Intellectual Property Protection for Integrated Systems Using Soft Physical Hash Functions

François Durvaux, Benoît Gérard, Stéphanie Kerckhof,
François Koeune, and François-Xavier Standaert

Université catholique de Louvain, UCL Crypto Group,
B-1348 Louvain-la-Neuve, Belgium

Abstract. Intellectual property right violations are an important problem for integrated system designers. We propose a new solution for mitigating such violations, denoted as soft physical hash functions. It combines previously introduced ideas of soft hash functions (in the field of image processing) and side-channel leakage (in the field of cryptographic hardware). For this purpose, we first introduce and formalize the components of an intellectual property detection infrastructure using soft physical hash functions. Next, we discuss its advantages over previous proposals aiming at similar goals. The most important point here is that the proposed technique can be applied to already deployed products. Finally, we validate our approach with a first experimental study.

1 Introduction

While technology shrinking has been the enabling factor for producing increasingly powerful micro- and nano-electronic devices, it has also made the manufacturing process of these devices an increasingly difficult and expensive task. Besides, the high complexity of present hardware/software co-designs leads most system developers to assemble various pieces of hardware and software produced by different companies. This has motivated the development of new businesses, relying on the selling of Intellectual Property (IP) cores. A recent study from Semico Research estimated that the IP core market reached 4 billion US$ in 2009, i.e. a growth of 23,2% compared to the 2005 figures [28]. The advantages of re-using IP cores are enormous in terms of cost and development time reductions. But they also raise important risks regarding IP theft. Namely, the high-valued nature of hardware/software systems naturally creates a strong incentive for piracy, cloning of products and copyright infringement. In addition, the selling of IP cores implementing certain functionalities, that have to be integrated in larger developments, raise compatibility constraints that make the situation even more challenging. This is in part due to the versatility of these IP, that can be distributed under different shapes. For example, IP can be found in high-level format (i.e. described with a high-level language) or in low-level format (i.e. as completely laid out functional blocks restricted to a given technology). As a result, various solutions have been proposed to decrease the risk of unlicensed IP usage.

D.H. Lee and M. Yung (Eds.): WISA 2012, LNCS 7690, pp. 208–225, 2012.

Looking at low-level IP, two main general approaches are usually suggested in the literature. A first line of research can be denoted as "permission-based". The idea is that before running, any functional system performs some test in order to make sure that it has the right permissions. The use of passwords is a typical example of such an approach. More robust ideas have also been introduced in two main directions. On the one hand, the test could check the presence of a cryptographically-enhanced security chip (rather than a simple password), as typically suggested in [3,22]. On the other hand, one can also take advantage of Physically Unclonable Functions, in order to make sure that the system is running on the correct (unique) piece of hardware, e.g. as suggested in [16,29].

A second active line of research is to exploit watermarking techniques. In general, a watermark is a piece of information that is embedded in a digital signal, in order to verify its authenticity or the identity of its owners. Its main evaluation criteria are *robustness* (i.e. the watermark should be detectable even if the digital signal is slightly modified by some processing, e.g. filtering), *imperceptibility* (i.e. the original signal and the masked signal should be perceptually close) and *capacity* (i.e. the embedded mark should be large). By contrast to the permission-based approach, watermarking is useful a posteriori, in order to detect illegal copies at lower cost than reverse engineering. While first investigated in the context of image processing, watermarking has also been applied for software IP protection [7,25]. More recently, it has attracted attention for hardware IP protection, e.g. [1,18,19].

Two types of limitations can be highlighted for these approaches. The first one mainly applies to the performance and security of permission-based protections. That is, in order to include such mechanisms in a circuit, one needs to modify its original layout. This means that some logic gates are lost (and consume power) for this purpose. Besides, as permission checks imply the addition of some circuitry in the chips, an adversary who has access to the source code (by reverse engineering or any other means) can remove this part of the designs, hence producing illegal copies without any protection. The second limitation is more general and relates to the flexibility of the solutions. Namely, for both types of IP protections, the decision to protect an implementation has to be part of the design process (and must sometimes be made early in this process). Hence, it cannot help for already deployed products.

In parallel, security enhancements that are specific to high-level IP and take advantage of software facilities have also been proposed. One typical example is source code or bitstream encryption that is used, e.g. in FPGA [9]. While this solution is useless in the case of hardwired IP, as there is no code to encrypt/decrypt, it can be a useful complement in some IP protection systems, where a part of the value to preserve is software. Note that the encryption/decryption of the source code or bitstream then has to be secure against other types of physical attacks, e.g. using side-channels [24]. It is also necessary that the encryption/decryption keys are stored in a sufficiently secure memory (a problem for which the tamper resistant property of PUF could help [31]).

In this paper, we propose a new IP protection scheme that applies both to high-level and low-level IP. It mitigates some of the limitations, and can be efficiently combined with, previously introduced solutions for this purpose. More precisely, we take advantage of two main ingredients. First, we build on the idea of soft (aka robust, aka perceptual) hash functions, that are yet another tool used to protect the copyright of digital data (e.g. images) [13,21,23,32]. Second, we exploit the physical representation of the IP, e.g. through its power consumption, as proposed with side-channel watermarks [4,35]. In addition, we use the formalism introduced in the context of Physically Unclonable Functions [2], in order to specify the components of our IP detection infrastructure. Combining these elements, we introduce soft physical hash functions as a powerful yet flexible way to deal with the IP of microelectronic circuits and systems. We then define the main properties of such functions, namely their *content sensitivity* and *perceptual robustness*. Finally, we instantiate a first soft physical hash function with simple signal processing tools and analyze its properties based on an experimental case study. Namely, and based on the implementation of 10 different block ciphers in an Atmel microcontroller that have been made public through the ECRYPT project [11], we show experimentally that a soft physical hash function can efficiently discriminate these different implementations, while being robust to some elementary code transformations that should not modify the IP (perceptually).

