Received on 28th February 2014
Revised on 10th September 2014
Accepted on 18th September 2014

Design space extension for secure implementation of block ciphers

Giovanni Agosta, Alessandro Barenghi, Massimo Maggi, Gerardo Pelosi

Department of Electronics, Information and Bioengineering – DEIB, Politecnico di Milano, Piazza Leonardo da Vinci, 32, I-20133 Milano, Italy
E-mail: gerardo.pelosi@polimi.it

Abstract: Security has been identified as a critical dimension in the design of embedded systems for almost a decade. A well-recognised critical threat against the security of embedded systems is represented by 'side-channel attacks (SCAs)', which mandate the application of specially tailored countermeasures. These countermeasures are significantly demanding in terms of computation effort, and have traditionally been applied by hand. The recent introduction of a methodology to gauge the security margins provided by software cipher implementations, allows the integration of the automated application of countermeasures into platform-based system-level design methodologies. The authors introduce in the design space of block cipher implementations a new metric concerning the resistance against SCAs, provide a systematic method for the selection of the most appropriate cipher given the security and performance trade-offs, and point out the performance requirements for the random number generator. Moreover, they discuss the implications of the design space extension on system runtime adaptivity. The experimental evaluation demonstrates that a single cipher does not cover optimally a range of convenient operating points and that ciphers like a Serpent, which are considered slow in non-protected implementations, can outperform primitives like the Advanced Encryption Standard when implementations with equal security guarantees against SCAs are considered.

1 Introduction

An increasing need to consider security as a critical dimension in the design of embedded systems has been arising for almost a decade [1, 2], besides more traditional design dimensions such as performances, cost, area and energy. The need for ubiquitous security has increased because of the widespread diffusion of embedded systems in sensitive domains such as health-care, automotive and industrial control. While several works tackled the problem of providing security oriented solutions for hardware design flows [3–5], a significant number of embedded systems are built on top of general purpose platforms, and thus rely on software-based encryption primitives. In addition, software solutions provide a greater design flexibility, a feature which has been acknowledged by the standardised cryptographic protocols allowing a choice in the algorithms to be employed. Software-based security layers are also employed as a fall-back solution in case the hardware based ones are compromised, an increasing trend given the technological progress of IC debugging and testing tools [6]. A significant part of the security margin of a secure device is provided by its resistance against the attacks involving an adversary having temporary physical access to it. This enables the adversary to gather information regarding the secret parameters involved in a cryptographic algorithm through measuring related environmental parameters such as the time needed for the computation, the power consumption or the electromagnetic emissions of the device. All these environmental parameters provide unintentional, and much unwanted, 'side-channels' over which a significant amount of information regarding the secret parameters of a cryptographic primitive is leaked. Indeed, exploiting a statistically significant number of side channel measurements, the attacker is able to infer the value of the secret parameters being used in the computation, regardless of the cipher robustness against theoretical cryptanalysis techniques. These attacks have been proven practically viable even with limited equipment, against production grade implementations of ciphers, thus representing a realistic threat [7]. To this end, a significant amount of work has been done to devise countermeasures against 'side-channel attacks' (SCA), which try to prevent the attacker from collecting information from the side-channel [8–12]. Countermeasures against SCAs are known to impose a very significant overhead on the block cipher execution, thus driving an effort to devise a methodology to apply them automatically only to the portions of the cipher which need to be protected [8, 9, 13]. The selective application of countermeasures has been recently enhanced in terms of efficiency and made tunable, thanks to the introduction of precise measurements of the security margins provided by them [14]. It is thus now possible to integrate SCA countermeasure application in the current platform-based system-level design methodologies [15], effectively integrating the security dimension in the

design space. We note that considering the security as a dimension of the design space is a favourable choice in the case of embedded systems design, as the cost of a possible post-deployment patching mandated by a security breach is significantly higher than the one on general purpose systems.

1.1 Contribution

We provide the following main contributions: (i) we extend the system design space of block cipher implementations with the definition of a metric (denoted as 'attacker effort') and input parameters related to the resistance to SCAs, including both automated countermeasures parameters and algorithm selection; (ii) we show that the extended design space provides critical insights for the selection of the most appropriate cipher to the system requirements, and we point out the fact that protecting an entire cipher suite is overly expensive and does not yield significant benefits; (iii) we show the impact of the security dimension on the hardware requirements, in particular on the performance of the 'random number generators' (RNGs), which is crucial to implement the countermeasures. Finally, we discuss the implications of the design space extension on system adaptivity.

1.2 Organisation of the paper

In Section 2, we introduce the preliminary notions on power-based SCAs, and quantify the computational effort needed to execute an attack against a secure cipher implementation provided with 'hiding' and 'masking' countermeasures. In Section 3, we define the extended design space, while in Section 4 we provide an accurate experimental campaign to assess the impact of the security design dimensions, together with the classical ones, on the implementation choices for two hardware platforms, the issues of dynamic adaptivity and the impact of the RNGs on the overall throughput. Finally, in Section 5, we draw our conclusions.

