Characterising and Comparing the Energy Consumption of Side Channel Attack Countermeasures and Lightweight Cryptography on Embedded Devices

David McCann, Kerstin Eder, Elisabeth Oswald
University of Bristol
Department of Computer Science
Merchant Venturers Building, Woodland Road, BS8 1UB, Bristol, UK
{David.Mccann, Kerstin.Eder, Elisabeth.Oswald}@bristol.ac.uk

No Institute Given

Abstract. This paper uses an Instruction Set Architecture (ISA) based statistical energy model of an ARM Cortex-M4 microprocessor to evaluate the energy consumption of an implementation of AES with different side channel attack (SCA) countermeasures and an implementation of lightweight ciphers PRESENT, KLEIN and ZORRO with and without Boolean first order masking. In this way, we assess the additional energy consumption of using different SCA countermeasures and using lightweight block ciphers on 32 bit embedded devices. In addition to this, we provide a methodology for developing an ISA based energy model for cryptographic software with an accuracy of $\pm 5\%$. In addition to providing our methodology for developing this model, we also show that using variations of instructions that reduce the size of code can reduce the energy consumption by as much as $30\% - 40\%$ and that memory instructions reduce the predictability of our energy model.

1 Introduction

Cryptography is a vital application for authenticating and securing data communications and secure systems. It is widely used in the Internet, smart cards, authentication systems and many other security related applications and is becoming more relevant as more things become interconnected digitally.

To secure such systems, symmetric key cryptography is widely deployed on embedded devices. AES is the most popular symmetric key cipher [2]. Lightweight ciphers are a group of symmetric ciphers designed to provide cryptography for constrained environments (such as size, speed or power), often at the cost of security. Given the well known threat that side channel attacks pose on embedded systems running cryptographic applications, such implementations often need to be hardened against side channel attacks using countermeasures, such as masking or hiding.

There are many different kinds of processors which are relevant for practical embedded systems. In this work, we focus on a the ARM Cortex-M4 processor which has a 32-bit architecture and runs Thumb and Thumb-2 assembly [13]. As demonstrated by the emergence of lightweight ciphers in recent years, factors such as power consumption and energy efficiency have become an important design consideration for cryptographic primitives, as cryptography can be a computationally demanding application and is used on many mobile devices where energy efficiency is extremely important for extending battery life [14]. This has become particularly important for the Internet of Things (IoT), where many interconnected mobile devices will interact each other, making security and energy consumption both important design considerations. In this context, resistance of IoT devices against DPA-style side channel attacks may also be an important design consideration to ensure the security of devices.

Measuring energy consumption, however, traditionally requires custom instrumentation of the hardware; a barrier that prevents energy consumption from being treated as a first class software design goal. As cryptographic implementations for software are often developed directly in assembly language and are relatively simple, power models that work on the ISA level would seem to be ideal for profiling, understanding and improving their energy consumption. One type of energy model that works on this level is a statistical model proposed by Tiwari et al. [18] that uses measured information about the energy cost of different instruction sequences to derive a model for estimating and giving transparency to the energy consumption of any given piece of assembly code.

1.1 Previous Work

Previous work in this area has been done in [4], [5] and [8]. [4] examines the design of an optimisation methodology that uses its own split masking method as an alternative reduced energy consumption masking countermeasure for an embedded device. [5] proposes a masking method based on an increased number of lookup tables to provide a more energy efficient masking method against higher order attacks. This work focuses on the design of its own type of masking aimed at reducing the energy consumption overhead at the expense of other factors (such as memory in [5]).

[8] looks at the energy cost, and other features, of lightweight cryptography in hardware. The work provides a good insight into the energy consumption of lightweight ciphers however focuses only on hardware implementations and does not asses the additional energy of using side channel attack countermeasures.

1.2 Our Contributions

This paper provides a methodology of the development of a static energy model for an assembly instruction set typically used in symmetric key cryptography on a processor that is commonly used in practice (ARM Cortex-M4). The accuracy of this model is shown and then used to profile AES with a number of different

SCA countermeasures and three lightweight ciphers (PRESENT, KLEIN and ZORRO) with and without a first order Boolean masking SCA countermeasure.