2 IP in Hardware and Software Design Flows

This section briefly introduces the hardware and software design flows, and discusses where the IP lies in such systems. We conclude by describing two general attacks against IP in integrated systems. The hardware design flow is illustrated in the left part of Figure 1. It is essentially made of five successive steps for which a succint description is given next.

1. *Description.* This step consists in specifying the implementation of an algorithm with a Hardware Description Language (HDL), e.g. VHDL and Verilog. Such languages determine the number and type of operations that have to be implemented on chip, as well as their scheduling. The result of this step is called the hardware source code.
2. *Synthesis.* This step consists in converting the hardware source code into an idealized electronic circuit. The synthesis tools automatically translate a high-level description into a lower-level one (e.g. gate-level). It is functionally equivalent to the hardware source code and corresponds to the target platform (e.g. ASIC or FPGA). The result of this step is called a netlist. Netlists are idealized in the sense that they do not correspond to a physical description of the chip. Yet, the result of synthesis is highly dependent on the tool options (e.g. space- or time-oriented optimizations).
3. *Place and route.* At this point of the design flow, the automated tools build a physical model for all the logic and memory elements of the netlist. They

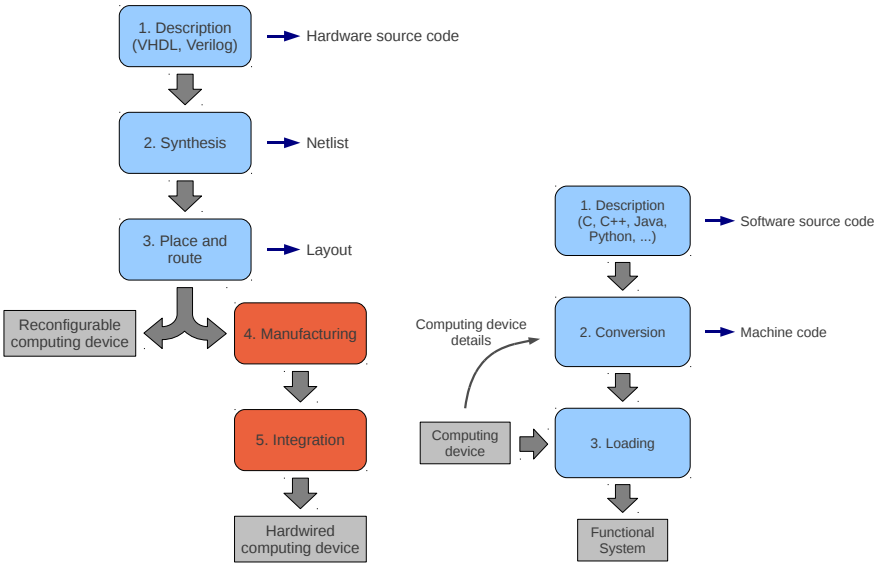

Fig. 1. Hardware (left) and software (right) design flows.

also simulate their mapping on the circuit surface, taking the position of each cell, the width and length of connections, ... into account. The result of this step is called the layout. It is again dependent on the tool options.

Note that for reconfigurable devices such as FPGA, the design flow essentially stops here (i.e. moves to final Step 5). Namely the layout can be directly loaded into the device that will next act as a dedicated circuit with the same functionality. By contrast for ASIC, an additional Step 4 has to be carried out:

4. *Manufacturing.* Once the design of the chip is finished, its layout is sent to a manufacturer in order to be physically built. Manufacturing usually includes a packaging step that consists in protecting the chip with a plastic coating.
5. *Integration.* Eventually, and optionally, the chip can be integrated into a larger system, combining different electronic pieces together.

In general, most systems used in current applications combine (hardwired or reconfigurable) computing devices with software programming. Software is used either to control the dedicated hardware, or to perform some general purpose computations for which standard computing platforms (microcontrollers, microprocessors) provide sufficient performance. The software design flow is illustrated in the right part of Figure 1. It is essentially made of three steps that we describe next.

1. *Description.* As for the hardware flow, a high-level description language is used (e.g. C, C++, Java, ...). It specifies a sequence of (sometimes data-dependent) operations to execute on some fixed hardware in order to perform an algorithmic task. The result of this step is called the software source code.

2. *Conversion.* This step consists in converting the high-level software source code into a lower-level description, denoted as the machine code. As for the hardware flow, it exploits automated tools (i.e. interpreters or compilers) the options of which may significantly affect the final performances.
3. *Loading.* This last step consists in writing the machine source code into the computing device memory, from which instructions are executed. As a result, a fully functional system is obtained and ready to use in actual applications.

Where Does the IP Lie? From the previous design flows, it appears that different types of IP are manipulated by the functional systems that are used in final applications. On the one hand, the source codes contain the high-level IP. They are easy to manipulate and provide the most readable description of algorithmic tricks used to obtain efficient implementations. In addition, they are extremely *malleable*, as it is easy to slightly modify them, or to run again the synthesis and place and route (resp. conversion) steps, in order to produce two computing devices (resp. systems) with the same functionality and slightly different layouts (resp. machine codes). On the other hand, the layout and machine codes contain the low-level IP that is less malleable but possibly includes further optimization efforts performed during the synthesis and place and route (resp. conversion) steps. The netlist in a hardware design flow falls between these two extremes.

