacks, registers, and (optionally) various kinds of external I/O [5].

The processor operates completely asynchronously from other connected devices. While basic instructions are executed in about 1.5 ns without instruction fetches, memory operations can take up to 5 ns. Since one of the main optimization goals of the processor is energy-efficiency, these executions only cost 7 pJ and the F18A can be suspended even in mid-instruction. When suspended, a small leakage of about 100 nW keeps the power consumption at a minimum [7].

The two available stacks are a ten-element data stack and a nine-element return stack. The top two elements of the data stack are represented by **T** and **S** and the remaining eight form a circular memory. The return stack is build similarly, the top of the stack is stored in register **R** and the following eight elements are stored in so-called circularly addressed registers. Additionally, every F18A has an instruction register **I**, a program counter **P**, a general-purpose read/write register **A**, and a write-only register **B** [5].

There are four different ways for a F18A to communicate with external devices. These are not available for every computing node and are realized depending on its position on the GA144 chip. Speaking in major classes of I/O, a F18A can utilize general purpose input/output (GPIO), analog I/O, 18-bit parallel buses, and a high speed serializer/deserializer (SerDes) [8].

The data flow between F18A nodes is realized using communication ports which enable the nodes to exchange data at high speed. Additionally to exchanging data, instructions can be transferred over these ports to accomplish port execution. This means the receiving F18A executes them directly and, since the program counter is not incremented, a whole instruction stream can be processed in short time. Moreover, this technique can be utilized to save program memory in the receiving node because no additional code is required for port execution. In the most extreme case a node has no program code at all and just waits for instructions to be fed to it from the outside. As a result the program memory of this node is empty and can be used to store other data. Another application for this is reprogramming a F18A during runtime by rewriting the program memory using port execution and then jumping to an address to start the program.

B. The GA144 Multi-core Processor

The GA144 consists of 144 F18A nodes which are arranged in an 18x8 array. The F18As are connected with their adjacent nodes and 22 computers are equipped with special I/O capabilities. This structure has the advantage to allow massive parallel computations distributed over a high number of nodes. It also benefits the energy consumption, since unused nodes can be suspended to minimize the overall consumption. Figure 1 depicts the layout of the GA144 processor.

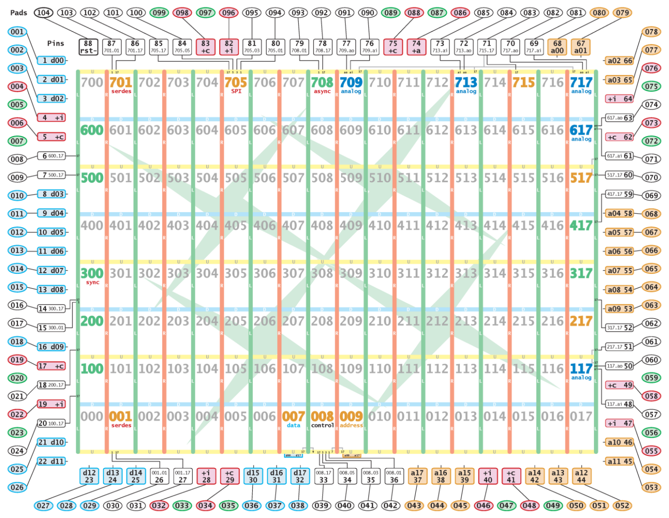


Fig. 1. Layout of the GA144 chip, consisting of an 18x8 array of F18A computing nodes [6].

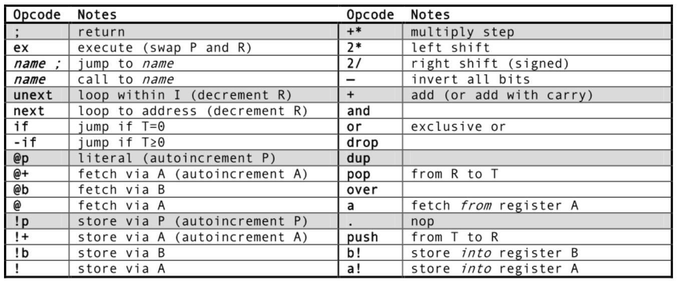
There is a wide field of possible applications for the GA144 chip. The low power consumption makes the GA144 predestined for mobile devices and sensors with a limited energy supply as an extended battery life is of high importance in such applications. Moreover, the high computational power opens the field to more complex control systems and research applications. Due to a low price of around 20 USD the GA144 could also be applied in a cluster with multiple chips.