The static energy model developed uses the statistical method outlined by Tiwari et al. [18] to develop an energy model for the instruction set commonly used in symmetric key cryptography. In doing so we demonstrate that this type of energy model provides results with a $\pm 5\%$ error margin, and so could be used to accurately profile and improve the energy consumption of cryptographic software on a typically deployed processor. We find that memory instructions result in greater unpredictability and thus more error in estimating the energy consumption of any given piece of assembly code, and that variations of instructions that reduce the number of instructions required (such as shifting an operand of an ALU instruction) significantly reduce the energy consumption of instructions by around 30/40%.

We show that for our implementation of AES adding Boolean first order masking to AES increases the energy consumption by 56% and adding Boolean first order masking and hiding roughly doubles the energy consumption. Adding only affine or Boolean second order masking has a much higher energy consumption overhead of 6 and 6.7 times respectively.

We show that for the an implementation of PRESENT, KLEIN and ZORRO on the 32 bit ARM Cortex-M4 processor, these lightweight ciphers consume a higher amount of energy than AES. This is largely due to the implementation of the ciphers on a 32 bit architecture when they are designed with another implementation platform in mind (in particular hardware for PRESENT) or the "lightness" of the cipher being designed for size or higher order masking efficiency (as in the case of KLEIN and ZORRO respectively). In this way we demonstrate that designing lightweight ciphers for speed will produce ciphers that are more energy efficient. However, optimising for size through reducing the size of look up tables (as in the case of PRESENT and KLEIN) provides a significantly lower energy overhead for implementing Boolean first order masking (half as much as for AES).

The following section of this paper gives a background introduction to implementing masking, hiding and lightweight blockciphers. We then describe our energy modelling methodology before using the model on our implementations of AES. We then go on to analyse the energy consumption of the lightweight ciphers with respect to AES with and without Boolean first order masking.

2 Background

2.1 Masking

Since 1999, when Kocher et al. [9] showed that the secret data (such as cryptographic keys) of cryptographic algorithms could be recovered by monitoring the timing, power consumption or electromagnetic field of the device performing the cryptography (known as side channel attacks), software implementations of cryptographic algorithms such as AES have also become more involved, as the

code running on the processor can no longer be viewed as a black box but must be designed to resist these sorts of attacks. A very powerful type of SCA is known as Differential Power Analysis (DPA) which uses the correlation between points on many different execution traces to recover the cryptographic key. Examples of this type of attack have been practically proven possible against symmetric key ciphers [9, 11].

In order to design code that is resistant to SCAs, complete control over the instructions executed by the processor is essential to ensure that no secret information is inadvertently made recoverable by DPA style attacks. This means that secure cryptographic implementations for embedded systems are often written directly in assembly code which include countermeasures to ensure side channel resilience. One of these countermeasures is known as masking which combines the algorithm's secret information with random data, known as a mask, in such a way that it makes it very difficult for an adversary to derive the state or key of the algorithm given the masked state or key, which could be obtained using a DPA style attack [11, 12]. Masking schemes are widely used in many cryptographic implementations in practice, as demonstrated by papers on masking published by industry, such as [15].

There are a number of different masking methods that provide different levels of security. Three of these methods are Boolean first order, affine and Boolean second order masking. The first and third of these both use addition to add the random mask to the secret information, where addition is modulo two, carried out with the x-or operation. The second method, affine, uses multiplication and addition of the form $ax + b$ over $GF(2^8)$, where a and b are random and x is the state being masked. Boolean first order and affine provide security against first order attacks (where a less powerful adversary is considered) with affine being around 30-40 times more resistant than first order Boolean masking [3]. Boolean second order is considered a more secure implementation as it can give security against second order attacks (where a more powerful adversary is considered) [17].

2.2 Hiding

Another type of countermeasure is known as hiding. Hiding is a technique that aims to make the power consumption of a device independent of the intermediate values or operations performed such that the power consumption of each clock cycle is either random or equal. Hiding countermeasures split into two groups: one that aims to hide the sensitive data by altering the time dimension (these include randomly inserting dummy operations or shuffling the operations of the algorithm) and one that aims to hide it in the amplitude dimension (such as by increasing the level of noise or reducing the the signal). [11].

