Keyed Side-Channel Based Hashing for IP Protection using Wavelets

Timo Bartkewitz

Horst Görtz Institute for IT-Security,
Ruhr-University Bochum,
Universitätsstraße 150, 44801 Bochum, Germany
`timo.bartkewitz@rub.de`

Abstract. The protection of intelligent property (IP) is a challenging task, especially in secured embedded systems where program code that is supposed to be a plagiarism cannot be simply read-out for further inspection. This is even more aggravated if the original source code was modified to prevent comparisons of any kind. For instance, watermarks that are actually hidden in the code are at risk to be rendered useless if the attacker has full access to the original code and some knowledge about the watermark. The unlicensed use of patented algorithms is a further problem that belongs to IP plagiarism as well. A Recent work presented a framework based on perceptual hashing to detect intelligent property in hardware and software designs. In this work we consequently extend this framework to detect IP plagiarism in embedded systems that can reliably match contents even in the presence of attacks. Therefore, we propose an adapted signal feature extraction method, the wavelet transform, to form a keyed side-channel hash function.

Keywords: Side-Channel, IP Protection, Embedded Systems, Perceptual Hashing, Software Plagiarism

1 Introduction

The plagiarism of intelligent property is without doubt a serious threat to the software industry. In 2011 almost 42% of the worldwide distributed software was pirated incorporating a financial damage of over 63 billion US dollars [6]. Although these numbers mainly concern general purpose computer software they are likely to be a good indicator for embedded software as well. The embedded software industry, however, has virtually no opportunity to confirm a suspected plagiarism. Products on the market are equipped with memory read-out protections, and thus a legitimated IP holder can only evaluate suspicious embedded software on the basis of its functionality but providing no indications.

However, side-channels can assist plagiarism detection in embedded systems. Side-channels arise from the fact that electronic devices emit physically observable quantities, for instance the power consumption, electromagnetic emanation, or timings. These quantities possess information about internal states of the device which are actually exploited to extract a secret information, *e.g.* a cryptographic key. The authors in [3] applied the idea of the correlation power analysis attack [5] to embed a watermark in assembler source code of the Keeloq algorithm. In [4] the power consumption of a microcontroller is used to determine the Hamming weight in each cycle that corresponds to a prefetch of a certain instruction. Thus, the resulting map of Hamming weights can be matched with a second map to expose the similarity of two program codes. Obviously, these approaches rely on the assumption that the watermark code will not be removed, especially since it is non-functional code, and the program source code is subject to minor changes only.

In [7] the authors use a relaxed formalism for security features of physical functions defined in [2]. In [7] however, it is centered around the physical representation of the intelligent property, *e.g.* the power consumption or intrinsic hardware properties like physically unclonable functions (PUFs), for later identification. The information derived from the physical properties are then applied to a perceptual hash function to extract a discriminative IP content sensitive output value. In a first experimental case study several public implementations of different block ciphers, where each of which represent an IP, are involved to evaluate the performance. The power consumption of the device running the block cipher implementations was measured and compressed with fast Fourier transformation (FFT) to cancel out high frequency noise. To compare these compressed power consumption vectors the Pearson correlation coefficient was applied to decide whether IPs are similar or not. It has been shown that the implementations can be efficiently distinguished even with respect to minor code transformations.

Our contribution: Extending this formalism, we propose a perceptual hashing method that is parametrized by a key to counteract various code transformation attacks. Usually, an attacker would try to eliminate the similarities between the original IP and the illegitimate plagiarism of it he intends to use. For instance, such a plagiarism can easily be achieved by gradually transforming, respectively modifying, the IPs original code and generating the hash value until the similarity can no longer be proven. Using a keyed hashing method the attacker may still be able to perform attacks but without knowing the correct key he cannot guess the particular influence on the content sensitive hash value, and thus the attacker is not able to examine the hash value after each modification. In this work we utilize and adapt the wavelet transform in order to obtain signal features dependent on a key. For comparison reasons the signal features are first quantized and then assessed with the help of the *bit error rate*.

Organization of the paper: Section 2 introduces the concept of perceptual hashing. Section 3 provides an overview of the wavelet transform. In Section 4 we describe our hashing approach assuming meaningful attacks. Finally, Section 5 reports our experimental results before we conclude in Section 6.