2 SCA framework

The typical passive SCA workflow is an instance of either a 'known plaintext attack or a known ciphertext attack', aiming at the retrieval of the secret key being employed in the cipher implementation. The attacker is assumed to know the full details of the cipher implementation and to be able to measure an environmental parameter of his choice, from which he will derive the information regarding the secret key. The main strength of an SCA is the possibility of considering the effect of subsets of the secret-key bits on the computation separately. The direct consequence is that the attacker can perform an exhaustive search over a reduced amount of key of bits at once, significantly decreasing the computational effort. We will now provide a complexity analysis for the computational effort of the attacker, which will be one of the goals to be optimized in the design space.

2.1 Attacker model

In the passive side channel attack scenario, the attacker is able to perform a measurement of the side channel (e.g. power consumption) during the computation of a sensitive operation of the cipher, for an arbitrary number of cipher runs, with different inputs. The attacker gains knowledge on a cumulative working parameter of the system, depending on one or more key bits. To obtain the actual key bits values, the attacker needs to compute a key dependent side-channel hypothesis for all their possible values, and compare its results against the actual measurements by means of a statistical test [7, 16].

2.2 Typical attack framework

A typical attack selects an intermediate value of the cipher depending on a small key portion (usually 8 bits) and a known value (either the plaintext or the ciphertext). The side-channel is continuously measured during the aforementioned operation, for a large set of randomly distributed inputs, as it is usually the case that the exact time instant where the sensitive operations take place is not precisely known. Subsequently, the attacker tries to predict the actual power consumption of the device relying on the knowledge of the inputs and making an hypothesis for all the possible values of the secret key portion taken into consideration. This leads to a set of predictions of the side-channel values (e.g. power consumption) for each value taken by the portion of the secret key.

2.3 Required computational effort

Consequentially, to recover a k-bit secret key, tackling b bits at once, the typical attacker will need to perform a computational effort not lower than $\Omega((k/b)2^b)$. This effort is due to the fact that the side channel predictions have to be computed for all the 2^b values of the b bits of the key involved in the computation. The aforementioned computation has to be repeated for (k/b) times in order to recover the whole cipher key.

The obtained predictions must then be compared with the actual side channel measurement performed in the time instant when the sensitive instruction(s) are computed. To this end, since it is technically challenging to spot with full precision the time instant when an instruction is computed, the usual attack flow collects a very small temporal series of l measurements around it and performs the aforementioned computations for each one of them. In our analyses, we will assume conservatively that the attacker is able to know exactly when the operation is executed in an unprotected implementation, and take into account the
Nyquist's sampling theorem. Thus, the effort for an attacker is not lower than $\Omega(2.5(k/b)2^b l)$, where l is the length, expressed in number of instructions, of the code portion targeted by the attack. In the case of an attack against an unprotected implementation $l = 1$, and the lower bound is $\Omega(2.5(k/b)2^b)$.

2.4 SCA countermeasures

Countermeasures against SCAs are conventionally split in two types: 'hiding' and 'masking'. Hiding increases the uncertainty of the attacker regarding the time location of the sensitive instructions by means of random delays inserted in the computation [7, 10], whereas masking employs random values to split the computation of the sensitive values in multiple shares [7, 11, 17].

In case of hiding countermeasures, the best strategy for an attacker is to perform an attack after computing a moving window average of the measurements. The effects of the averaging compensate for the uncertainty at the cost of

adding extra noise to the measures and requiring measurements to be obtained [7]. We will consider the contribution of a countermeasure based on the temporal uncertainty of the executed instructions as an increase in the value of the parameter l employed in the aforementioned lower bound. Masking countermeasures act through padding the sensitive values with a freshly extracted random value for each one of them. The net effect is that the attacker is not able to build a correct side-channel leakage model as the random masks are changed every time, effectively acting as a 'one-time-pad' on the sensitive values. The effect of the random masks is removed at the end of the computation. The attack strategy against this countermeasure mandates to collect measurements of two operations involving both the masked value and its mask, and find a way to combine their side-channel leakage so that it no longer depends on the random-pad value. The masking countermeasure can be enhanced to prevent this attack employing a higher number m of masks ($m > 1$), for each sensitive operation, a procedure known as 'high-order' masking. It has been proven in [11] that given a proper masking scheme with m masks, the attacker needs at least $m + 1$ measurements to be able to combine them into a mask-independent leakage value. Such attacks are known in open literature as 'high-order' SCAs [7].

Masking raises the computational effort of the attacker as he should be able to pick the correct measures to be recombined into the mask independent value, that is, pick $m + 1$ samples out of the l contained in the measurement. In this work, we will assume that the attacker has no information on the execution order of the masked instructions, which, in turn, implies a $\binom{l}{m+1}$ increase in the computation effort as he has no information on how to select the instructions. We note that an increase in the knowledge of the mutual positions of the instructions performing a masked operation may reduce the effect of this term, and can be taken into account in case the design scenario considers such an attacker model. Masking and hiding have a strong synergy in enhancing the efforts required by the attacker, as hiding provides an effective way to raise the value of l, providing a significant boost to the aforementioned computational effort. Currently, there is no practical evidence of a high-order SCA beyond order 3, which has been led in a partially simplified environment to facilitate the measurements [12, 17].

Summing up, a lower bound for the computational effort needed to attack a cipher implementation, protected with both hiding and masking countermeasures, is given by: $\Omega\left(2.5\frac{k}{b}2^b l\binom{l}{m+1}\right)$.