Attacks against the IP. Leaving technical details aside, two main types of attacks can be mounted to violate IP. First, "recovery and clone" attacks imply that the adversary just re-uses the stolen IP as such. They typically target the layouts or machine codes. Second, "recovery and modify" attacks imply that the adversary modifies the IP before re-using it in an illegal system. They typically target source codes. Clearly, the second attacks are much more powerful and, in general, harder to prevent. As mentioned in the introduction, permission-based IP protections are useless against such attacks, as the adversary can remove them in his modified design. But in cases where a watermark is somewhat "additive" (e.g. [4,35]), the risk that an adversary identifies the watermark insertion and removes it also increases. In practice, there exist many ways in which attacks against the IP can be implemented. Reverse engineering generally comes in the first place for ASIC. A rather complete survey is given in [30]. One can also mention approaches taking advantage of side-channel analysis [8,12,17,27]. But threats caused by dishonest manufacturers, producing more chips than licenced, or industrial IP theft, are other examples. There finally exist many attacks that are specific to one technology. For example, bitstream security is a serious issue for SRAM-based FPGA [34]. In order to remain generic, most of our discussions in the following section are independent of the technical solutions used to perform IP theft.

3 A New Approach Based on Soft Hash Functions

Looking at previous works, the protection of IP in integrated circuits and systems appears to easily take advantage of general solutions for the protection of

digital data. As a result, and besides the watermarking and permission-based techniques, it is natural to investigate other tools that have been proposed to prevent the piracy of digital images. Following, we propose to exploit robust hash functions [13,32], also known as soft hash functions [21] or perceptual hash functions [23], for this purpose. We will use the term "soft hash functions" in the rest of the paper. By contrast to cryptographic hash functions, for which the output string is highly sensitive to small perturbations of the input, soft hash functions are such that similar objects should return highly correlated digests (i.e. be *perceptually robust*), while different objects should produce uncorrelated ones (i.e. be *content-sensitive*). Soft hash functions exist in keyed and un-keyed version. We will only consider the second category and leave the design of a keyed version as a scope for further research.

In practice, the soft hash functions we will consider are generally made of two main ingredients. First, a *feature vector evaluation* phase outputs an intermediate response that is expected to represent the object to characterize in the most accurate manner. In other words, this intermediate string should be very content-sensitive. Second, an *extraction phase* should apply signal processing and summarize the feature vector into a smaller output hash value that best trades content sensitivity for perceptual robustness. In addition, soft hash function infrastructures use a *detection tool* in order to compare different hashes and determine whether they correspond to the same object. The main idea that we propose in this paper is to extract a physical feature from the objects to protect. We denote the resulting tool as a *Soft Physical Hash function* (SPH) that will be the main component of our *IP detection infrastructure*. Interestingly, SPH can be viewed as a particular type of physical function systems, as formally defined in [2]. But whereas this previous work focuses on the robustness, unclonability and unpredictability of physical functions, we rather care about the previously mentioned perceptual robustness and content sensitivity. Relations with the formal definitions from [2] will be given in the next section. Intuitively, an IP detection infrastructure can be specified as follows:

1. *Object to protect.* This could be any type of IP, namely a source code, a netlist, a layout, or a machine code, as defined in Section 2.
2. *Physical feature vector evaluation.* This could be any physical emanation of the device running the IP. For example, side-channels such as the power consumption [20] or the electromagnetic radiation [14] are natural candidates.
3. *Extraction.* This could be any signal processing outputting a hash value that best represents the IP. For example, one could compress by selecting the "points of interest" in a side-channel measurement trace.
4. *Detection.* This could be any statistical tool that allows determining the level of similarity between two hash values. Again, techiques from the side-channel attack community could help, e.g. correlation [5] or template attacks [6].

In addition, one generally has to consider the context in which the IP protection mechanism will be applied. Namely, we could consider:

1. *Known or unknown inputs.* That is, can the IP claimer select the data that is manipulated by the target device during the physical feature evaluation (in which case the soft hash can characterize both operation and data dependencies)? Or is he limited to executions on random, and possibly unknown, inputs (in which case the soft hash only characterizes operation dependencies)?
2. *Known or unknown source code.* That is, does the IP claimer know the full source code he is trying to detect or does he only have access to its soft physical hash value (e.g. if the IP claimer is a third party)?
3. *Same or different technologies.* That is, are the soft hash values to be compared extracted from devices in the same technology or not?
4. *Same or different setups.* That is, are the soft hash values to be compared extracted with the same measurement apparatus or not?

Obviously, certain scenarios are more challenging than others and the perceptual robustness and content sensitivity of a SPH can significantly vary accordingly. Before entering into more details and instantiation issues, we would like to point out a few advantages of the proposed approach. In the first place, and contrary to the permission-based and watermarking approaches, using SPH does not require to modify the designs a priori. This allows improved flexibility, as one can decide even after implementation to exploit this idea for IP theft detection. It also means that implementing the protection mechanism implies no performance overhead at all. Second, SPH theoretically allow preventing some recovery and modify attacks. This is due to the fact that there is nothing to remove from a protected design, but the IP itself (i.e. the protection is fully intrinsic). Of course, the detection phase will be significantly more challenging when modifications are substantial (because content sensitivity is inherently limited once modified designs become too different), but it can at least tolerate some malleability in the objects to protect. Third, the proposed solution remains cheaper than reverse engineering, as it only requires to measure a physical feature.