2 Perceptual Hashing

Complying with the definitions in [7] a *perceptual hash function* is a probabilistic procedure $\mathcal{H} : (IP, content) \rightarrow h$ that outputs a hash value given a physically observed quantity of a device, referred to as the content, while it runs the IPs code. The main step of the hash value generation is the *feature extraction* that filters significant samples which are either invariant for similar contents or distinct for different contents. Given two hash values a *similarity detection function* $\mathcal{D} : (h, h^*) \rightarrow [0, 1]$ then makes a decision about the grade of similarity.

A perceptual hash function is probabilistic since it deals with physically measured information that is inherently subjected to measurement noise which means that two measurements of the same IP yield similar hash values but not identical ones. Therefore, the authors assess the detection performance by two properties, the threshold-bounded *perceptual robustness* and *content sensitivity*. The perceptual robustness states the probability that two IPs which are actually similar reach a similarity score larger than a predefined threshold $\tau \in [0, 1]$. Consequentially, the content sensitivity states the probability that two IPs which are actually different have a similarity score smaller than τ (*cf.* [7], 4.2).

Nevertheless, in this work we slightly redefine the properties while assuming a Gaussian distributed binary classifying system with the similarity score distribution $\mathcal{N}sim(\tau_{PeRo}, \sigma^2_{PeRo})$ when the IPs are similar and the distribution $\mathcal{N}_{dif}(\tau_{CoSe}, \sigma^2_{CoSe})$ when the IPs are different. The perceptual robustness can be interpreted as the ability to perceive similarities (complement of false rejection), while the IPs code is subjected to a set of transformation functions $\mathcal{F}_{pre}$ that preserve the content. Here, it is expressed by threshold $\tau_{PeRo} \in [0, 1]$ which is the expected similarity score value for similar IPs. The content sensitivity can be interpreted as the ability to perceive differences (complement of false acceptance), while the IPs code is subjected to a complementary set of transformation functions $\overline{\mathcal{F}}_{pre}$ that significantly change the content. It is expressed by threshold $\tau_{CoSe} \in [0, 1]$ which is the expected similarity score value for different IPs.

Hence, a theoretically optimal similarity detection function has a perceptual robustness threshold τ_{PeRo} of exactly one (the probability that similar IPs are detected as similar is one) and content sensitivity threshold τ_{CoSe} of exactly zero (the probability that different IPs are detected as similar is zero). In practice however, we obtain variances larger than zero (at least due to measurement noise), and thus we require the difference of the threshold values τ_{PeRo} and τ_{CoSe} to be large to reduce the probability of misclassification errors.

Remark 1. A perceptual hash function, particularly its similarity detection function, needs to maximize the threshold difference $\tau_d = \tau_{PeRo} - \tau_{CoSe}$ satisfying $0 \leq \tau_{CoSe} < \tau_{PeRo} \leq 1$ in order to provide a meaningful decision.

With this definition of perceptual hashing, attacks are straightforward since the attacker (who will be called the *plagiarist* in the remainder) is able to generate the hash value after each code transformation to examine its impact on the similarity detection. Given an original IP, an attack is successful if

- the original IP is transformed with a function $f \in \overline{\mathcal{F}}_{pre}$, *s.t.* it is detected as different: $\mathcal{D}[\mathcal{H}(IP, content), \mathcal{H}(f(IP, content))] \leq \tau_{CoSe}$, or
- a replicated IP is transformed with a function $f \in \overline{\mathcal{F}}_{pre}$, *s.t.* it is detected as similar: $\mathcal{D}[\mathcal{H}(IP, content), \mathcal{H}(f(IP_{rep}, content_{rep}))] \geq \tau_{PeRo}$.

Therefore, the hash value can be generated using a secret key so that the plagiarist cannot predict the hash value. The key-dependence can be used in different components of the hashing scheme (Fig. 1). The content can be randomized before feature extraction which is a kind of content scrambling. Further, the key can influence the feature extraction, *e.g.* randomizing parameters, and eventually, the hash value can be randomized, *e.g.* a key addition. However, in this work we concentrate on a key-dependent feature extraction. An improved perceptual robustness is a consequent inherent property of a key-dependent perceptual hash function since intentionally malicious code transformations should be suppressed in the hash value if the correct secret key is unknown. This raises another property, namely that the same content using two distinct secret keys should lead to a significantly different hash value, thus $\mathcal{D}[\mathcal{H}_{k_1}(IP, content), \mathcal{H}_{k_2}(IP, content)] \leq \tau_{CoSe}$ for $k_1 \neq k_2$.