3 Extending the design space with the security dimension

Modern embedded system design methodologies rely on platform-based design techniques, where the application is mapped to software and hardware platforms. However, many parameters can be tuned to satisfy the needs of the specific application scenario [15]. This tuning can be usefully automated to decrease both time to market and development costs through design space exploration (DSE) techniques [18, 19]. This set of automated optimisation techniques aims at solving a multi-objective optimisation problem, where the goals and constraints are imposed by the application scenario, and the tunable parameters are those exposed by the software and hardware platforms adopted. Since the optimisation problem addresses multiple objectives such as performance and area, there is usually no single solution, but a Pareto-set of non-dominated solutions. An introduction to DSE techniques and a comparison of optimisation algorithms can be found in [20, 21].

In this framework, the strength of the countermeasures against SCA, together with the choice of a specific cryptographic algorithm, can be counted as a parameter exposed by the platform. At the same time, the effective computational effort required for an attacker to breach the security of the system should be taken into account as one of the goals of the multi-objective optimisation problem.

To this end, the evaluation of the security level, and a tunable application of the countermeasures must be performed automatically. Compiler-based tools such as those presented in [8, 13, 14, 22] serve to this purpose. In particular, the security-oriented data-flow analysis (SDFA) technique proposed in [14] to practically gauge the security margin provided by a countermeasure application does not require the use of *ad-hoc* domain specific languages, as in [22], and does not need profiling information obtained from a prototype as in [13]. Consequentially, the work in [14] represents an attractive solution for an application in DSE, where simulation and profiling are often very expensive, in particular in the early stages of design. We thus chose to employ the SDFA as the building block for our DSE.

The SDFA provides an instruction-level assessment of SCA resistance, defining the 'instruction resistance' metric: a conservative approximation of the number of cryptographic key bits that influence, directly or indirectly, the computation of each intermediate value used in an unprotected cipher implementation. The instruction resistance effectively marks the lower bound for the value of the b bits to be guessed at once by an attacker targeting that specific instruction.

3.1 Design parameters

The design space parameters related to the security domain for considering SCA countermeasures are the following: the choice of the cryptographic algorithm; the security margin; and the masking order.

3.1.1 Cryptographic algorithm: The specific ciphersuite used in a system can be a design parameter of the platform, especially in the case where cipher flexibility should be provided to allow the product to be compliant with multiple standards. We note that, in some cases, the choice of the algorithm may be fixed by the application scenario: in those cases the effective design parameter space has the cryptographic algorithm choice as an enforced constraint.

To provide a comprehensive analysis, we allow the algorithm to be chosen in the design space from the following ones: (i) the Advanced Encryption Standard (AES), with all the standardised key lengths (128-, 192-and 256-bit keys) [23]; (ii) Camellia, developed by Mitsubishi and NTT, and recognised as ISO/IEC 18033-3 standard [24]; (iii) the strengthened variants of the Data Encryption Standard (DES), triple DES (with both 112- and 168-bit key lengths) and DES-X (184-bit key) [25]; and (iv) Serpent, one of the finalists of the AES contest, highly regarded for its security margin and known to be slower than AES in SCA-unprotected implementations [26]. All the codebases to perform the SDFA were picked from the

libgcrypt [27], the backend encryption library employed by GNU Privacy Guard, save for AES, which is the reference implementation for small memory footprints, as we target an embedded system design. The choice of the cipher determines the value of the key length k in the computation of the attacker effort.

3.1.2 Security margin: This parameters specifies the maximum 'resistance' [14] value of the instructions to which masking countermeasures should be applied to prevent the attacker from exploiting them. For each instruction of the cipher implementation, the instruction resistance is informally defined as the minimum number of key bits influencing any bit of its output value. To obtain this information automatically, a SDFA [14] is performed by a compiler extension to detect the amount of key material involved in the computation of any intermediate value of the cipher. The analysis is able to identify the portions of the executable program amenable to passive SCAs. In particular, an instruction is deemed to be vulnerable if computing a model of its behaviour for each possible value of the key bits b from which its output value depends is computationally feasible. Computing the aforementioned model is the ground on which passive SCAs are built (see Section 2), as its predictions are matched against the measured behaviour of the considered device. The information to be traced is the data-dependence between any bit computed by a program instruction and every bit of the cipher-key. Such a choice is mandated by the need to consider possible SCA models predicting the behaviour of the computation of a single bit within a word-wide value computed by the underlying platform.

When the cipher instructions with a resistance lower than the security margin are protected with a masking countermeasure, the attacker's choice for the value of the b bits of the secret key to be guessed in the attacker effort formula is lower bounded by the security margin itself. In the experiments, the considered range of b is 1–128.

3.1.3 Masking order: According to the attacker model against which protection is desirable, the designer picks the minimum masking order m to be employed [11, 12, 17]. However, it is possible that, in the case of high resource availability, a higher order protection may be desirable – for example, to prolong the lifetime of the final product, or to widen its target market. The masking order ranges, in our experiments, from 1 to 2, since there is no evidence of a third order attack against real world devices in open literature.