4 Definitions

4.1 The IP Detection Infrastructure

The main goal of this section is to provide some formalism to define and study the aforementioned IP detection procedure, based on the idea of SPH. This procedure naturally fits in the generic framework for physical functions presented in [2], that is depicted in Figure 2. Indeed, SPH perfectly correspond to the definition of *physical function systems*. Hence, we reuse and lighten the formalism provided in [2] and apply it to our IP detection using SPH. The generic framework we propose for IP detection is illustrated in Figure 3, where a *soft physical hash function* is a *probabilistic procedure* SPH(IP, X) that returns a *hash value* h.

$$\mathrm{SPH} : (\mathrm{IP}, X) \mapsto h.$$

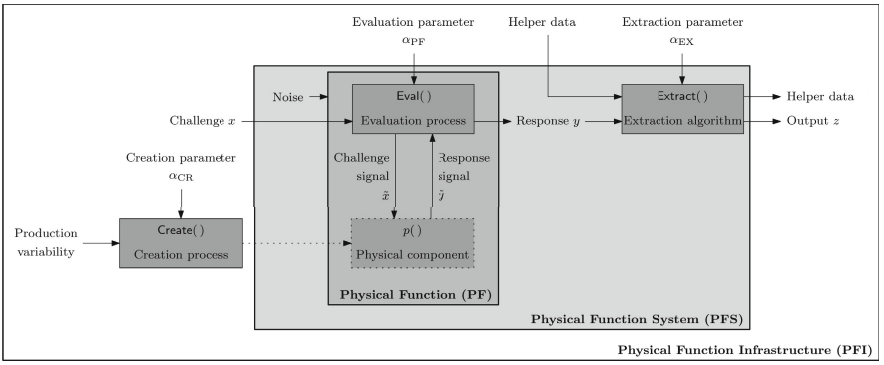

Fig. 2. Generic framework for physical functions [2]

Such outputs are obtained in two steps. First, a *physical feature vector* is obtained (with the Eval(·) procedure), corresponding to the measurement of some physical quantity observed during the processing of data X (that may contain more than one input) by a device running the given IP. The probabilistic nature of a SPH comes from this procedure and is due to many different reasons, such as the variability that can be observed between different devices running the same IP, or the noise in the measurements. This part of the system is the one bringing the *content sensitivity* property but, as just stated, there is a non-negligible variability that should be removed for reaching *perceptual robustness*. This is precisely the role of the Extract(·) procedure. During this second step, the physical feature vector is processed to extract the robust information it contains, returning a hash value that should characterize the IP running on the device.

Next, an *IP detection infrastructure* is composed of two *soft physical hash functions* $\mathrm{SPH}(\mathrm{IP}_{\mathrm{ref}}, \cdot)$ and $\mathrm{SPH}(\mathrm{IP}_{\mathrm{sus}}, \cdot)$, corresponding to the reference and the suspicious devices. Each SPH outputs a hash value that should identify the IP running on the device under test. Additionally to these two SPH, a detection tool $\mathrm{Detect}(\cdot, \cdot)$ is used to compare the hash values h_{ref} and h_{sus} returned by the SPH. This procedure outputs a *similarity score* that should be close to 1 for similar IP, and close to 0 for different ones:

$$\mathrm{Detect} : (h_{\mathrm{ref}}, h_{\mathrm{sus}}) \mapsto s \in [0, 1].$$

Performance metrics are required to evaluate SPH. In the following section, we define *perceptual robustness* and *content sensitivity*, so that we will be able to quantify these properties and to determine relevant tools for extracting hash values.

4.2 Performance Metrics

The IP detection infrastructure aims at confirming/invalidating the presence of an IP running on a suspicious device. As mentioned earlier in the paper, this

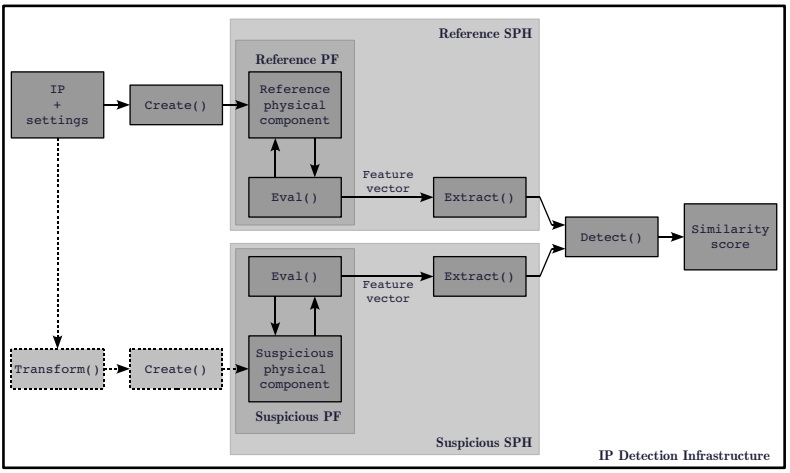

Fig. 3. Generic framework for IP detection

IP can be of different nature, such as a layout or source code for instance. The main difficulty for formally defining our problem comes from the fact that some modifications of the source code should not be considered as defining a new IP. For example, trivial modifications (e.g. changing variable names, switching registers, adding dummy operations, changing compiler options) can clearly be regarded as preserving the IP, but other modifications such as code optimizations can also be considered as improving, yet relying on, the initial IP. Defining the boundary between IP reuse and new IP definition is outside the scope of this paper[1]. Here we assume the existence of such a definition and concentrate on providing a tool allowing us to decide whether a given device is based on a given IP or not, in the sense of this definition. More formally, we consider a set $\mathcal{F}_{pre}$ of *IP-preserving transformations*. The detection tool should consider as "close" an IP and its transform if the transformation function belongs to $\mathcal{F}_{pre}$, and as "distant" if the transformation belongs to $\bar{\mathcal{F}}_{pre}$.