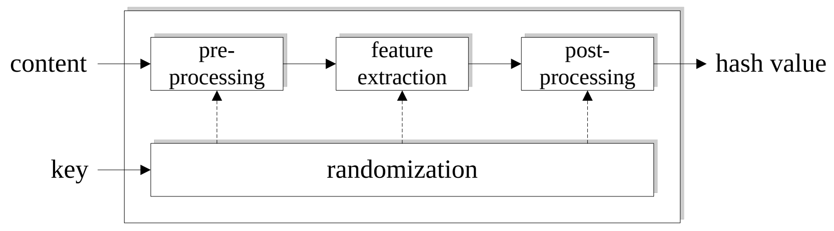

Fig. 1. The perceptual hashing scheme basically consists of three mentionable steps where each of which can be used with a key-dependent randomization. Generated hash values are then provided to a similarity detection function.

3 The Wavelet Transform

The wavelet transform applies a sliding window in form of a modulated window function that contains the frequency information. This function is called a wavelet denoted with $\Psi(t)$. Generally speaking, it is a damped oscillation that has compact support. The support, denoted with p, is half the time-width where the amplitude is non-negligible distinct from zero, and thus $2 \cdot p$ is the meaningful time-width of a wavelet. Various wavelets have been presented including *Haar*, *Gauss*, *Morlet*, *Mexican Hat*, and so forth (*cf.* Fig. 2). Each has its own base frequency f_0 and support p. In order to make wavelets capable to handle signals that have a time-width greater than $2 \cdot p$ and frequencies distinct from f_0, wavelets are accordingly scaled and shifted, such that $\Psi_{a,b}(t) = 1/\sqrt{a} \cdot \Psi(\frac{t-b}{a})$ with $a \in \mathbb{R}^+$ and $b \in \mathbb{R}$. The support is then given by $a \cdot p$, the wavelet is shifted by b along the time axis, and the frequency changes to $f_a = f_0/a$. This leads to the continuous wavelet transform

$$W(x, a, b) = \int_{-\infty}^{\infty} x(t)\Psi_{a,b}(t)dt = \langle x, \Psi_{a,b} \rangle \,, \quad (1)$$

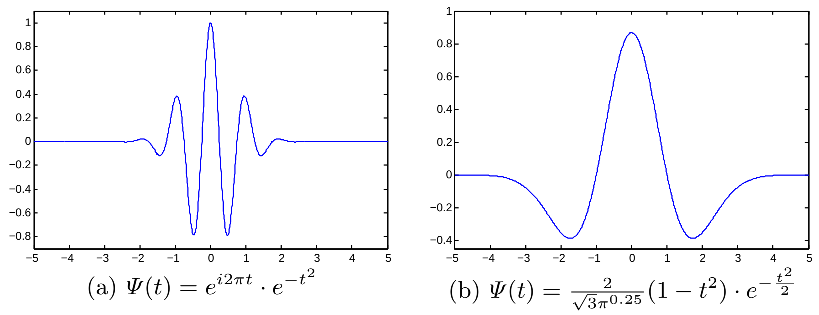

Fig. 2. (a) shows the Morlet wavelet with support $p = 4$ and (b) the Mexican Hat wavelet with support $p = 5$.

given a signal $x(t)$, where $W(x, a, b)$ is the wavelet coefficient of x that states how well the wavelet fits the overlapping fraction of $x(t)$ with $0 \leq W(x, a, b) \leq \max(|x|^2, |\Psi_{a,b}|^2)$. Numerous pairs (a, b) ensure that $x(t)$ is appropriately covered (Fig. 3), so that $x(t)$ can be represented by the wavelets. However, in practice it

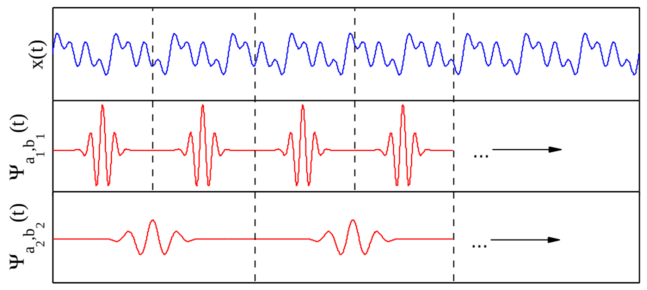

Fig. 3. The wavelet transform is carried out with different scaled versions of a wavelet (Morlet) which are successively shifted from left to right along signal x to produce the coefficients $W(x, a, b)$, and hence capture the entire information within x. Exemplarily, the wavelet in the middle has a small a and covers high frequencies of signal x, whereas the lower wavelet has a larger a to cover low frequencies of x.