3.2 Optimisation goals

In general, a multi-objective optimisation problem involves the minimisation (maximisation) of a function with several arguments which maps two or more goals (e.g. performance and security level), thus making the definition of optimality not unique. In particular, decisions need to be taken in the presence of trade-offs between two or more conflicting goals. Minimising cost (or/and maximising performance), while maximising the security level of a cryptographic implementation is an instance of a multi-objective optimisation problem involving two (or three) goals. For a non-trivial instance of the problem, there does not exist a single solution (i.e. a set of input argument values for the target function) that simultaneously optimises each goal. A solution is called 'non-dominated' or 'Pareto optimal' if none of the goals can be improved in value without degrading some of the others. Without additional subjective preference information, all Pareto optimal solutions are considered equally good. Therefore a DSE framework identifies the Pareto optimal solutions leaving to the designer the selection of the solution corresponding to the most appropriate trade-off for the considered application scenario [19].

In our analysis, we assume the design space to be used to solve a multi-objective optimisation problem with the following three goals, the first of which is specific to the security dimension.

3.2.1 Attacker effort: A measure of the overall computational effort an attacker needs to exert to break the cipher, under the assumed attacker model. In this work, we consider the attacker to be employing Pearson's sample linear correlation coefficient r as the statistical tool. Consequentially, the effective attacker effort can be obtained taking into account the lower bound on the effort described in Section 2 and updating the one of computing the metric. The total effort is $\Omega\left(10n_{\text{meas}}\frac{k}{b}2^{b}l\binom{l}{m+1}\right)$, where n_{meas} is the number of measurements to be obtained for the Pearson's r to be statistically significant. In this work, we consider the worst case scenario, that is, the one more advantageous to the attacker, thus taking $n_{\text{meas}} = 30$, which is the minimum to satisfy the Gaussian assumption on the variable distribution needed in Pearson's r index. The measurements length l is taken to be exactly 2.5× the number of instructions involved in the attack, that is, the ones required to break the masking scheme, plus the uncertainty provided by the hiding countermeasure. We assume that the hiding countermeasures provide a ten dummy instruction time window, a sensible value according to common practice [7], into which the actual ones are inserted.

3.2.2 Performance: The encrypted data throughput provided by the protected cipher. Performance is the typical metric which is taken into account by most of the design frameworks. In particular, the computations that must be performed in a system for the purpose of security can easily overwhelm the computational capabilities of processors in both low- and high-end embedded systems. This is because, for example, to the augmented throughput demands on components such as the true RNG, which in turn affects the performance of the secure cipher implementation.

3.2.3 Code size: The code size of the protected cipher, like the 'performance' one, it is a typical goal (especially for designs involving platforms with limited instruction cache/memory size), and is also directly impacted by the countermeasures. This goal is especially relevant in code-memory constrained devices such as microcontrollers, or in the case where multiple protected cipher implementations are stored for the sake of run-time adaptivity.

4 Experimental evaluation

This section reports the results of an exploration of the design space, showing the usefulness of the information provided by the security oriented parameters and goals introduced in the previous section. We chose as target platforms two ARM SoCs: a TI-OMAP4460 SoC, which is the core of the Pandaboard-ES, and a Marvel Kirkwoood SoC present on

the Pogoplug NAS. The former, is based on a dual core Cortex-A9MPCore clocked at 1.2 GHz, 1 MiB L2 cache and 1 GiB DDR2 RAM, running Linaro 12.09 Linux distribution (`armv7l` target), while the latter is based on an ARM 926EJS core (`armv5te` target), clocked at 1.2 GHz, with 256 MiB DDR RAM running Arch Linux 3.16.1.

4.1 Characterisation of the design space

Fig. 1 reports the set of feasible solutions for the considered design space for both target platforms. The three optimisation goals (throughput, attacker effort and code size) are depicted for each solution, together with the cipher selection parameter. The attacker effort computation has been made employing the lower bound formula reported in Section 3.2. We chose l to be 2.5 × 'the number of samples picked by the attack' (which is the 'masking order' plus one), to take into account a safety margin over sampling the side channel signal exactly at the bound required by the Shannon's theorem (which would mandate sampling at least twice as fast as the chip clock frequency). Considering a hiding countermeasure providing a time window made of ten dummy instructions, and a first-order masking ($m + 1 = 2$), we obtain $l = \lceil 2.5(10 + 2) \rceil = 30$ taking into account the side channel sampling. Similarly, taking into account a second-order masking strategy, $m + 1 = 3$ results in $l = \lceil 2.5(10 + 3) \rceil = 33$.

For all the ciphers, an increase in the attacker effort is characterised by a decrease of the throughput, because of the higher computational overhead introduced by the application of the share splitting countermeasure to a larger number of instructions. In particular, we note that the significant throughput reduction which characterises the 20–48 bit attacker effort interval is due to the fact that the Ishai-Sahai-Wagner (ISW) masking countermeasure needs to be applied to a significant amount of non-linear operations (i.e. the ones different from `xor`) at each iteration of the block cipher round. This, in turn, results in a significant cost, as the Boolean `and` and `or` operations are the most expensive, given the countermeasure in use. The increase in the code size because of the larger amount of instructions is also evident for both platforms.