C. Programming the Chip

The programming language for the GA144 is a Forth-derivate called *colorForth* in which different colors give different meanings to parts of the program code. For example, comments have their own color, as have addresses and labels. Note that Forth itself follows a completely different programming paradigm compared to conventional programming languages such as C or Java and that the coloring is part of the programming as opposed to the syntax highlighting in other languages.

The instruction set for this platform features 32 5-bit opcodes. A 18-bit word consists of three 5-bit slots and one 3-bit slot. Therefore, up to four opcodes can fit into one word. Since the last slot is only 3 bits wide, only eight particular opcodes can be used. These are all the opcodes of which the two least significant bits are zero and therefore are ignored. Figure 2 lists the F18A's instruction set.

The major challenge for a programmer is the strong memory constraint for each node. With only 64 18-bit words in each node larger applications need to be split up and distributed to multiple nodes. This however also offers the opportunity to exploit inherent parallelism in the applications and possibly allow for an improved computation time.



Opcode	Notes	Opcode	Notes
;	return	+*	multiply step
ex	execute (swap P and R)	2*	left shift
name ;	jump to *name*	2/	right shift (signed)
name	call to *name*	–	invert all bits
unext	loop within I (decrement R)	+	add (or add with carry)
next	loop to address (decrement R)	and	
if	jump if T=0	or	exclusive or
-if	jump if T≥0	drop	
@p	literal (autoincrement P)	dup	
@+	fetch via A (autoincrement A)	pop	from R to T
@b	fetch via B	over	
@	fetch via A	a	fetch *from* register A
!p	store via P (autoincrement P)	.	nop
!+	store via A (autoincrement A)	push	from T to R
!b	store via B	b!	store *into* register B
!	store via A	a!	store *into* register A

Fig. 2. The instruction set of the F18A processor consisting of 32 5-bit opcodes [5].

III. Implementing AES on the GA144

The Advanced Encryption Standard (AES) is the most widely used symmetric block cipher and was designed to be efficiently implementable both in hardware and in software. Although it is a well-established block cipher, we briefly review its structure to give rationales for our design choices for the GA144.

A. The Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) block cipher uses a block size of 128 bit and supports key sizes of 128, 192, and 256 bit. During en-/decryption the input block is run multiple times through three different layers. The *Key Addition Layer* and the *Diffusion Layer* apply linear transformations, the non-linear element is the *Byte Substitution Layer* [16].

During *Key Addition* a 128-bit roundkey is XORed to the state. The roundkeys are derived from the main key by a key derivation function.

The *Diffusion Layer* consists of two sub-layers. First, the *ShiftRows Layer* applies a row-wise byte rotation to the state. Afterwards, the bytes are passed to a matrix multiplication in tuples of four bytes within the *MixColumn Layer*.

The *Byte Substitution Layer* substitutes all bytes with their (slightly modified) multiplicative inverse in $\mathbb{F}_{2^8}$ and is usually implemented as table-lookup (S-box) of 256×8-bit entries.

B. Design Considerations

Since memory is very scarce in each node of the GA144 it is necessary to spread the tables and the entire encryption over multiple nodes. A natural approach is to split the AES computations by its separate layers. However, while *Key Addition* and *ShiftRows* are simple operations and fit into one node, this does not apply for the *Substitution Layer* and the *Key Schedule*.

We decided to implement the *Substitution Layer* as straightforward 8-bit look-up table. One F18A node can store up to 64 18-bit words. Since the S-box table has 256 8-bit entries, a minimum of $\lceil \frac{256 \cdot 8}{64 \cdot 18} \rceil = \lceil 1.77 \rceil = 2$ nodes are required to store the table. Therefore, we divided the table into two halves and stored two 8-bit entries in one 18-bit word in the program memory. Thus, two nodes are required to store the complete table. The two tables take up the complete program memory of both nodes so that there is no space for additional control. Therefore, a third node is utilized to handle the table lookups appropriately.

The *MixColumn Layer* fits into one node. However, as detailed in the next section, we found that distributing the task over four nodes yields a much better overall performance. Each node mixes one of the state columns using shift and XOR operations to realize polynomial multiplications.

The *Key Schedule* is split over six nodes. One node permanently stores the main key. Another node holds the currently required roundkey and delivers it to the *Key Addition Layer* while a third node schedules the next roundkey. Since the *Substitution Layer* is used when scheduling a roundkey and due to the fact that the key schedule is computed in parallel to the cipher, a second instance of the *Substitution Layer* needs to be added.

As described in the next section, the overall structure of the cipher is very compact and does not need many control or routing nodes. Only one extra node is needed to realize the I/O and control the implementation.