is only possible to chose a and b from a finite set provided that the reconstruction of $x(t)$ is possible. For instance, the discrete dyadic wavelet transform can afford this by using $a = 2^m$ and $b = n \cdot 2^m$ for $m, n \in \mathbb{Z}$. For further details we refer to [1].

4 Keyed Side-Channel based Hashing

It is assumed that the power consumption of utilized devices is substantially related to the instruction codes, meaning that locally different instructions lead to a locally different power consumption. But this is certainly true for devices based on CMOS technology, like microcontrollers and FPGAs since the power consumption can be modeled as the sum of an instruction-dependent and data-dependent, as well as a constant and a noise component [9], thus

$$P_{total} = P_{instruction} + P_{data} + P_{constant} + P_{noise}. \qquad (2)$$

In this work we focus on microcontrollers. In the remainder we will call the physically observed content, *i.e.* the measured power consumption traces, the *power consumption shape* or short *shape* to emphasize that we are interested in the progression of the amplitude, formed by the instructions while processing data, and not in data-dependent differences.

4.1 Our Attacker Model

In our approach we assume three adversarial plagiarists.

- *Adv. plagiarist 1* possesses the ready-to-upload system code file due to theft.
- *Adv. plagiarist 2* possesses the program source code due to theft or reverse-engineering.
- *Adv. plagiarist 3* possesses the algorithm and intends unlicensed use.

Obviously, by possessing the system code file only (adv. plagiarist 1), it is very challenging to obtain a different power consumption shape by editing the file without risking errors in the program code. Whereas possessing the program code (adv. plagiarist 2) raises a few more possible attacks which are *i.* recompilation with different parameters, *ii.* inserting dummy instructions or random delays, and *iii.* permuting instructions. Further, those are also the possible attacks if the plagiarist intends unlicensed use (adv. plagiarist 3) but here she already considers them during the programming phase.

Code recompilation might be a pure trial-and-error process since the plagiarist can merely guess the influence on the power consumption instead of targeting accurate changes. Nevertheless, the attack is sufficient as long as the shape will change.

The static insertion of additional instructions or delays have an inherent effect on the power consumption but, concurrently, negatively affect the efficiency of the program code. The same applies to randomly inserted delays but further aggravated by the problem of finding a good source of randomness. In side-channel attacks one may also be confronted with random delays which can be efficiently bypassed by alignment methods [16, 13]. For side-channel hashing, several power consumption traces can be aligned with each other before generating the hash value.

Permuting instructions is only possible where the semantical functionality remains inviolated. Moreover, permuting recurring code segments that only differ in the processed data will not change the power consumption significantly since variations induced by the data slightly affect the amplitude. Although this is crucial for side-channel attacks, it is negligible for side-channel hashing due to the similar shape of the power consumption.

4.2 Our Proposal using the Wavelet Transform

The scheme relies on the short oscillation of the wavelets such that each sample of a power consumption shape is either damped or amplified depending on a key. Thus, a plagiarist does not know where changes in the shape S will significantly influence the hash value. We suggest the following procedure in order to generate a side-channel hash value $\mathcal{H}_k(IP, S) : \mathbb{R}^n \to \{0,1\}^{l \cdot r}$ where l is the amount of hash coefficients used and r the quantization resolution.

Generation of Side-channel Hash Values The wavelet transform (Eq. 1) is applied to build a key-dependent feature extraction as the randomly chosen scaling parameter a and shift parameter b function as the key whereas the output consists of l hash coefficients c_i:

1. Choose a wavelet function $\Psi_{a,b}(t)$ with support p
2. Generate key $k \leftarrow \{(a_i, b_i)\}^l$ where $a_i \in_R [a_{min}, a_{max}]$ and $b_i \in_R \{0, 1, \ldots, 2 \cdot a_i p - 1\}$
3. Compute hash coeff. $c_i = \sum_{m \cdot a_i p - b_i < n} W(S, a_i, m \cdot a_i p - b_i)$, $m \in \{1, 3, 5, \ldots\}$ in accordance with Figure 3.