Fig. 1 reports the results obtained through either first or second-order masking protections on the same plot. Different operating points with similar attacker effort may yield a different throughput because of the application of a masking countermeasure with a different order. The two separate trends in the attacker effort increase are caused by second-order countermeasures being significantly computationally demanding w.r.t. the first-order ones (the cost grows quadratically with the order). For example, considering the Serpent algorithm, the dark blue data series refers to the application of the first-order masking, whereas the second-order one is referred by the lower light blue one. To understand which ciphers should actually be used for the selected target platforms, we report the sets of Pareto-optimal solutions in Fig. 2. It is possible to see that the DES variants are definitely suboptimal, and provide no favourable solution. On the other hand, the Serpent cipher provides optimal solutions when code size is not an issue, whereas Camellia proves better when code size must be kept low – often a significant issue in embedded systems.

Despite being the world-wide standard, the AES implementations provide optimal solutions only where protection against SCA is a very limited issue (i.e. for computational efforts in the range of 2^{20}–2^{32}, which are considered practically unsafe being below at least 80 bits of protection). It is interesting to note that the Serpent implementation considered turns out to be both faster and more compact in terms of code size with respect to the AES counterpart, despite being slower when SCA countermeasures are not considered.

4.2 Run-time adaptivity

While in traditional platform-based design the parameters of the system are set at design time, the increasing complexity of embedded systems makes it valuable to preserve flexibility after the design and deployment phases. To this end, adaptability to operating conditions can be added to a system by allowing it to change its behaviour among a set of configurations ('operating points') at run-time, through the use of a Run-time Resource Manager [18, 28].

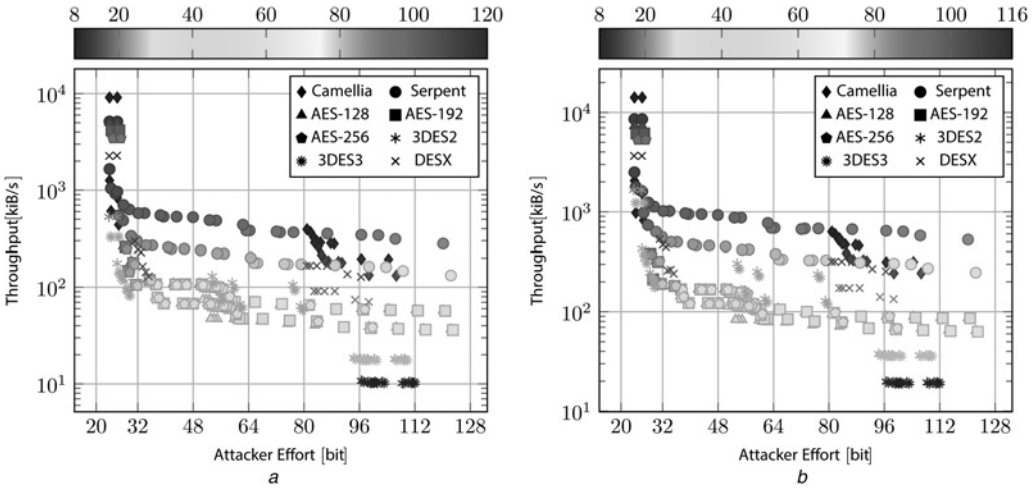

Fig. 1 *Solution space for the design problem on the*

a ARM-926 and

b ARM Cortex-A9 platforms

Attacker effort is expressed in `bit`, as the attacker should be able to perform at least 2^{bit} computations to breach the system

Colours denote the code size of the output code segment in kiB

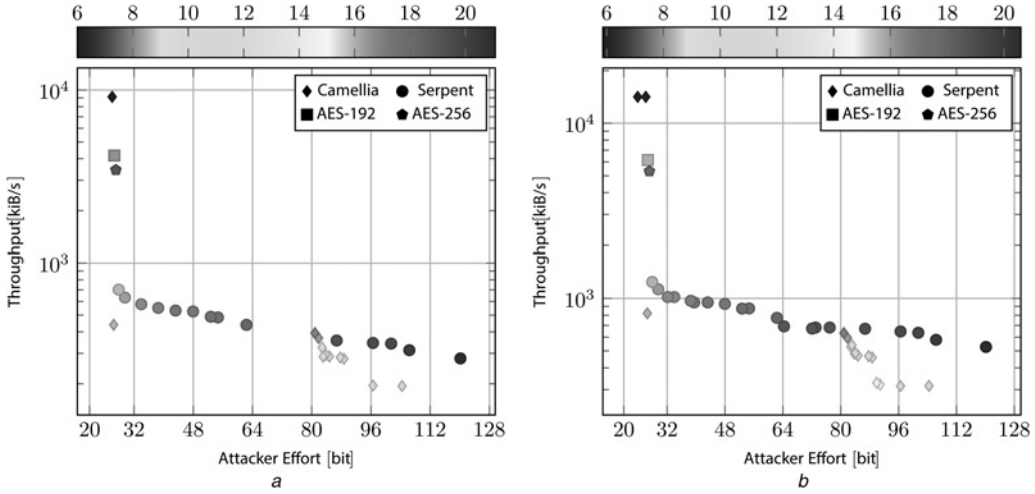

Fig. 2 *Pareto-optimal solutions for the design problem on the*
a ARM-926 and
b ARM Cortex-A9 platforms
Attacker effort *b* is expressed in `bit`, as the attacker should be able to perform at least 2^b computations to breach the system
Colours denote the code size of the output code in kiB