C. Implementation Aspects

The basic layout for the AES encryption does not change for different key lengths. It can be reused when switching to another key length by making minor changes to the layer nodes and some major changes to the key schedule. In total our implementation occupies 17 out of the 144 available nodes.

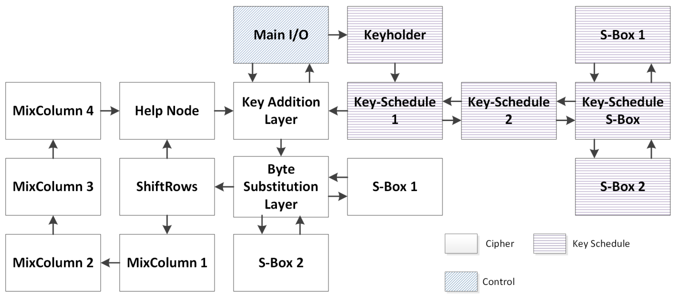


Fig. 3. Data flow diagram of the AES encryption (same for all key sizes).

As shown in Figure 3 the implementation can be divided into three parts: Control, Key Schedule, and Cipher. Each part communicates with the other two throughout the encryption process. The information flow is depicted by the arrows between the nodes, the start-up node being Main I/O. The I/O part is only represented by one node because there is no need for more in our current scenario. This can be easily extended to fulfill more challenging demands. In contrast to the small I/O part, the other two make up the majority of the implementation. The Key Schedule consists of six nodes, while the Cipher occupies nine F18As.

The Main I/O node starts the encryption. It activates the Key Schedule and the Cipher and is also the last node to be active after the encryption is done. Before or after an encryption, Main I/O signals the Key Schedule to reset its roundkeys. Afterwards, the Cipher is started and the plaintext is send to the Key Addition Layer. Then Main I/O waits for the cipher to finish, accepts the ciphertext and stores it for further use. This marks the end of the encryption and all nodes are suspended until the next encryption is started.

The key schedule can be divided into three different parts: the KEYHOLDER, the KEY SCHEDULE NODES 1 and 2, and the KEY SCHEDULE S-BOX. The actual key schedule is performed by the KEYSCHEDULE NODES 1 and 2 with the help of the KEYSCHEDULE S-BOX. The current roundkey is held in KEY SCHEDULE 1 and send to the KEY ADDITION LAYER in each round. Since every designated key length has a key schedule which is in some ways a bit different than the other two, the key schedule has to be adjusted if the length of the key changes.

The cipher itself consists of three different layers as explained before, which are implemented in nine nodes. The HELP NODE'S task is to either forward the result of the *MixColumn Layer* to KEY ADDITION LAYER or skip the *MixColumn Layer* in the last round by redirecting the output of SHIFTROWS to the KEY ADDITION LAYER.

Through simulation we acquired performance figures for the separate layers and compared them among each other to find possible bottlenecks. In the first version of our implementation we realized the *MixColumn Layer* in one node. It turned out that it consumed 40% more time than second slowest layer which was the *Byte Substitution Layer*. Therefore, we decided to distribute the *MixColumn Layer* over four nodes to improve its performance. This helped to reduce the overall time consumption of the layer by 70%.

Our AES decryption implementation uses a similar layout as the encryption. A main difference is the necessary precomputation of the roundkeys. This procedure needs additional nodes and raises the total number of nodes from 17 to 21. The main differences between decryption and encryption are a reorganized key schedule and a added possibility to store the roundkeys. Furthermore, the layers are inverted and the order of them is changed as well.

D. Multiple Instances of AES

Since only 17 out of 144 nodes are used for one AES encryption unit, it is possible to place multiple instances on one chip and run them simultaneously. However, there are a few constraints on how and where to place these instances. For a start, each instance has to have a connection to external I/O nodes of the GA144 to be able to communicate with the outside world. Furthermore, the internal structure of the nodes obviously needs to be preserved.

A possible layout with six instances of the AES encryption unit is shown in Figure 4. Some instances need extra nodes to reach the I/O nodes, so a total of 112 nodes instead of $17 \cdot 6 = 102$ are occupied in this design.

In case of multiple decryption instances one has to keep in mind that the extended key schedule requires more nodes for the decryption unit. Thus, the same layout cannot be reused. Again, the same constraints as in case of the encryption unit hold. A possible layout of five decryption instances on the GA144 occupies 120 nodes. This is one instance less than in case of encryption but had to be expected. Moreover, the same technique as before was applied to connect the MAIN I/O node

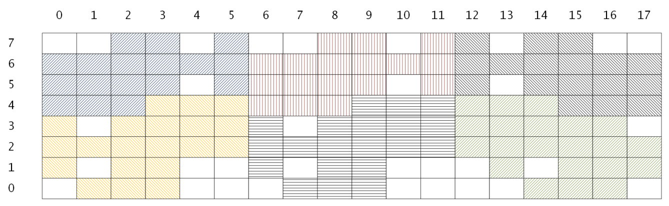


Fig. 4. Distribution of six AES encryption instances on a GA144 chip.

of each instance with the external I/O of the chip to ensure functionality.