Subsequently, the hash coefficients are quantized to form the final side-channel hash value $\mathcal{H}_k(IP, S) = \{Q(c_i)\}^l$ where Q is the quantizer. Two types of quantizers are imaginable. A uniform quantizer generates bins so that all coefficients are distributed among equidistant quantization steps. Contrarily, a non-uniform quantizer generates bins so that the bins are equiprobable distributed over all coefficients where the quantization point is the mean of the coefficients within a bin. If the number of bins is 2^r the final hash value contains $l \cdot r$ bits.

The scaling values a_i are bounded to reduce inherent negative effects on the performance (*cf.* [8]). Wavelets containing either too high frequencies (small a) or too low frequencies (large a) are not perceptual and give poor performance due to the dominating randomization on one side and severe information loss on the other. That means the perceptual robustness decreases with smaller scaling values but larger values decreases the content sensitivity otherwise. Obviously, the interval should be large in order to provide a sufficient number of distinct scaled wavelets. Therefore, finding a proper scaling interval is a trade-off problem and will be addressed in Section 5. To include all samples of S in each hash coefficient, multiple wavelet coefficients are added up with periodically consecutive wavelets. Hence, the wavelets can be shifted effectively by the b_i in the above given range.

Similarity Detection of Side-channel Hash Values To compare two binary hash values the *complementary bit error rate*, as suggested by other works [8, 10, 14], can serve as the similarity score which is simply

$$CBER(\mathcal{H}_k(IP, S), \mathcal{H}_k(IP^*, S^*)) = 1 - \frac{HD(Q(c_i), Q(c_i^*))}{l \cdot r} \tag{3}$$

with HD being the Hamming distance that counts the number of positions at which the bits are different. It is expected that two similar hash values reach a CBER close to 1 and two distinct hash values a CBER close to 0.5.

4.3 Practical Limitations using Wavelets

As already pointed out, we make the following important assumption that is basically considered when we demonstrate the performance of our approach in the next section.

Assumption 1 *The plagiarist is able to change parts of the shape according to the considered attacks, but not manages to change the entire shape significantly while preserving the full efficiency of the code at the same time.*

We recall that the similarity between program codes can *never* be proven if the shape has changed substantially. This is an inherent property of the perceptual hashing since otherwise arbitrary independent codes would appear similar when they are definitely not. Consequently, a plagiarist who mounts a "large impact" attack with no regards to the code efficiency will *always* be successful in the sense of making two similar program codes independent.

5 Experimental Results

Before we assess the practical performance of our approach in the presence of attacks, we need to figure out the appropriate scaling interval $[a_{min}, a_{max}]$ for the wavelet feature extraction (*cf.* Sec. 4.2) and the influence of the number of hash coefficients, respectively the number of quantizing bins.

Optimization of the Wavelet Parameters For this purpose, we initially simulated three shapes with 5000 samples each represented by an 8-bit value. The first is randomly generated, the second is a modified copy of it to obtain two similar shapes affected by considerable measurement noise. In particular, Gaussian noise with a standard deviation of $\sigma_{sim_1} = 1$ is added to each sample and to 20 consecutive samples out of 100 samples with $\sigma_{sim_2} = 5$. A third shape is again a modified copy of the first but with a Gaussian noise of $\sigma_{dif} = 15$ added to each sample. Hence, the third shape is different from the first two shapes but not entirely independent of them. This might seem somewhat arbitrary at first sight, but it is appropriate since measured power consumption shapes of two different IPs share a common constant power consumption component and vary only due to an instruction-dependent component, respectively data-dependent component, while executed on the same device (*cf.* Eq. 2). We further used the Gauss0 wavelet ($p = 5$) with 200 coefficients and 2^8 quantizing bins; values that turned out to be optimal as shown below. Figure 4 shows the performance of our approach along decreasing scaling intervals where the maximum scaling value is initially set to $a_{max} = n/(2p) = 5000/(2 \cdot 5)$ for which the wavelet is as long as the shape. We randomly generated ten thousand ($10k$) different keys from each scaling interval and computed the mean and the standard deviation of the complementary bit error rate involving the two similar shapes one and two, respectively the two different shapes one and three. As can be seen the corresponding mean values τ_{PeRo} and τ_{CoSe} remain nearly constant while decreasing the scaling interval length. In order to indicate *false acceptance* (FA) and *false*