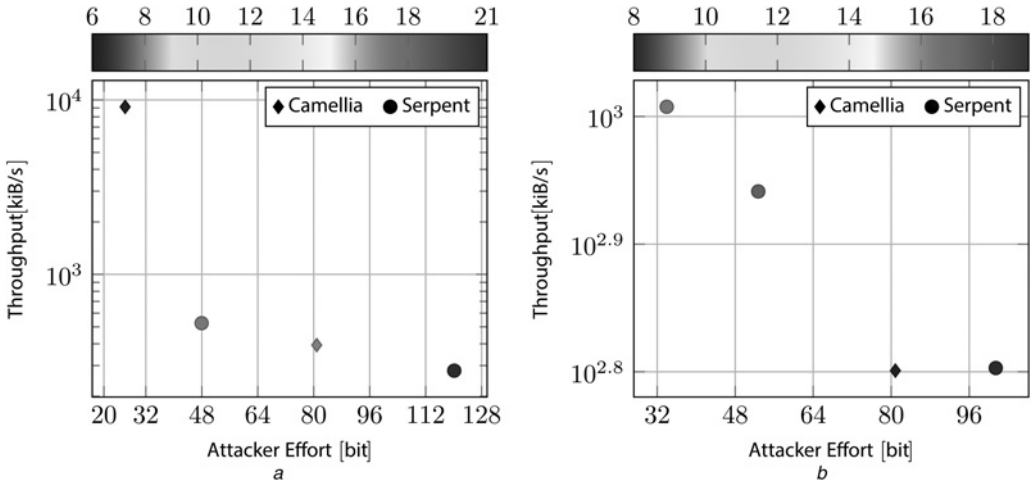

Fig. 3 *Operating points for ARM-926,* `armv5te` *architecture*
a Standard ISA and
b Cortex-A9 `armv7` architecture, Thumb2 ISAE

As it is commonplace for cryptographic protocols to allow the choice of different block ciphers for their operation, most security libraries have already integrate multiple ciphers. An adaptive application which performs communication with symmetric key encryption can leverage this support for negotiating the use of a cipher that fits its current requirements. For example, critical information may be transmitted using a cipher implementation warranting a high attacker effort, whereas in a high-load condition, less sensitive information might be protected with a faster, albeit less secure, one. To take advantage of this opportunity, it is possible to extract from the design space a set of appropriate operating points. Considering the Pareto-optimal solutions in Fig. 2, we show in Fig. 3 the set of operating points that a designer, supported by a DSE tool, would choose. The solutions worthy of being actually deployed on the platform should include: (i) points with highest throughput and a minimum attacker effort of 80 bits; (ii) points with the

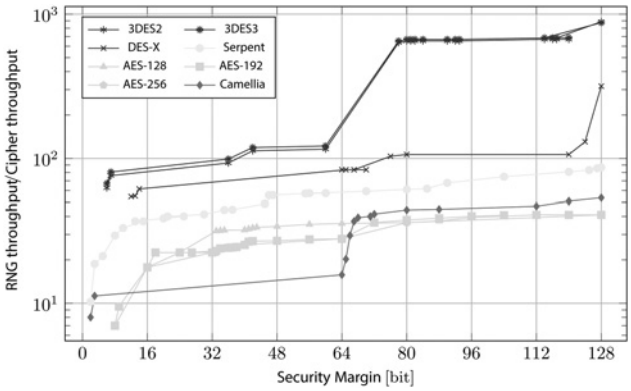

Fig. 4 *Required amount of random masks per ciphertext output byte, considering a first-order masking*

Table 1 Impact of the RNG throughput on the protected cipher implementation performance

Platform	Attacker effort		AES			DES family			Serpent	Camellia
			128	192	256	DES-X	3DES2	3DES3		
Cortex-A9	80-bit	throughput, kiB/s	92.43	95.30	95.55	310.99	36.83	35.61	654.20	617.74
		speedup w/o the RNG bound	1.40×	1.32×	1.30×	1.69×	1.88×	1.88×	4.23×	1.24×
		speedup w/o computation bounds	12.44×	15.40×	15.36×	1.58×	1.76×	1.77×	1.06×	4.22×
	128-bit	throughput, kiB/s	83.80	83.98	83.81	248.04	34.89	34.01	479.92	301.80
		speedup w/o the RNG bound	1.44×	1.43×	1.41×	2.06×	1.89×	1.87×	4.23×	1.39×
		speedup w/o computation bounds	11.96×	11.94×	11.96×	1.56×	1.76×	1.76×	1.06×	2.90×
ARM-926	80-bit	throughput, kiB/s	61.20	64.02	63.30	163.03	18.12	17.59	348.02	384.10
		speedup w/o the RNG bound	1.39×	1.30×	1.30×	1.78×	1.58×	1.58×	3.04×	1.24×
		speedup w/o computation bounds	11.57×	14.12×	14.28×	1.86×	2.20×	2.21×	1.23×	4.18×
	128-bit	throughput, kiB/s	55.66	55.44	54.81	125.01	17.19	16.70	253.41	186.22
		speedup w/o the RNG bound	1.42×	1.41×	1.41×	1.76×	1.59×	1.57×	2.96×	1.39×
		speedup w/o computation bounds	11.09×	11.14×	11.26×	1.90×	2.20×	2.21×	1.24×	2.89×