IV. IMPLEMENTING RSA ON THE GA144

The RSA cryptosystem is still one of the most widely used asymmetric cryptographic scheme. Its security is based on the integer factorization problem which is believed to be hard to solve for appropriately chosen parameters. In the following we briefly summarize RSA and give design rationales for the GA144.

A. The RSA Asymmetric Encryption/Signature Scheme

The main operation used in RSA is exponentiation in the integer ring $\mathbb{Z}_m$ where $m = p \cdot q$ represents the modulus. The public key $pk = (e, m)$ consists of an exponent e and a modulus m. The secret key $sk = (d, p, q)$ consists of the inverse of the encryption exponent $d = e^{-1} \ mod \ \phi(m)$ and the prime factors p and q. For secure applications the parameters are at least 1024-bit in size which makes the modular exponentiation operations very complex.

To encrypt a plaintext g the sender computes $c = g^e \ mod \ m$ using the receivers e and m. The receiver decrypts c by calculating $g = c^d \mod m$. Digital signatures can be created using the same procedure by switching the roles of the exponents.

B. Design Considerations

In contrast to AES, RSA does not iterate over a series of layers but uses modular arithmetic with large parameters (≥ 1024 bits). This again is a particular problem due to the scarce memory in each node. For our design, we chose to use RSA-1024 with parameters of length 1024 bit. Thus, one RSA parameter fits exactly into the memory of one F18A node.

For exponentiation we use the standard square-and-multiply algorithm [15] in combination with a word-based version of the Montgomery multiplication algorithm [15] which works with words of 16 bits. The core multiplication is implemented in serial-parallel fashion, i.e., multiplying 1024 bits with one word. The multiplier itself is implemented as systolic array [12]. Figure 5 shows a subset of the basic multiplication structure.

The systolic array consists of MULTIPLICATION and STORAGE UNIT pairs. They interact with each other to compute their corresponding results by propagating carries. The MULTIPLICATION UNITS have the task of calculating a multiplication of a one-word factor with multiple words supplied from the STORAGE UNITS. Each of the STORAGE UNITS holds three words of the multiplier, of the intermediate result, and of the

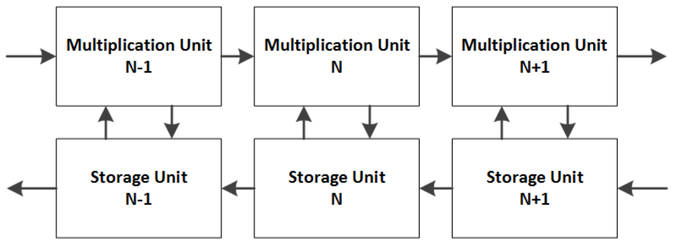


Fig. 5. An example extract of the systolic multiplication array.

modulus m. During the steps of the *Montgomery Product*, they send the required parts to their corresponding MULTIPLICATION UNIT, receive the result and store it. Moreover, they operate independently of the MULTIPLICATION UNITS at the end of the product by shifting the final result one word to the right.

At the beginning of the multiplication the factor is fed to the MULTIPLICATION UNITS from the left. It is copied and forwarded through the whole array so that all units receive a copy. Then the actual multiplication is computed in parallel in all nodes and afterwards the carries are exchanged with the next MULTIPLICATION UNIT in the array. This has the advantage of speeding up the process by allowing parallelism. For RSA-1024 a total of 22 storage units is required since

$$22 \cdot 3\ words \cdot 16\ \frac{\text{bits}}{\text{word}} = 1056\ bits > 1024\ bits.$$

Note, intermediate results during the *Montgomery product* cannot exceed 1048 bit at any moment, thus overflows during the multiplication do not occur.

C. Implementation Aspects

Our RSA-1024 implementation takes up 107 nodes on the GA144. As mentioned before it consists of several different elements that are combined to form the complete cipher. The *Montgomery multiplication* is the largest component and occupies 55 nodes, whereas the 1024-bit modulus reduction requires 15 nodes. The remaining nodes are used to implement the MAIN CONTROL, storage nodes for the parameters, an extended Euclidean algorithm, I/O, and connection nodes.