Remark. The use of sets $\mathcal{F}_{pre}$ and $\bar{\mathcal{F}}_{pre}$ is a purely formal way of dealing with the perceptual notion of closeness involved in this problem. These sets should be complementary but obviously, in practice, we will only consider some transformations in both sets. Notice that $\bar{\mathcal{F}}_{pre}$ may contain modifications of an IP, but also transformations that simply consist in changing the IP. For instance, let IP_1 be the IP corresponding to the source code of a social network and IP_2 be the one corresponding to a plane autopilot. Then, formally, there exists $\bar{F} \in \bar{\mathcal{F}}_{pre}$ such that $\mathrm{IP}_1 = \bar{F}(\mathrm{IP}_2)$. Nevertheless, and as will be clear in the next section, such a definition is useful to capture practical transforms that can be considered as IP-preserving, while providing the flexibility to evaluate a wide range of scenarios.

[1] In Section 5 we will run experiments on typical examples, where, on the one hand, IPs are modified and reused, and, on the other hand, different IPs are considered.

As a result, the robustness and sensitivity of a SPH are related to the set of transformations considered to be IP-preserving, to the detection tool that is used, to the threshold τ that has been fixed to make the decision, and to the data complexity N corresponding to the number of IP runs that will be measured. We denote by $\mathcal{X}_N$ the set of N-input requests that can be sent to the devices and define our evaluation metrics as follows.

Definition 1. *(τ-perceptual robustness): Let $\tau \in [0,1]$, the τ-perceptual robustness of a SPH relatively to a detection tool* Det*, a set of transformations $\mathcal{F}_{\text{pre}}$ and a data complexity N, is the probability that the similarity score of a reference IP and its transformation by $F \in \mathcal{F}_{\text{pre}}$ is larger than τ:*

$$\text{PeRo}_{\tau}^{N}(\text{Det}, \mathcal{F}_{\text{pre}}) \triangleq \Pr_{\text{IP}, X \in \mathcal{X}_N, F \in \mathcal{F}_{\text{pre}}} \left[\text{Det}\left(\text{SPH}(\text{IP}, X), \text{SPH}(F(\text{IP}), X)\right) \geq \tau\right].$$

Definition 2. *(τ-content sensitivity): Let $\tau \in [0,1]$, the τ-content sensitivity of a SPH relatively to a detection tool* Det*, a set of transformations $\mathcal{F}_{\text{pre}}$ and a data complexity N, is the probability that the similarity score of a reference IP and its transformation by $\bar{F} \in \bar{\mathcal{F}}_{\text{pre}}$ is smaller than τ:*

$$\text{CoSe}_{\tau}^{N}(\text{Det}, \bar{\mathcal{F}}_{\text{pre}}) \triangleq \Pr_{\text{IP}, X \in \mathcal{X}_N, \bar{F} \in \bar{\mathcal{F}}_{\text{pre}}} \left[\text{Det}\left(\text{SPH}(\text{IP}, X), \text{SPH}(\bar{F}(\text{IP}), X)\right) \leq \tau\right].$$

An IP detection infrastructure aims at combining an SPH to a relevant detection tool in order to ensure good performances regarding both perceptual robustness and content sensitivity. This context is the one of binary hypothesis testing where τ corresponds to a threshold, $1 - \text{PeRo}$ is non-detection error probability and $1 - \text{CoSe}$ is the false-alarm error probability.

5 First Experimental Results

This section demonstrates the use of SPH based on power consumption traces to perform IP detection. First, we propose a simple instance of IP detection infrastructure, that uses classical tools in signal processing and side-channel attacks. Then, we provide experimental results that confirm the content sensitivity and perceptual robustness of this instance. These results are obtained for a particular set of IP and some representative code transformations that we considered as IP-preserving.

5.1 An Exemplary Instance

We first present our instance by specifying the four elements that define an IP detection infrastructure enumerated in Section 3. Then, we detail the context we consider for detecting IP. Finally, we provide information about our measurement setup.

Object to protect. As a first case study, we investigated the relevance of our IP detection based on SPH in the context of software implementations. In particular, we took advantage of the cryptographic algorithms made available as open source codes in [11]. We considered the assembly codes of these lightweight ciphers as IP to protect, in order to analyze situations ranging from very similar algorithms to totally different ones.

Physical feature evaluation. Inspired by side-channel attacks, we characterize an IP by measuring the power consumption of a device running this IP. As the literature on power analysis attacks suggests, such a physical feature vector is expected to be highly content sensitive.

Extraction. We somehow need to compress the feature vector, removing part of the variability due to the "noise"[2], while keeping the content sensitive information. The Fourier transform is a natural candidate for such compressing and filtering purposes. The technique we propose is to apply a Fast Fourier Transform (FFT) to the traces obtained, to take its modulus, and to only extract the values corresponding to frequencies below the clock frequency. Information in higher frequencies does not refer to logic gates but to the device itself. We provide below (Figure 5) results that emphasize the positive effect of this preprocessing.

Detection. As a detection tool, we propose to use the widespread Pearson correlation coefficient that has shown its interest in many previous works in the domain of power-based cryptanalysis [5]. Let us recall that the Pearson correlation coefficient between two vectors x and y of length n having mean values $\bar{x}$ and $\bar{y}$ is defined as:

$$\rho(x,y) \triangleq \frac{\sum_{i=1}^{n}(x_i-\bar{x})\cdot(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}.$$

Context. As mentioned in Section 3, the IP detection infrastructure can be used in different contexts. We specify the settings of our experimental work as follows. First, inputs to the device are unknown to the detection infrastructure and a single trace is used to perform the detection. Second, the source code is also unknown to the IP detection infrastructure. Third, the same technology, i.e. an Atmel 644P, has been used to perform all the measurements. Finally, the same measurement setup has been used for all these measurements.