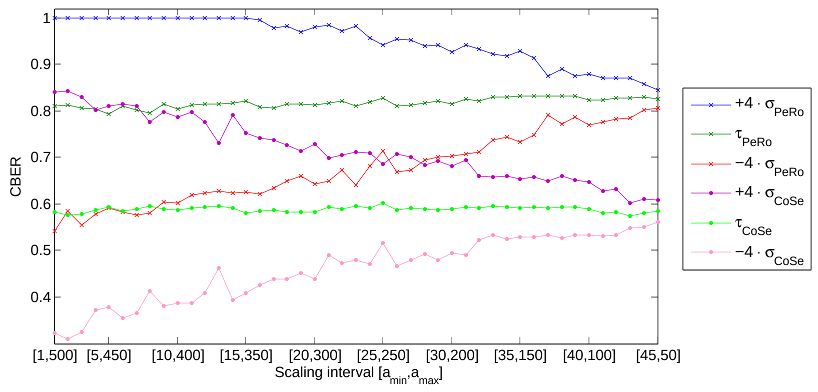

Fig. 4. The performance with simulated power consumption shapes for several scaling intervals whereas each interval was evaluated with 10 k randomly chosen keys. Further, we used the Gauss0 wavelet ($p = 5$) and a hash value that consists of $l = 200$ coefficients quantized with $r = 8$.

rejection (FR) errors, respectively, we need to investigate the impact of deviating CBER values. As we assume the CBER values to be Gaussian distributed (*cf.* Sec. 2), we apply a property of the Gaussian distribution that states that 99.99% of all possible values are within eight standard deviations around the mean. It is observable in Figure 4 that intervals larger than $[30, 200]$ (to the left) contain a higher chance of FA and FR errors since the borders given by the added standard deviation for different shapes $(+4 \cdot \sigma_{CoSe})$ and the subtracted standard deviation for similar shapes $(-4 \cdot \sigma_{PeRo})$ overlap. To include a clearance we suggest using the scaling interval $[40, 100]$ for which FA and FR errors are negligible. Moreover, this interval is sufficiently large to provide an appropriate key space. We recall that a plagiarist would need to modify the code considering each possible key, so this key space is not to be confused with key spaces from crypto systems.

Next, we investigate the influence of different numbers of quantizing bins and hash coefficients, respectively. As Figure 5 depicts, while processing similar shapes and different shapes the respective mean values decrease with increasing bins until convergence. This is because the hash coefficients are generally close to each other, and hence an appropriate number of bins is necessary to obtain meaningful results. With approximately 2^{10} bins the mean values converge but with 2^8 bins the performance is sufficient already. The results are equal for both

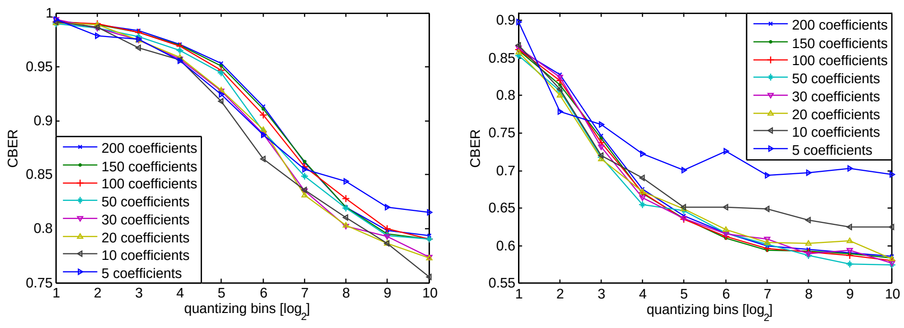

Fig. 5. Evolution of the mean values τ_{PeRo} (left) and τ_{CoSe} (right) using different numbers of quantizing bins and hash coefficients within the scaling interval $[40, 100]$.

the uniform and the non-uniform quantizer. The amount of coefficients has a less dominant effect. Actually, the performance is only poor when using less than 50 coefficients since convergence is not reached (Fig. 5). Nevertheless, at least 100 coefficients should be used for optimal performance.

Finally, we tested the Mexican hat wavelet which gives slightly less performance and the Morlet as well as the Haar wavelet which both give poor performance.