One way of implementing hiding in the time dimension in software implementations is by using a random shuffling method. This method seeks to randomly change the sequence in which the operations of an algorithm are performed. In AES, for example, the order in which the S-Box look up is carried out on each of the 16 bytes of the state does not affect the state of the algorithm at the end of

SubBytes. This kind of shuffling can be done either by selecting a random start index for loops or by permuting the state bytes of the algorithm before entering the loop [19].

2.3 Lightweight Blockciphers

Lightweight blockciphers are blockciphers designed to work in constrained environments, often at the cost of security. They can generally be optimised for different things depending on the nature of the constraint they aim to overcome, such as size, speed or power. Three of these lightweight blockciphers are PRESENT, KLEIN and ZORRO.

PRESENT is a popular lightweight cipher that has a key size of either 80 bits or 128 bits and a block size of 64 bits. PRESENT was designed to be implemented efficiently in hardware where it can be engineered to have a very low gate count (1570 GE, which is competitive with leading compact stream ciphers) and high speed for each block (32 clock cycles). The cipher consists of an SP network, comprising a four bit sbox and a permutation layer. There are 32 rounds and a key schedule is used to derive 32 round keys each of which is 64 bits long [1].

KLEIN is another lightweight cipher that was developed primarily for resource constrained RFID tags. KLEIN has a 64 bit block length and can be used with either 64, 80 or 96 bit key lengths. The structure of KLEIN has a very similar structure to AES with the AddKey, SubBytes, ShiftRows and MixColumns primitives being the same but working on nibbles rather than bytes [7]. Consequently, the SBox is 16 nibbles in size (rather than the 256 bytes of the AES SBox), significantly reducing the required memory. In order to make up for the lower security achieved through the smaller SBox and other design features, the cipher has 16 rounds as opposed to the 10 for AES.

ZORRO was developed primarily as a cipher that could be masked easily for higher order masking by reducing the total number of non-linear operations in the cipher. The structure is very similar to AES with the main difference being the values of the S-Box and number of bytes substituted with the SBox values, both designed to reduce the level of non-linearity thus making the masking of the cipher easier. Other differences between ZORRO and AES is the absence of the KeySchedule, with the same key being used for each round, the round counter is also x-ored into the state after ShiftRows. To account for the lower security of the changed SBox and absence of the KeySchedule, 24 rounds are implemented instead of the 10 for AES [6].

2.4 Energy Modelling

The method of analysing the energy consumption of the ARM Cortex-M4 processor is one proposed by Tiwari et al. [18]. This method models the energy consumption of a processor on the ISA level, and can be used to estimate the energy consumption of a given sequence of assembly instructions. It does so by

taking statistical energy measurements of the processor for each instruction, for every possible pair of instructions, and for external effects.

More specifically, the energy consumption (E_p) of a sequence of assembly instructions is calculated as the base cost of each instruction (B_i) multiplied by the number of times the instruction occurs (N_i), combined with an inter-instruction switching overhead from instruction i to instruction j, ($O_{i,j}$) multiplied by the number of occurrences of this switch ($N_{i,j}$) as well as the cost of external effects, such as cache misses and resource constraints, (k). This is expressed in Eq. 1 which is taken from [18].

$$E_p = \sum_i (B_i \cdot N_i) + \sum_{i,j} (O_{i,j} \cdot N_{i,j}) + \sum_k E_k \tag{1}$$

3 Characterisation of Base and Inter-Instruction Costs

3.1 Setup and Profiling Approach

For the model developed for the ARM Cortex-M4 processor (clocked at 16MhZ), the base and switching costs were measured using an open source power measurement board [10]. The measurement board consists of an STM-Discovery board with an ARM Cortex-M4 processor (of the same specification as the board that was tested) with an additional measurement board that is mounted on top of the Discovery board. This additional measurement board receives the input of the power source of the board that is being tested to derive the energy consumption over a period of time given by a trigger signal.

To measure the base and inter-instruction energy costs, a set containing each instruction that was used in the masked implementations of AES and each possible pair of instructions was identified. Each set of instructions and pair of instructions was then triggered with random 32 bit data, applying constraints to that data if necessary, and the energy measured. The registers being read from and written to were also changed at several intervals to ensure any impact of reading from and writing to different registers was balanced out in the average. The average of 20,000 measurements for each instruction/pair was calculated to account for any noise or measurement errors.