Greyed out cells point to the factor being the most limiting to the implementation performances between the computational efficiency of the platform and the RNG speed.

highest absolute throughput; (iii) points with the highest attacker effort; (iv) and points corresponding to a total code size not exceeding 64 kiB (about 10× the size of the smallest Pareto-optimal solution). Performing a *k*-means clustering, we took one representative point per cluster fitting requirements (i)–(iii), and we bounded the number of clusters taking into account the fourth constraint, thus obtaining four clusters. Fig. 3 depicts the extracted operating points: the results show that it is possible for the designer to instrument only the Camellia and Serpent out of the eight examined ciphers, and pick the operating point according to the throughput and security margin required by the operating conditions. In particular, we note that the operating points selected for the Serpent cipher provide a moderate and very high protection while retaining an acceptable code size (17 and 21 kiB, respectively). The low protection point provided by the Camellia cipher may be useful to obtain a significant throughput improvement (roughly 10× w.r.t. the other operating points) while retaining a minimal protection against casual attacker. The 80-bit operating point provided by Camellia represent a particularly interesting solution, as the cipher retains a small code size (8 kiB), while providing a very sound security margin and reasonable throughput.

4.3 Performance impact of random number generation

Since masking countermeasures require a significant amount of random data to pad the computations, it is useful to analyse this requirement, as an adequate RNG, either a hardware TRNG or a proper secure PRNG should be included in the platform. We note that the PRNG requirements are a function of the chosen security margin, masking order and cipher, with the masking order acting as a plain multiplicative factor on the throughput requirements. In the following, we will present the results on the requirement imposed on the RNG, considering the Pandaboard platform (Cortex-A9) employed in our previous experiments. Fig. 4 reports the amount of random bytes needed to compute a cipher byte for all the ciphers of our exploration, in logarithmic scale, considering a first-order masking. AES and Serpent are characterised by a linear dependence of the requirement from the security margin because of the regular structure of the cipher round. Camellia has very low RNG requirements up to 64 bits, as the key material is added to the plaintext after a thorough mixing, achieving a steep rise in the security margin with a very low amount of protected instructions. The DES family has considerable RNG throughput requirements as it is characterised by a large number of instructions, because of the large number of bitwise operations involved in its computation. The steep increase for the 3DES2 and 3DES3 is justified by the fact that increasing the security margin above 56 bits implies protecting two full DES executions (the first and the last), considerably raising the amount of masked instructions.

To provide practical grounding, Table 1 compares the actual throughput obtained from protected ciphers on the case study platforms against the one achievable with the speedup obtained through either removing the RNG overhead or removing the bottleneck caused by the CPU computation capabilities. The first value is computed through actually running the implementations with a constant value instead of invoking the RNG, while the second one is derived from the RNG requirements in Fig. 4 and considering the throughput of the RNG running alone on the platform. The results show that the only cipher which performances are limited by the RNG on both platforms throughput is Serpent, while AES and Camellia would benefit from a more performing CPU. The DES family is characterised by being roughly equally limited by both the CPU performances and the RNG throughput: this results in being slightly bound by the RNG speed on the faster platform, while being capped by the CPU on the slower one.

5 Concluding remarks

We extended the system design space with security related metrics and parameters of software implementations against SCAs. We have shown that favourable trade-offs can be selected depending on the desired security margin, throughput and code size providing viable working points, which may also be used for runtime adaptivity purposes.

6 References

1 Ravi, S., Kocher, P.C., Lee, R.B., *et al.*: 'Security as a new dimension in embedded system design'. Proc. Design' Automation Conf. 2004, San Diego, CA, USA, June 7–11 2004, pp. 753–760

2 Ravi, S., Raghunathan, A., Kocher, P.C., *et al*.: 'Security in embedded systems: design challenges', *ACM Trans. Embed. Comput. Syst.*, 2004, **3**, (3), pp. 461–491

3 Guo, X., Fan, J., Schaumont, P., Verbauwhede, I.: 'Programmable and parallel ECC coprocessor architecture: tradeoffs between area, speed and security', in Clavier, C., Gaj, K. (Eds.): 'Cryptographic hardware and embedded systems, CHES 2009' (Springer, 2009), pp. 289–303

4 Narayanan, S.H.K., Kandemir, M.T., Brooks, R.R.: 'Performance aware secure code partitioning'. Proc. Design Automation and Test in Europe 2007, Nice, France, 16–20 April 2007, pp. 1122–1127

5 Tiri, K., Verbauwhede, I.: 'A VLSI design flow for secure side-channel attack resistant ICs'. Proc. Design Automation and Test in Europe 2005, Munich, Germany, 7–11 March 2005, pp. 58–63

6 Boit, C., Helfmeier, C., Kerst, U.: 'Security risks posed by modern IC debug & diagnosis tools'. Proc. 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography 2013, Los Alamitos, CA, USA, 20 August 2013, pp. 3–11