Practical Evaluation For our practical experiments we used a Microchip PIC18F2520 microcontroller [12] running at 3.68 MHz together with the Microchip MPLAB C Compiler for PIC18 of version *3.42* which also includes the Microchip MPASM assembler. The power traces were acquired with a PicoScope *5203* and a sampling rate at 125 MS/s. The measurements were done with a 1 Ω-resistor in the ground line. Further, we chose the AES to be the reference algorithm, existing in a slightly optimized assembler and a straight C implementation. Further, we choose the DES to be a true negative reference.

We measured the first round of both AES implementations with each measurement containing 1000 power traces using uniformly distributed inputs. Afterwards, the traces are compressed with peak extraction [9] and averaged to form the power consumption shapes to be evaluated. The side-channel hash values are generated using the Gauss0 wavelet, the scaling interval $[40, 100]$, 200 coefficients, and 2^8 uniform quantizing bins. Again, we computed the mean and standard deviation of the complementary bit error rate to state the performance involving $10k$ different keys.

In the easiest attack scenario the plagiarist employs the same code without modifications. This corresponds to the adversarial plagiarist 1 in our model (*cf.* Sec. 4.1). The results indicate (Tab. 1) that the measurement noise is negligible and therefore, the perceptual robustness is close to one. Contrarily, the similarity scores involving different implementations point out a good content sensitivity as well.

Table 1. Wavelet based hashing performance with unmodified and unoptimized code. The DES implementation was adapted such that a single round almost requires the same number of cycles as a single AES round. Exceeding cycles were cut off.

Implementation	CBER mean	CBER std. dev.	#samples
AES / AES Assembler	0.9646	0.0097	1235
AES / AES C	0.9857	0.0082	3503
DES / DES Assembler	0.9716	0.0086	1235
DES / DES C	0.9901	0.0077	3503
AES / DES Assembler	0.5765	0.0316	1235
AES / DES C	0.5826	0.0299	3503

Next, we consider a recompilation of the AES C implementation with different compiler parameters. The Microchip C compiler offers numerous optimization parameters from which we select all those which verifiably affect the implementation, particularly the power consumption. The selected parameters are branch optimization (BRA-OPT), banking optimization (BAN-OPT), code straightening (CS), and tail merging (TM). We refer to [11] for details. We observed that the parameters primarily lead to shortened shapes due to speed optimizations. But we also reproducibly noticed that recurring parts are more uniform which is likely caused by loop unrolling. Table 2 summarizes our results concerning recompilation.

Table 2. Wavelet based hashing performance with recompiled code. The fully optimized AES C code is compared to AES C code optimized with a single parameter or the fully unoptimized AES C code (UNOPT), respectively.

Implementation	CBER mean	CBER std. dev.	#samples
AES C BRA-OPT	0.7917	0.0113	3503
AES C BAN-OPT	0.9804	0.0083	3503
AES C CS	0.7952	0.0123	3503
AES C TM	0.7894	0.0125	3503
AES C UNOPT	0.7704	0.0134	3503

Insertion attacks aim at perturbing the alignment of the power consumption shapes with delays and additional instructions. In the pre-processing step alignment methods [16, 13] can therefore be used or methods to detect and remove dynamic insertions [15]. Thus, we concentrate on statically inserted instructions that cannot be detected with such methods in our scenario. This is because we do not know for sure whether these instructions were introduced by an attack or they are actually a part of a different unique code. For the experiments we target the Sbox-layer of our AES implementations which are both realized as a table look-up with implicit row shifting. After always two table look-ups we increasingly inserted further non-functional table look-ups (#TL). We repeat this using the no-operation instruction instead (#NOP). The performance results are given in Table 3 for the AES C implementation. The results of the assembler implementation show no difference. It can be seen that a plagiarist has to add

Table 3. Wavelet based hashing performance with stat. inserted instructions (AES).

Implementation	CBER mean	CBER std. dev.	#samples
C 1TL	0.7901	0.0113	3503
C 2TL	0.7632	0.0141	3503
C 4TL	0.7414	0.0159	3503
C 1NOP	0.7943	0.0125	3503
C 2NOP	0.7740	0.0134	3503
C 4NOP	0.7419	0.0143	3503

at least the same amount of non-functional instructions to the Sbox-layer as it contains functional instructions to significantly lower the perceptual robustness.