3.2 Choice of Instructions

The number of instructions used in the implementations of the ciphers is very small, consisting of `eor`, `ldr`, `str`, `lsl`, `ror`, `and`, `sub`, `mul`, `b`, `cmp` and `add` instructions. Within the ISA of the Cortex-M4 processor, however, many variations of these instructions are offered to provide increased speed and ease of programming - such as loading and storing a single byte (`ldrb` and `strb`) or including an extra addition or shift. These variations of instructions are used extensively in the implementations, so these were also included in the model. In total 28 instructions were characterised, 17 of which are variations of parent instructions.

3.3 Resulting Model

Energy consumption of ALU instructions was seen to be roughly the same, costing on average $1.65nJ$. The instructions that used the least energy were the `mov` and `cmp` instructions which used $1.19nj$ and $1.20nJ$ respectively. Memory instructions such as `ldr`, `str` use significantly more energy at $2.78nJ$ and $2.66nJ$. The energy cost of storing a single byte (`strb`) is significantly less than a normal store at $1.53nJ$ (-45%). However, loading a single byte (`ldrb`) is significantly higher at $3.90nJ$ (+47%). A non-conditional branch instruction showed an energy cost of $3.45nJ$. Adding a condition that is false to a branch instruction uses $0.9nJ$ less energy than a non-conditional branch. When the condition is true, the energy consumption is roughly equal to a non-conditional branch instruction.

Variations of instructions (by having an extra addition or shift) give a slightly higher energy cost in most cases. The effect of having extra addition or shifting with a `mov` instruction showed an increase of around $0.37nJ$. Extra addition on a `ldr`, `str`, `ldrb` or `strb` has an extra energy cost of around $0.30nJ$ more for each instruction. Having shifting with ALU instructions has no observable effect on energy consumption; it gives an energy cost roughly equal to the main ALU instruction. For memory instructions this is slightly different, with an extra shift having no effect on the energy consumption of a `ldr`, but having an extra cost of around $1nJ$ for a `str`. has no extra energy cost compared to the standard ALU instruction. Performing an `eor` instruction and `lsl` instruction, for example, could be achieved using two instructions, an `eor` ($1.66nJ$) and an `lsr` ($1.62nJ$) leads to a total energy cost of $2.81nJ$, while doing this as a single instruction that performs the `lsr` operation on the second operand to be x-ored has the same energy cost as the `eor` instruction ($1.66nJ$), thus showing an energy saving of 41%. Having a variation of a memory instruction that includes an extra addition on the second operand also makes a significant saving over using two instructions (a `ldr` and an `add`) of 29.5% (reducing from $4.34nJ$ to $3.06nJ$). This confirms that using such variations of instructions instead of separate instructions produces a significant improvement in energy efficiency, and is thus in line with the findings in [16]. The additional energy cost of the inter-instruction effects varies slightly for the individual instructions with an overall average overhead of $0.26nJ$ for each instruction switch. ALU operations typically provide a switching cost of around $0.16nJ$. When combining the instructions with an extra addition or shift, the results are typically far higher at around $0.22nJ$. The results become more varied when loading, storing and branching instructions are considered. `ldr` and `str` instructions have a far higher switching cost than other instructions of around $0.45nJ$ and show a fairly consistent switching cost. `ldrb` and `strb` have a switching energy cost similar to that of the `ldr` and `str` instructions, however show slightly more variability with their results. Including extra additions or shifts in load and store instructions increases the switching cost to around $0.50nJ$. Branching instructions, and in particular conditional branching where the condition is true, provide more anomalous results, giving negative values in many instances which would not be expected [18].

3.4 Validating and Calibrating the Model

To validate the model, a large variety of test data was used. Validation was performed by profiling tests using the energy consumption model and comparing the results with the energy measured when running the tests on the target hardware. The random test data contained sequences of instructions with and without any loads and stores, high and low frequencies of switching, longer and shorter sequences of instructions and a varying number of branches. The results showed that the model typically overestimated the total energy cost in all scenarios but that the error increased when loading and storing were present and in particular when the `ldrb` instruction was present (rising as high as 18% where 7500 instructions were used with 16% of the instructions being `ldrb` compared to an error of 6% where no `ldrb` were used with the same number of instructions).