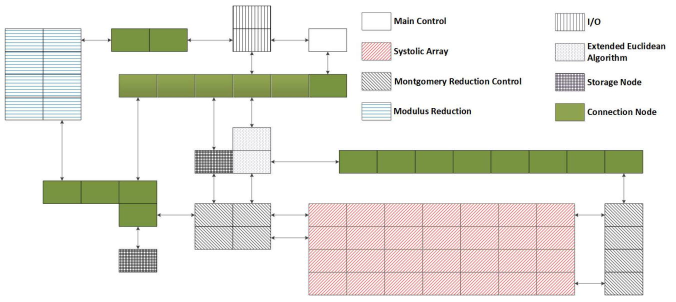


Fig. 6. Simplified data flow diagram of our RSA implementation.

Figure 6 depicts a simplified diagram of our implementation. It shows the connections between the fundamental parts of the encryption. The parts are reduced and do not consist of the actual number of occupied nodes. Every process is controlled by the MAIN CONTROL node at the top which is also the last node to be active after the computation is finished.

The RSA parameters can be set from the outside using the I/O CONTROL node which forwards the parameters to the desired storage nodes. Additionally, the result of the exponentiation can be made accessible from the outside by transferring it to the I/O MEMORY. This procedure is utilized after the RSA operation is finished in order to retrieve the output.

The task of the MAIN CONTROL unit is to send out commands to the surrounding nodes. It is accessed through an asynchronous connection and supplies the outside with simple functions to control the flow of the cipher. Especially the I/O CONTROL node is activated by it to set the parameters of the exponentiation.

Since the RSA implementation takes up more than half of the available nodes on the GA144, instantiating multiple units as shown for AES is not feasible.

V. RESULTS

In the following we present our results and measurements of the execution time and power consumption and compare them to other published results on similar platforms.

We measure the power consumption and execution time using a digital oscilloscope (PicoScope 5203 [1]) with two channels. *Channel A* is set as trigger and *Channel B* is used to measure the voltage over a $18\,\Omega$ resistor inserted into the groundpath. Immediately before the encryption starts the trigger pin is set and held until the end of the encryption. Thus, the computation time of the cryptosystems can be extracted by taking the time between a rising and a falling edge of the trigger signal. In addition, we extract the power consumption of the chip from *Channel B*, which we use in combination with the execution time to compute the overall energy consumption.

Note, the performance of an asynchronous processor is influenced by its supply voltage. Thus, we measure and analyze the implementations for three different supply voltages (1.8 V, 2.0 V, 2.2 V).

A. AES Results

For each supply voltage we average our measurements over 10,000 AES en-/decryptions. The keys and plaintexts change after every encryption and the resulting ciphertext is used as a source of randomness for the next key and plaintext. Our results are summarized in Table I.

TABLE I. EXECUTION TIME AND ENERGY CONSUMPTION OF AES-128,-192,-256 EN-/DECRYPTION ON THE GA144.

Supply voltage	AES mode	Encryption			Decryption		
	Key size [bit]	128	192	256	128	192	256
1.8 V	Time [μs]	45.1	51.4	64.1	74.2	85.2	110.6
	Energy [μJ]	0.63	0.65	0.86	0.97	1.14	1.53
2.0 V	Time [μs]	41	49.6	56	72.4	83.3	100.2
	Energy [μJ]	0.77	0.74	1.078	1.36	1.57	1.86
2.2 V	Time [μs]	38.0	44.3	52.7	66.6	76.7	93.7
	Energy [μJ]	0.90	0.92	1.26	1.35	1.60	2.19

AES encryption with 128-bit keys is the fastest and most energy efficient operation mode. This is due to the fact that this variant executes the smallest number of rounds and thus needs less operations to complete. In comparison to encryption, decryption takes more time and energy which can be explained by the required roundkey pre-computation and the additional computations for the inverse layers.

With a rising supply voltage, the execution time of all modes decreases while the overall energy consumption rises. Interestingly, the energy consumption only slightly differs between the 128-bit and the 192-bit encryption. At a supply voltage of 2.0 V the 192-bit mode even outperforms the 128-bit mode in terms of energy-efficiency.

B. RSA Results

The RSA implementation is measured in a similar manner, averaging over 1,000 RSA operations. We change the modulus after every operation and the exponent and the message after every second operation. Note, RSA encryption and decryption use the same exponentiation unit. Hence, their runtimes are similar and only depend on the chosen parameters. Furthermore, we decided to select full-size random parameters for a fair comparison and do not use short exponents to speed up the encryption. Our results are presented in Table II.