7 Mangard, S., Oswald, E., Popp, T.: 'Power analysis attacks-revealing the secrets of smart cards' (Springer, 2007)

8 Agosta, G., Barenghi, A., Pelosi, G.: 'A code morphing methodology to automate power analysis countermeasures'. Proc. Design Automation Conf. 2012, San Francisco, CA, USA, 3–7 June 2012, pp. 77–82

9 Agosta, G., Barenghi, A., Pelosi, G., Scandale, M.: 'A multiple equivalent execution trace approach to secure cryptographic embedded software'. Proc. Design Automation Conf. 2014, San Francisco, CA, USA, 1–5 June 2014, pp. 1–6

10 Coron, J.-S., Kizhvatov, I.: 'Analysis and improvement of the random delay countermeasure of CHES 2009', in Mangard, S., Standaert, F.-X. (Eds.): 'Cryptographic hardware and embedded systems, CHES 2010' (Springer, 2010), pp. 95–109

11 Ishai, Y., Sahai, A., Wagner, D.: 'Private circuits: securing hardware against probing attacks', in Boneh, D. (Ed.): 'Advances in cryptology – CRYPTO 2003' (Springer, 2003), pp. 463–481

12 Tillich, S., Herbst, C.: 'Attacking state-of-the-art software countermeasures-a case study for AES', in Oswald, E., Rohatgi, P. (Eds.): 'Cryptographic hardware and embedded systems, CHES 2008' (Springer, 2008), pp. 228–243

13 Bayrak, A.G., Regazzoni, F., Brisk, P., *et al*.: 'A 'First step towards automatic application of power analysis countermeasures'. Proc. Design Automation Conf. 2011, San Diego, California, USA, 5–10 June 2011, pp. 230–235

14 Agosta, G., Barenghi, A., Maggi, M., Pelosi, G.: 'Compiler-based Side channel vulnerability analysis and optimized countermeasures application'. Proc. Design Automation Conf. 2013, Austin, TX, USA, 29 May–7 June 2013, pp. 81:1–81:6

15 Keutzer, K., Newton, A., Rabaey, J., *et al*.: 'System-level design: orthogonalization of concerns and platform-based design', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2000, **12**, (19), pp. 1523–1543

16 Barenghi, A., Pelosi, G.: 'On the security of partially masked software implementations'. Proc. 11th Int. Conf. on Security and Cryptography, Vienna, Austria, 28–30 August 2014, pp. 138:1–138:8

17 Schramm, K., Paar, C.: 'Higher order masking of the AES'. Topics in Cryptology – CT-RSA 2006, The Cryptographers' Track at the RSA Conf. 2006, 2006, pp. 208–225

18 Mariani, G., Avasare, P., Vanmeerbeeck, G., *et al*.: 'An industrial design space exploration framework for supporting run-time resource management on multi-core systems'. Proc. Design Automation and Test in Europe 2010, Dresden, Germany, 8–12 March 2010, pp. 196–201

19 Palesi, M., Givargis, T.: 'Multi-objective design space exploration using genetic algorithms'. Proc. Tenth Int. Symp. on Hardware/Software Codesign, CODES 2002, Estes Park, CO, USA, 6–8 May 2002, pp. 67–72

20 Calborean, H., Jahr, R., Ungerer, T., *et al*.: 'A comparison of multi-objective algorithms for the automatic design space exploration of a superscalar system', in Dumitrache, L. (Ed.): 'Advances in intelligent systems and computing' (Springer Berlin Heidelberg, 2013), pp. 489–502

21 Silvano, C., Fornaciari, W., Palermo, G., *et al*.: 'MULTICUBE: multi-objective design space exploration of multi-core architectures'. Proc. 2010 IEEE Computer Society Annual Symp. on VLSI (ISVLSI), Lixouri, Kefalonia, 5–7 July 2010, pp. 488–493

22 Moss, A., Oswald, E., Page, D., *et al*.: 'Compiler assisted masking', in Prouff, E., Schaumont, P. (Eds.): 'Cryptographic hardware and embedded systems, CHES 2012' (Springer, 2012), pp. 58–75

23 Daemen, J., Rijmen, V.: 'The design of Rijndael: AES-the advanced encryption standard' (Springer, 2002)

24 Aoki, K., Ichikawa, T., Kanda, M., *et al*.: 'Specification of Camellia-A 128-Bit Block Cipher'. https://www.info.isl.ntt.co.jp/crypt/eng/camellia/dl/01espec.pdf, accessed September 2014

25 NIST: 'FIPS-46-3: Data Encryption Standard (DES)', http://www.itl.nist.gov/fipspubs/, accessed September 2014

26 Anderson, R.J., Biham, E., Knudsen, L.R.: 'The case for serpent'. Proc. AES Candidate Conf., New York, USA, 13–14 April 2000, pp. 349–354

27 Koch, W.: 'FSF: Libgcrypt', http://www.directory.fsf.org/wiki/Libgcrypt, accessed September 2014

28 Ykman-Couvreur, C., Avasare, P., Mariani, G., *et al*.: 'Linking run-time resource management of embedded multi-core platforms with automated design-time exploration', *IET Comput. Digital Techn.*, 2011, **2**, (5), pp. 123–135