The effect of switching was also observed to be greater than the model indicated. Another interesting observation is that when the number of instructions in the sequence increases, the error gets larger. These results are likely to be due to external effects ($\sum_k E_k$) which take into account the effect of communication with cache memory (with cache hits or misses causing the energy results of memory instructions to show more unpredictability, hence the greater error when memory instructions are included in the test code) and other effects which could occur in the pipeline of the processor.

To increase the accuracy of the model, the base and switching energy costs were calibrated to take the observations of the causes of error into account. These calibrations include reducing the cost of all instructions by 1% and `ldrb` by 15% and raising the energy cost of switches by 15%.

4 Application to AES with SCA Countermeasures

Four software implementations of masked/unmasked AES were developed in ARM assembly: one without masking and the other three with Boolean first order, affine and Boolean second order masking respectively. The main difference between the masked and unmasked implementations of AES is that the pre-computation of the masked S-box stage is required in the masked implementations to pre-compute the S-box so that the S-box lookup table will be correct for the masked state that needs to be looked up. In addition to this additional S-box pre-computation stage, the AddRoundKey, SubBytes and MixColumns stages are also implemented slightly differently in the masked implementations with only the ShiftRows stage remaining the same.

The numbers of instructions (Inst.s), numbers of different instruction pairs (Swit.s), modelled energy consumption and measured (on the hardware) energy consumption of each of the implementations of AES is shown in Table 1.

Table 1. Modelled and measured energy consumption of AES with different masking countermeasures.

Implementation	Inst.s	Swit.s	Model (μJ)	Measured (μJ)	Error (%)
No Mask	3808	3405	9.72	9.80	-0.78
Boolean First Order	6204	5229	15.17	15.26	-0.59
Affine	25055	20552	58.14	57.19	1.67
Boolean Second Order	25681	19032	65.03	62.37	4.25

Examining the results for the modelled energy consumption in Table 1 shows that adding Boolean first order masking to AES increases its energy consumption by around 55%, from $9.72\mu J$ to $15.17\mu J$, rising with the increased number of instructions (63% increase). Implementing affine masking has a substantially larger increase in energy consumption (6 times greater than unmasked AES). Our implementation of Boolean second order masking has the highest energy consumption of $65.03\mu J$, 6.7 times greater than the un-masked implementation and 12% higher than the affine masked implementation - although with only 2% more instructions due to the higher proportion of memory instructions (49.3% compared to 21.1% for the affine implementation) which require more energy.

The results also show that the energy model provides an accurate estimate of the actual energy cost measured on the hardware. The accuracy of the model is shown by all results having less than 5% error with three of the results showing less than 2% error. Boolean second order implementation shows by far the highest error of 4.25%, due to the increased proportion of memory instructions (which produce less predictable results).

To understand the additional energy consumption of side channel attack countermeasures, hiding was implemented alongside Boolean first order masking in AES. The hiding countermeasure implemented was based on randomly shuffling the initial order of the state bytes. Adding the hiding countermeasure requires 1380 more instructions (22.4% more than the original first order Boolean masking implementation). The hiding implementation was profiled using the energy model. The results are compared with the first order Boolean masking implementation without hiding in Table 2, which shows that the implementation with hiding has a 31.11% higher energy cost. This is higher than the 22.4% increase in the number of instructions due to the proportion of additional memory instructions required for the hiding countermeasure.

5 Energy Consumption of Lightweight Blockciphers

Here we examine the difference between the energy consumption of AES and three lightweight blockciphers, PRESENT-80, KLEIN-80 and ZORRO. We im-

Table 2. Comparison between Boolean first order implementation of AES with and without hiding.

Implementation	Calibrated Model (μJ)
Boolean First Order	15.17
Boolean First Order with Hiding	19.89

plement a version of these lightweight ciphers in ARM assembly with and without Boolean first order masking and use the energy model developed in this paper to analyse their energy consumption with respect to AES. We did not examine implementations with affine masking, higher order Boolean masking or hiding for the lightweight implementations as Boolean first order masking is used most commonly in practice and we aim to understand the energy overhead of implementing this countermeasure on lightweight ciphers.