TABLE II. EXECUTION TIME AND ENERGY CONSUMPTION OF RSA EN-/DECRYPTION ON THE GA144.

Supply voltage	1.8 V	2.0 V	2.2 V
Time $[ms]$	598.1	513.7	462.9
Energy $[mJ]$	22.3	24.6	27.9

Again, with a rising supply voltage the execution time is decreased at the cost of an increased energy consumption.

C. Comparison

In the following we compare the time and energy consumption of our implementations of AES and RSA to implementations on similar embedded platforms.

Since one field of application are portable devices and remote sensing, we compare our AES implementation with an implementation on a *MicaZ* sensor node for a wireless sensor network (WSN). It is a widely-used 2.4 GHz board equipped with an 8-bit Atmel AVR ATmega128L microcontroller clocked at 8 MHz and is especially designed for low-power applications. In [20], a AES software implementation as well as a hardware-supported AES implementation (using the Chipcon CC2420 RF transceiver chip) are evaluated on this platform.

A comprehensive survey of the performance and energy-efficiency of several block ciphers (among them AES) on an inexpensive, low-power Atmel AVR ATtiny45 8-bit microcontroller is given in [3].

In [10] and [11], ASIC implementations of AES aiming for a low power consumption in WSN and RFID applications are presented. We include these results to show the gap between microcontrollers and special-purpose hardware implementations.

Additionally, we compare our implementation with a recent AES implementation [13] for a multi-core processor architecture prototype that offers 167 cores [17].

Since the above mentioned implementations focus on AES encryption with a 128-bit key, we compare them against our single-instance AES-128 encryption at a supply voltage of 2.2 V. The comparison is given in Table III.

TABLE III. COMPARISON OF THE TIME AND ENERGY CONSUMPTION OF AES-128 ENCRYPTION ON EMBEDDED PLATFORMS.

Platform	Time $[\mu s]$	Energy $[\mu J]$
GA144 (0.18 μm)	37.9	0.90
AVR ATtiny45 [3]	455.7	19.2
MicaZ software [20]	885.8	19.16
MicaZ HW-assisted [20]	350.6	26.82
Many-core prototype (65 nm) [13]	0.126	0.198
ASIC (0.13 μm) [11]	2140	0.051
Low-power ASIC (0.13 μm) [10]	1.23	0.005

The AES implementations on the ATtiny45 and the MicaZ microcontroller are clearly outperformed by our implementation. In terms of speed, AES encryption runs 12-23 times faster on the GA144. Even when compared to a hardware-accelerated AES available on the MicaZ board, our implementation still executes a AES encryption more than 9 times faster. Note, in this comparison we restrict ourselves to a single instance of AES. As stated before, up to six instances of the AES-128 encryption can run in *parallel* on a single GA144 which would further increase the GA144's advantage.

Moreover, the energy consumption of AES on the GA144 is 21 times less compared to AES software implementations on the microcontrollers. Using the Chipcon CC2420 hardware-accelerated AES even consumes 30 times more energy than the GA144.

A recent AES implementation for the AsAP many-core processor array prototype occupies 39 out of 164 programmable cores. The processor is fabricated using a considerably smaller CMOS technology (180 vs. 65 nm) and its cores appear to be more powerful than the F18As. It seems to be a promising architecture with fast execution times and a low energy consumption with the drawback of only being a prototype as compared to the freely available GA144. Note, the AsAP result are reported for a fixed key implementation.

When comparing our results to special-purpose hardware designs it becomes clear, that even with a low-power processor and carefully optimized implementations, the performance and energy consumption can not keep up. The advantages of an ASIC naturally come at the cost of much less flexibility, higher development costs and longer development cycles.

RSA implementations have been reported for similar microcontroller platforms as in the case of AES. In [9] and [19], RSA implementations for the Mica2 and the MicaZ WSN platform are presented. Furthermore, a RSA implementation for an AVR ATmega128 is given in [14]. Unfortunately, the energy consumption is only reported for the implementation on the Mica2 platform. Since the same microcontroller is used by all implementations, we estimate the energy consumption of [14] and [19] by linear interpolation taking into account the different execution time and clock frequency.

TABLE IV. COMPARISON OF THE TIME AND ENERGY CONSUMPTION OF RSA ENCRYPTION ON EMBEDDED PLATFORMS. *THE ENERGY CONSUMPTION IS LINEARLY INTERPOLATED BASED ON THE MICA2 RESULTS.