Measurement setup. We obtained power consumption traces by measuring the voltage variations around an inductance at the input of the power supply of the microcontroller. The clock frequency of the controller was fixed to 20 MHz, and the sampling frequency to 50 MHz. For each IP, a reference trace corresponding to a single encryption was stored. Next, the suspicious traces were cut or padded

[2] We stress that the term noise is generic. It can refer to measurement noise, but also to noise due to minor transformations of the code or the use of different inputs for the evaluation phase.

(by re-encrypting the ciphertext), in order to deal realistically with the (frequent) situation were IP have different cycles counts[3].

5.2 Testing Content Sensitivity

Using Different Ciphers. As a first step, we applied our instance of SPH on different ciphers. We chose to focus on the 10 block ciphers in Table 1. This first experiment aims at confirming that the proposed IP detection infrastructure indeed provides content sensitivity and perceptual robustness in a simple case, where the set of IP-preserving transformations $\mathcal{F}_{\text{pre}}$ is reduced to identity, and the set $\bar{\mathcal{F}}_{\text{pre}}$ consists in choosing another cipher in the list.

Table 1. Table of ciphers used as IP

AES	DESXL	HIGHT	KASUMI
KATAN	MCRYPTON	NOEKEON	PRESENT
SEA	TEA		

For each of the 100 possible couples of IP (reference, suspicious) we measured 20 traces (corresponding to different inputs) for both IP, and applied the IP detection tool for each of the 400 possible pairs of traces. The similarity scores obtained are plotted in Figure 4.

Black dots are the scores obtained when IP are compared to $F(\text{IP})$ with $F \in \mathcal{F}_{\text{pre}}$ and gray dots are the one obtained when the transformation belongs to $\bar{\mathcal{F}}_{\text{pre}}$. This picture is encouraging : both content sensitivity and perceptual robustness are reached in experiments using a single trace. Indeed, for any τ value between 0.86 and 0.27, the detection rule makes no mistake for this particular set of IP and these particular sets $\mathcal{F}_{\text{pre}}$ and $\bar{\mathcal{F}}_{\text{pre}}$.

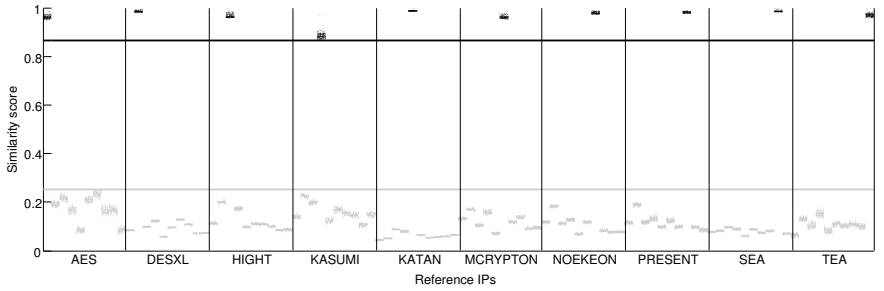

Fig. 4. Similarity scores for different IP

[3] We first ran tests with a single execution of the IP but then it was too easy to detect IPs. Since a cipher is generally used to encrypt more than one block we chose to perform tests on looping IPs.

Remark on KASUMI. The reader may have noticed that almost all reference ciphers lead to scores higher than 0.95, except for KASUMI. The explanation is quite simple. Indeed, the execution path of KASUMI depends on the data being processed. To avoid timing attacks, cryptographers usually add `nop` operations to make the implementation time-constant. While this prevents the execution time from being data-dependent, this is not the case of instructions performed: traces obtained by processing different outputs will correspond to different instruction sequences. This explains why we observe these smaller scores for KASUMI. Note also that a few scores are larger than 0.9 in Figure 4: they correspond to the comparison of two traces obtained with the same input.

Using Different Implementations of a Same Cipher. As the setting of Figure 4 is quite favorable (since we are comparing totally different IP), we now move to a more challenging context and consider a single block cipher, namely the standard AES, and four of its implementations. Three of them are the *Fast, Furious* and *Fantastic* implementations that can be found in [26]. The fourth one is proposed in [11] and named *Francesco* in reference to the first-name of its programmer. Again, for each of the 16 possible couples of reference/suspicious IP, we performed 2500 experiments (corresponding to 50 traces for both IP). To highlight the positive effect of using an FFT, we plotted the results obtained with/without such extraction, in the right/left parts of Figure 5.

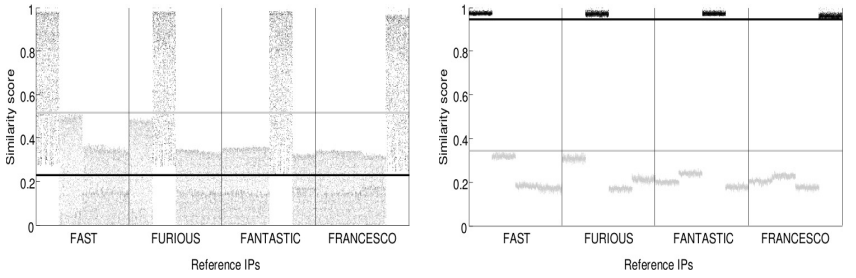

Fig. 5. Similarity scores for different AES implementations using the FFT (right) or not (left)

As can easily be seen, and as expected, using the FFT in the extraction step drastically decreases the variance observed in the similarity scores in the case where traces are directly compared (left part), while keeping the content sensitivity property. Indeed, each implementation should be considered as a different IP since they are really different one from the other. This emphasizes the need of an extraction tool that compresses the feature vector to ensure robustness: using FFT there is a wide range of thresholds (namely 0.37 to 0.95) for which both perceptual robustness and content sensitivity are equal[4] to 1 (that is, a 100%

[4] Obviously these values are only estimates since one cannot consider all possible IPs and all possible code modifications.

success rate) while without this tool, any value of the threshold will induce a non-zero error probability.