For KLEIN and ZORRO the implementations were based on the original implementation of AES with the differences described in Section 2.3. PRESENT was designed differently due to its great distinction from AES. The implementation of PRESENT derives the 32 round keys at the beginning of the cipher and computes the bit permutation arithmetically, rather than using a look up table. First order Boolean masking was implemented in the same way as AES for all ciphers. Table 3 shows a comparison of the energy consumption of these three ciphers and AES with and without masking.

Table 3. Energy consumption of lightweight ciphers with and without Boolean first order masking (Boolean).

Cipher	Masking	Inst.s	Swit.s	Energy (μJ)
AES	None	3808	3405	9.80
AES	Boolean	6204	5229	15.26
PRESENT	None	33010	30398	68.68
PRESENT	Boolean	34333	31483	71.91
KLEIN	None	5832	5317	14.82
KLEIN	Boolean	6822	6822	17.04
ZORRO	None	4006	3692	9.81
ZORRO	Boolean	6591	5800	16.00

Table 3 shows that the lightweight ciphers consume more energy than the implementation of AES. This is primarily due to the design of the cipher being for hardware (as in the case of PRESENT), or the cipher being designed to be optimised for size (as in the case of KLEIN) or higher order masking efficiency (ZORRO). PRESENT is shown to use 7 times more energy than AES for this implementation, lower than the 8.5 times the number of instructions due to the amount of arithmetic instructions used in the permutation layer. KLEIN can be

seen to use around 50% more energy than AES and ZORRO around the same as AES. Boolean first order masking schemes similar to the one implemented for AES were implemented on the different lightweight ciphers.

Table 3 also shows the additional energy cost required to provide a Boolean first order masking countermeasure for the lightweight ciphers in comparison to AES. The lightweight ciphers PRESENT and KLEIN require significantly less additional energy to provide this security, ($3.32\mu J$ and $2.22\mu J$ more respectively). This is primarily due to the much smaller SBox (16 bytes rather than 256 bytes) which is required to be masked at the beginning of encryption/decryption. This however is not true for ZORRO which requires the most energy ($6.19\mu J$, $0.73\mu J$ more than AES). The reason for this is the overhead in energy consumption for masking mix columns. As there are more rounds where mix columns is implemented in ZORRO, this gives rise to a higher energy cost of masking when compared with AES, even in the absence of the key schedule.

These results demonstrate that lightweight ciphers that have been optimised for speed will provide implementations that require less energy, however for our implementations in software on a 32 bit architecture, more instructions are required to encrypt a block and therefore more energy is required. Increased speed can often be achieved through the use of lookup tables, this is true of PRESENT for example where the permutation value could be taken from a lookup table rather than being computed arithmetically. This however will significantly increase the size requirement of the program thus providing a greater overhead in terms of energy consumption for masking, as the lookup tables will need to be masked. This is demonstrated by the lower additional cost of implementing Boolean first order masking with the lightweight ciphers PRESENT and KLEIN (which require less memory) than AES.

6 Conclusions

We have presented results that indicate that accurate energy consumption estimations for cryptographic software can be obtained from statistical energy models. We have shown how to build a statistical model and that calibrating the model with a variety of test data can be used to increase the accuracy of the model.

We note that for the ARM Cortex-M4 processor, the ALU instruction costs are roughly equal and that using extensions to these instructions (such as add or shift) produces a small increase in energy cost, but that this is significantly less ($\sim 30\% - 40\%$) than using two instructions to achieve the same result. In this way, optimising the code for speed will produce code optimised for energy consumption. It is also noted that avoiding switching instructions where possible and reducing the number of loads and stores (in particular loading a single byte) is also an effective means of reducing the overall energy consumption.

We give insight into how implementing masking and hiding SCA countermeasures affect the energy consumption of an implementation of AES. Measuring the energy consumption of our implementations shows that there is a 56% in-

crease in energy consumption when adding Boolean first order masking to AES compared to an increase of 6 and 6.7 times for adding affine and Boolean second order masking respectively. We then analyse the overhead of adding hiding and Boolean first order masking to AES and show that there is roughly a doubling of the energy consumption compared to AES with no SCA countermeasures.