Implementation	Time $[ms]$	Energy $[mJ]$
GA144 (0.18 μm)	513.7	24.6
Mica2 [9] [18]	10,990	304
MicaZ [19]	7,900	218.5*
AVR ATmega128 [14]	4,730	283.7*
DSRCP ASIC (0.25 μm) [4]	32.1	2.41
RCP ASIC (0.13 μm) [2]	3.84	1.75

Special-purpose reconfigurable cryptographic processors (RCP) are the counterpart to the microcontroller approach. Explorations on how to design an energy-efficient ASIC capable of performing RSA operations are given in [2] and [4].

All of the above mentioned implementations run RSA-1024, so they are a good fit for comparison. Table IV lists time and energy consumption of our RSA implementation at a supply voltage of 2.0 V and the reported results on other embedded platforms.

A similar behavior as in the case of AES can be observed when comparing RSA on a GA144 to other microcontroller platforms. An RSA operation is performed 9 - 21 times faster and consumes 12 times less energy at the same time. Specially designed ASIC implementations of RSA however are much faster and consume around a tenth of the energy.

Overall the GA144 chip turns out to be a good choice for WSN and RFID applications when compared to other low-power microprocessor for both ciphers, since it operates faster and is more energy efficient at the same time. Its performance and energy consumption lies in between common microcontrollers and special-purpose ASICs.

VI. FUTURE WORK

In the following we describe possible optimizations that could be evaluated in future implementations. They reach from a complete reorganization of the implementation structure to local changes of certain layers.

A. AES – Extended Byte Substitution Layer

As described before, the *Byte Substitution Layer* of AES is the implementation's bottleneck. In order to further improve the overall performance, the *Byte Substitution Layer* could be instantiated twice. With two instances running in parallel, the time consumption of the layer would be halved since the state bytes can be substituted independently of each other. Figure 7 shows a possibility to implement a second *Byte Substitution Layer* with four additional nodes, raising the total number of nodes to 21.

B. AES – Pipelined Implementation

A pipelined implementation of the AES encryption would reduce the time consumption if a large number of blocks has to be en-/decrypted. Every stage of the cipher would occupy a node on the GA144 and the roundkey would be fixed in this scenario. Figure 8 shows a possible solution to implement a pipelined AES-256 encryption on the GA144.

The main advantage of this implementation is achieved when a large number of blocks is consecutively en-/decrypted. The first block will arrive with a latency comparable to a non-pipelined implementation, but all following blocks will be processed much faster. This is possible since their computation can start immediately after the prior block has left the first stage.

A disadvantage are the fixed roundkeys, which require the chip to be reprogrammed if the key changes. Moreover, a different I/O setting has to be taken into consideration. The input to the cipher will be given from one side, iterate through the whole chip and will be available on the other side.

C. RSA Optimizations

In this work we make use of two algorithms to improve the performance of RSA. The Word-Level Montgomery Multiplication is already an efficient solution, but for the exponentiation part other algorithms could be evaluated. The Sliding Window algorithm [15] for example splits the exponent into small windows and uses precomputed values to speed up the exponentiation process. This requires additional nodes to store the results of the precomputation but is likely to increase performance.

Additionally, the Chinese Remainder Theorem (CRT) can be used if the prime factors p and q of m are known. The CRT splits the exponentiation into two separate instances based on the prime factors of the modulus which are only half the size of m. The computational complexity gets reduced by a factor of nearly 4 with this method but it is also infamous for enabling easy to apply fault injection attacks that can compromise the secret key.

VII. CONCLUSION

In the course of this paper we introduced the GA144 chip and its corresponding hard- and software. The main aspect of this work is the proposal of a novel implementation of AES and RSA for the chip.

We showed that it is possible to implement even very computationally expensive ciphers efficiently. Comparing performance and energy consumption to similar implementations on other devices, it turns out that our implementations on the

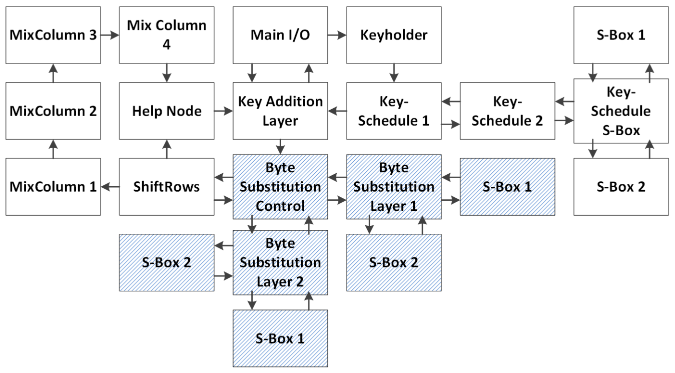


Fig. 7. Data flow diagram of an AES encryption with extended Byte Substitution Layer.