The last experiment is concerned with the permutation of instructions. We again target the Sbox-layer and arbitrarily reorder the bytes to be processed (REORD) on the one hand and separate the row shifting from table look-up (SEP) on the other. See Table 4 for the results. Clearly, reordering has no

Table 4. Wavelet based hashing performance with permuted instructions (AES).

Implementation	CBER mean	CBER std. dev.	#samples
C REORD	0.9732	0.0092	3503
C SEP	0.8031	0.0104	3503
Assembler REORD	0.9597	0.0098	1235
Assembler SEP	0.7731	0.0145	1235

effect since the instructions are equal. The second attack lead to slightly different results considering the implementation. The reason might be the C compiler that still rearranges the instructions, although such optimizations were disabled.

6 Conclusion

In this work we proposed wavelet based side-channel hashing which facilitates keyed perceptual hashing to detect intelligent property in embedded systems, especially considering plagiarism that is aggravated by source code modifications attacks. Our scheme fits the IP detection framework recently introduced by a previous work, but however applies a different similarity detection tool which is why the similarity score values of both approaches are not directly comparable. We tested several reasonable malicious code transformations that could be performed by an attacker without knowing the secret key. Hence, the influence of the code transformations cannot be predicted and experimental results indicate a good discriminative performance with respect to the perceptual robustness and content sensitivity of our approach in this case.

References

1. Addison, P.: The Illustrated Wavelet Transform Handbook. Taylor & Francis (2002)
2. Armknecht, F., Maes, R., Sadeghi, A.R., Standaert, F.-X., Wachsmann, C.: A Formalization of the Security Features of Physical Functions. In: Proceedings of the 2011 IEEE Symposium on Security and Privacy. pp. 397–412. IEEE Computer Society, Washington, DC (2011)
3. Becker, G., Burleson, W., Paar, C.: Side-Channel Watermarks for Embedded Software. In: IEEE NEWCAS 2011, pp. 478–481 (2011)
4. Becker, G., Strobel, D., Paar, C., Burleson, W.: Detecting Software Theft in Embedded Systems: A Side-Channel Approach. In: IEEE Transactions on Information Forensics and Security, vol. 7, pp. 1144–1154 (2012)
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. Business Software Alliance: Shadow Market. 2011 BSA Global Software Piracy Study. Business Software Alliance, Washingtion, D.C. (2012)
7. Durvaux, F., Gerard, B., Kerckhof, S., Koeune, F., Standaert, F.-X.: Intellectual Property Protection for Integrated Systems Using Soft Physical Hash Functions. In: Lee, D., Yung, M. (eds.) Information Security Applications, LNCS, vol. 7690, pp. 208–225. Springer, Heidelberg (2012)
8. Malkin, M., Venkatesan, R.: The Randlet Transform. Applications to Universal Perceptual Hashing and Image Identification. In: Allerton Conference on Communication, Control, and Computing 2004. Curran Associates, Inc. (2006)
9. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Heidelberg (2007)
10. Meixner, A., Uhl, A.: Robustness and Security of a Wavelet-Based CBIR Hashing Algorithm. In: MM&Sec 2006, pp. 140–145. ACM, New York (2006)
11. Microchip Technology Inc.: MPLAB C18 C Compiler User's Guide (2005)
12. Microchip Technology Inc.: PIC18F2420/2520/4420/4520 Data Sheet (2008)
13. Muijrers, R.A., Woudenberg, J.G., Batina, L.: RAM: Rapid Alignment Method. In: Prouff, E. (ed.) CARDIS 2011, LNCS, vol. 7079, pp. 266–282. Springer, Heidelberg (2011)
14. Nguyen, D.Q., Weng, L., Preneel, B.: Radon Transform-based Secure Image Hashing. In: De Decker, B., Lapon, J., Vincent, N., Uhl, A. (eds.) CMS 2011, LNCS, vol. 7025, pp. 186–193. Springer, Heidelberg (2011)
15. Strobel, D., Paar, C.: An Efficient Method for Eliminating Random Delays in Power Traces of Embedded Software. In: Kim, H. (ed.) ICISC 2011, LNCS, vol. 7259, pp. 48–60. Springer, Heidelberg (2011)
16. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving Differential Power Analysis by Elastic Alignment. In: Kiayias, A. (ed.) CT-RSA 2011, LNCS, vol. 6558, pp. 104–119. Springer, Heidelberg (2011)