Yet, it remains that the content sensitivity could come from the simple fact that each implementation requires a different number of cycles to perform an encryption. In order to rule out this possibility, we additionally tuned the *Fast* and the *Furious* implementations in such a way that they have the same number of clock cycles and their only differences consist in actual round variations. For this purpose, the key expansion, first key addition, and final rounds remained the same and the inner rounds of the *Fast* implementation has been padded with `nop` operations. As can be observed in Figure 6, and even in this case, both implementations are detected as different IP with, again, a large range of thresholds leading to perceptual robustness and content sensitivity of 1.

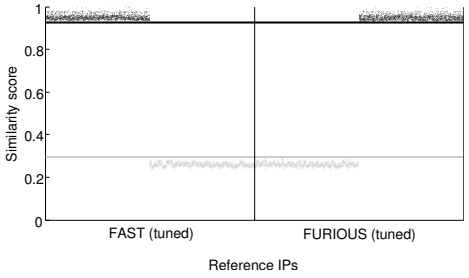

Fig. 6. Similarity scores for tuned versions of *Fast* and *Furious* AES implementations

5.3 Testing Perceptual Robustness

The previous experiments exhibit the good behavior of our SPH instance regarding content sensitivity. We now focus on its perceptual robustness. That is, can we still detect an IP when minor modifications have been applied to the original code? As previously mentioned, testing perceptual robustness requires to agree on a set of *IP-preserving* transformations. We identified two types of transformations, depending on whether they alter the performances or not:

- `addr`: i.e. changing registers or SRAM **addr**esses (non-altering);
- `swap`: i.e. **swap**ping instructions if possible (non-altering);
- `dumo`: i.e. adding **dum**my operations **o**ut of the rounds (altering);
- `dumi`: i.e. adding **dum**my operations **i**n the rounds (altering).

We chose to apply these modifications to the *Furious* implementation of AES. Our results are given in Figure 7. Dots correspond to scores obtained when comparing the reference implementation (`ref`) to its modification by one or more of the listed transformations. The number of added dummy-operations is the same for both `dumo` and `dumi` (57 additional cycles). As can be seen, the proposed instance is directly robust regarding all non altering transformations. Concerning the addition of dummy operations, we can observe that increasing

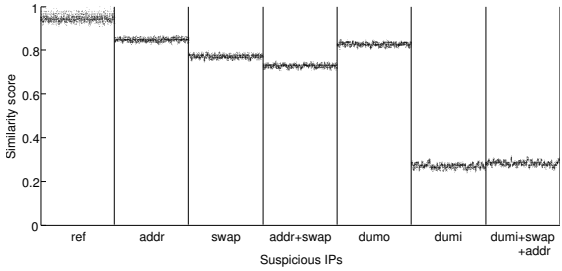

Fig. 7. Similarity scores for transforms of the *Furious* implementation

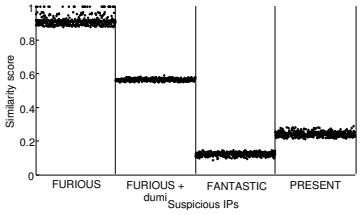

Fig. 8. Improved similarity scores for the *Furious* implementation, its `dumi` transform and other IP

the encryption time does not always critically decrease the robustness (cf. `dumo`) but, that adding dummy operations inside a repetitive part of the cipher is more damaging. This observation naturally relates more to the simple nature of our extraction and detection procedures than to the very idea of SPH. Interestingly, simple tweaks allow to deal with such more challenging contexts, as illustrated in Figure 8.

In this final figure, we slightly modified our SPH in order to first identify the block cipher rounds (using cross-correlation). Next, we applied the FFT separately on the different rounds, and used the average correlation over these rounds as detection procedure. While this approach remains clearly heuristic, the experimental results in Figure 8 show that it allows discriminating the addition of dummy operations within the rounds from the move towards another implementation of the AES.

To conclude, this set of experiments suggests that the concept we propose is valid and provides good results even when it is instantiated with classical tools. Moreover, we have shown that more challenging scenarios can be handled using more advanced techniques. A deeper investigation of these scenarios is the natural next step now that this preliminary work has shown promising results.

6 Conclusion and Future Works

This paper introduced, defined and proposed a first instantiation of SPH infrastructure, as a flexible and efficient way to protect the IP of integrated systems.

Experimental results in the context of software implementations confirm the relevance of the approach. Namely, we are able to discriminate proprietary IP using our instance of SPH, in the challenging scenario where the source code and inputs of the IP to protect remain unknown to the detection infrastructure, with minimum data complexity. Quite naturally, the strong contribution of the control logic in a software implementation makes it a target of choice for our IP detection infrastructure. Hence, the investigation of other contexts such as FPGA, hardware or more complex implementations is an interesting scope for further research. In view of the strong results obtained with simple tools in software, we expect the method to remain applicable in those cases, in particular in known input scenario (taking advantage of the strong literature in side-channel attacks) and/or if the source code is available to the detector (in order to take advantage of Hidden Markov Model cryptanalysis techniques such as in [33,10]). Eventually, another potentially interesting research direction would be the definition of design strategies (or code modifications) that could be applied to the IP before its release in order to facilitate the future detection of this IP with SPH.

Acknowledgements. This work has been funded in part by the ERC project 280141 (acronym CRASH) and the Walloon Region MIPSs project. Stéphanie Kerckhof is a PhD student funded by a FRIA grant, Belgium. F.-X. Standaert is a Research Associate of the Belgian Fund for Scientic Research (FNRS-F.R.S).