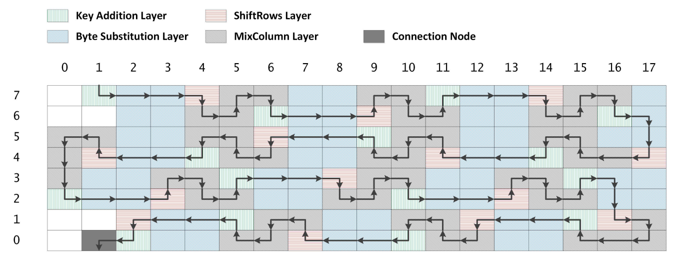


Fig. 8. A pipelined implementation of the AES-256 encryption on the GA144.

GA144 are very efficient and outperform their counterparts on other low-power devices.

We conclude from these results that the GA144 can be a powerful sensor node with a long lifespan, even if security services are required. This implication can be transferred to other mobile devices, since their requirements are very similar.

Furthermore, we hope to trigger future research in the area of evaluating cryptographic algorithms on novel processor platforms such as the one presented in this work.

Acknowledgment

This work was supported in part by grant 01ME12025 SecMobil of the German Federal Ministry of Economics and Technology.

References

[1] PicoScope 5200 Professional Oscilloscopes. http://www.picotech.com/picoscope5200.html, as of April 21, 2013.

[2] J.-H. Chen, M.-D. Shieh, and W.-C. Lin. A high-performance unified-field reconfigurable cryptographic processor. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(8):1145–1158, 2010.

[3] T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, et al. Compact implementation and performance evaluation of block ciphers in ATtiny devices. *Progress in Cryptology-AFRICACRYPT 2012*, pages 172–187, 2012.

[4] J. Goodman and A. P. Chandrakasan. An energy-efficient reconfigurable public-key cryptography processor. *Solid-State Circuits, IEEE Journal of*, 36(11):1808–1820, 2001.

[5] GreenArrays. DB001 - F18A Technology Reference. http://www.greenarraychips.com/home/documents/greg/DB001-110412-F18A.pdf, as of April 21, 2013.

[6] GreenArrays. DB002 - G144A12 Chip Reference. http://www.greenarraychips.com/home/documents/greg/DB002-110705-G144A12.pdf, as of April 21, 2013.

[7] GreenArrays. PB003 - F18A Computers. http://www.greenarraychips.com/home/documents/greg/PB003-110412-F18A.pdf, as of April 21, 2013.

[8] GreenArrays. PB004 - F18A I/O and Peripherals. http://www.greenarraychips.com/home/documents/greg/PB004-110412-F18A-IO.pdf, as of April 21, 2013.

[9] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 925–943, 2004.

[10] P. Hamalainen, T. Alho, M. Hannikainen, and T. D. Hamalainen. Design and implementation of low-area and low-power AES encryption hardware core. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 577–583. IEEE, 2006.

[11] J.-P. Kaps and B. Sunar. Energy comparison of AES and SHA-1 for ubiquitous computing. *Emerging Directions in Embedded and Ubiquitous Computing*, pages 372–381, 2006.

[12] H. Kung and C. Leiserson. Algorithms for VLSI processor arrays. *Introduction to VLSI systems*, pages 271–292, 1980.

[13] B. Liu and B. Baas. Parallel AES encryption engines for many-core processor arrays. *Computers, IEEE Transactions on*, 62(3):536–547, 2013.

[14] Z. Liu, J. Großschädl, and I. Kizhvatov. Efficient and side-channel resistant RSA implementation for 8-bit AVR microcontrollers. In *Workshop on the Security of the Internet of Things-SOCIOT*, 2010.

[15] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC, 1996.

[16] NIST. FIPS PUB 197: Advanced encryption standard. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, as of April 21, 2013.

[17] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, A. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, A. Tran, Z. Xiao, E. Work, J. Webb, P. Mejia, and B. Baas. A 167-processor computational platform in 65 nm CMOS. *Solid-State Circuits, IEEE Journal of*, 44(4):1130–1144, 2009.

[18] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 324–328. IEEE, 2005.

[19] H. Wang and Q. Li. Efficient implementation of public key cryptosystems on mote sensors (short paper). *Information and Communications Security*, pages 519–528, 2006.

[20] F. Zhang, R. Dojen, and T. Coffey. Comparative performance and energy consumption analysis of different AES implementations on a wireless sensor network node. *International Journal of Sensor Networks*, 10(4):192–201, 2011.