In addition to this, we model the energy consumption of three lightweight blockciphers (PRESENT, KLEIN and ZORRO) with and without Boolean first order masking and show that, for the software implementations on the 32 bit ARM Cortex-M4 processor, the energy consumption is significantly higher for PRESENT (7 times higher) and slightly higher for KLEIN (50% higher) than that of AES however the additional energy required to include Boolean first order masking is around half as much as for AES. ZORRO however has a very similar energy consumption to AES when no masking is present, however uses slightly more energy when masked due to the overhead of MixColumns, which is executed 14 times more in ZORRO. This information demonstrates that lightweight ciphers optimised for speed will lead to ciphers being more energy efficient, however ciphers optimised for size (with the absence of, or smaller, look up tables) will require less additional energy to mask.

References

1. Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. PRESENT: an ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings* (2007), P. Paillier and I. Verbauwhede, Eds., vol. 4727 of *Lecture Notes in Computer Science*, Springer, pp. 450–466.
2. Daemen, J., and Rijmen, V. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
3. Fumaroli, G., Martinelli, A., Prouff, E., and Rivain, M. Affine Masking against Higher-Order Side Channel Analysis. In *Selected Areas in Cryptography* (2010), A. Biryukov, G. Gong, and D. R. Stinson, Eds., vol. 6544 of *Lecture Notes in Computer Science*, Springer, pp. 262–280.
4. Gebotys, C. H. A split-mask countermeasure for low-energy secure embedded systems. *ACM Trans. Embedded Comput. Syst. 5*, 3 (2006), 577–612.
5. Gebotys, C. H. A table masking countermeasure for low-energy secure embedded systems. *IEEE Trans. VLSI Syst. 14*, 7 (2006), 740–753.
6. Gérard, B., Grosso, V., Naya-Plasencia, M., and Standaert, F. Block ciphers that are easier to mask: How far can we go? In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings* (2013), G. Bertoni and J. Coron, Eds., vol. 8086 of *Lecture Notes in Computer Science*, Springer, pp. 383–399.
7. Gong, Z., Nikova, S., and Law, Y. W. KLEIN: A new family of lightweight block ciphers. In *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers* (2011), A. Juels and C. Paar, Eds., vol. 7055 of *Lecture Notes in Computer Science*, Springer, pp. 1–18.

8. Kerckhof, S., Durvaux, F., Hocquet, C., Bol, D., and Standaert, F. Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings* (2012), E. Prouff and P. Schaumont, Eds., vol. 7428 of *Lecture Notes in Computer Science*, Springer, pp. 390–407.
9. Kocher, P. C., Jaffe, J., and Jun, B. Differential Power Analysis. In *CRYPTO* (1999), M. J. Wiener, Ed., vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 388–397.
10. MAGEEC. Energy Measurement Infrastructure. Available at: http://mageec.org/2014/04/23/energy-measurement-infrastructure/, 2013.
11. Mangard, S., Oswald, E., and Popp, T. *Power analysis attacks - revealing the secrets of smart cards.* Springer, 2007.
12. Messerges, T. S. Securing the AES Finalists Against Power Analysis Attacks. In *FSE* (2000), B. Schneier, Ed., vol. 1978 of *Lecture Notes in Computer Science*, Springer, pp. 150–164.
13. Microelectronics, S. Stm32f405xx stm32f407xx datasheet. Tech. rep., ST Microelectronics, June 2013.
14. Núñez-Yáñez, J. L., and Lore, G. Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip. *Microprocessors and Microsystems - Embedded Hardware Design 37*, 3 (2013), 319–332.
15. Prouff, E., Rivain, M., and Roche, T. On the Practical Security of a Leakage Resilient Masking Scheme. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings* (2014), J. Benaloh, Ed., vol. 8366 of *Lecture Notes in Computer Science*, Springer, pp. 169–182.
16. Roy, K., and Johnson, M. C. *Low Power Design in Deep Submicron Electronics.* Kluwer Academic Publishers, 1997, ch. Software Design for Low Power, pp. 433–460.
17. Schramm, K., and Paar, C. Higher Order Masking of the AES. In *CT-RSA* (2006), D. Pointcheval, Ed., vol. 3860 of *Lecture Notes in Computer Science*, Springer, pp. 208–225.
18. Tiwari, V., Malik, S., and Wolfe, A. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. VLSI Syst. 2*, 4 (1994), 437–445.
19. Tunstall, M., Whitnall, C., and Oswald, E. Masking Tables - An Underestimated Security Risk. *IACR Cryptology ePrint Archive 2013* (2013), 735.