References

1. Abdel-Hamid, A.T., Tahar, S., Aboulhamid, E.M.: A survey on IP watermarking techniques. Design Autom. for Emb. Sys. 9(3), 211–227 (2004)
2. Armknecht, F., Maes, R., Sadeghi, A.-R., Standaert, F.-X., Wachsmann, C.: A formalization of the security features of physical functions. In: IEEE Symposium on Security and Privacy, pp. 397–412. IEEE Computer Society (2011)
3. Baetoniu, C.: FPGA IFF copy protection using Dallas semiconductor/Maxim DS2432 secure EEPROMs. XAPP780 (May 28, 2010)
4. Becker, G.T., Kasper, M., Moradi, A., Paar, C.: Side-channel based watermarks for integrated circuits. In: Plusquellic, J., Mai, K. (eds.) HOST, pp. 30–35. IEEE Computer Society (2010)
5. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
7. Collberg, C.S., Thomborson, C.D.: Watermarking, tamper-proofing, and obfuscation-tools for software protection. IEEE Trans. Software Eng. 28(8), 735–746 (2002)
8. Daudigny, R., Ledig, H., Muller, F., Valette, F.: SCARE of the DES. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 393–406. Springer, Heidelberg (2005)

9. Drimer, S.: Authentication of FPGA Bitstreams: Why and How. In: Diniz, P.C., Marques, E., Bertels, K., Fernandes, M.M., Cardoso, J.M.P. (eds.) ARCS 2007. LNCS, vol. 4419, pp. 73–84. Springer, Heidelberg (2007)
10. Durvaux, F., Renauld, M., Standaert, F.-X., van Oldeneel tot Oldenzeel, L., Veyrat-Charvillon, N.: Cryptanalysis of the ches 2009/2010 random delay countermeasure. Cryptology ePrint Archive, Report 2012/038 (2012), http://eprint.iacr.org/
11. Eisenbarth, T., Gong, Z., Güneysu, T., Heyse, S., Kerckhof Sebastiaan Indesteege, S., Koeune, F., Nad, T., Plos, T., Regazzoni, F., Standaert, F.-X., van Oldeneel tot Oldenzeel, L.: Compact implementation and performance evaluation of block ciphers in ATtiny devices (2011)
12. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a side channel based disassembler. Transactions on Computational Science 10, 78–99 (2010)
13. Fridrich, J., Goljan, M.: Robust hash functions for digital watermarking. In: ITCC, pp. 178–183. IEEE Computer Society (2000)
14. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
15. Goubin, L., Matsui, M. (eds.): CHES 2006. LNCS, vol. 4249. Springer, Heidelberg (2006)
16. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
17. Guilley, S., Sauvage, L., Micolod, J., Réal, D., Valette, F.: Defeating Any Secret Cryptography with SCARE Attacks. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 273–293. Springer, Heidelberg (2010)
18. Kahng, A.B., Lach, J., Mangione-Smith, W.H., Mantik, S., Markov, I.L., Potkonjak, M., Tucker, P., Wang, H., Wolfe, G.: Watermarking techniques for intellectual property protection. In: DAC, pp. 776–781 (1998)
19. Kahng, A.B., Mantik, S., Markov, I.L., Potkonjak, M., Tucker, P., Wang, H., Wolfe, G.: Robust IP watermarking methodologies for physical design. In: DAC, pp. 782–787 (1998)
20. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
21. Lefèbvre, F., Czyz, J., Macq, B.M.: A robust soft hash algorithm for digital image signature. In: ICIP (2), pp. 495–498 (2003)
22. Linke, B.: Xilinx FPGA IFF copy protection with 1-wire SHA-1 secure memories. XAPP3826 (July 21, 2006)
23. Monga, V., Evans, B.L.: Perceptual image hashing via feature points: Performance evaluation and tradeoffs. IEEE Transactions on Image Processing 15(11), 3452–3465 (2006)
24. Moradi, A., Barenghi, A., Kasper, T., Paar, C.: On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-ii FPGAs. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM Conference on Computer and Communications Security, pp. 111–124. ACM (2011)
25. Myles, G.: Using software watermarking to discourage piracy. ACM Crossroads 12(1), 4 (2005)
26. B. Poettering. Fast AES implementation for Atmel's AVR microcontrollers, http://point-at-infinity.org/avraes/
27. Réal, D., Dubois, V., Guilloux, A.-M., Valette, F., Drissi, M.: SCARE of an Unknown Hardware Feistel Implementation. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 218–227. Springer, Heidelberg (2008)

28. Semico Research. Semiconductor intellectual property: The market hits its stride, http://www.design-reuse.com/news/11069/semico-research-report-semiconductor-%intellectual-property-market-hits-stride.html
29. Simpson, E., Schaumont, P.: Offline hardware/software authentication for reconfigurable platforms. In: Goubin, Matsui (eds.) [15], pp. 311–323
30. Torrance, R., James, D.: The State-of-the-Art in IC Reverse Engineering. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 363–381. Springer, Heidelberg (2009)
31. Tuyls, P., Schrijen, G.J., Skoric, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-proof hardware from protective coatings. In: Goubin, Matsui (eds.) [15], pp. 369–383
32. Venkatesan, R., Koon, S.-M., Jakubowski, M.H., Moulin, P.: Robust image hashing. In: ICIP (2000)
33. Walter, C.D., Koç, Ç.K., Paar, C. (eds.): CHES 2003. LNCS, vol. 2779. Springer, Heidelberg (2003)
34. Wollinger, T.J., Guajardo, J., Paar, C.: Security on FPGAs: State-of-the-art implementations and attacks. ACM Trans. Embedded Comput. Syst. 3(3), 534–574 (2004)
35. Ziener, D., Teich, J.: Power signature watermarking of IP cores for FPGAs. Signal Processing Systems 51(1), 123–136 (2008)