RAM: Rapid Alignment Method

by Ruben Muijrers

supervised by Jasper van Woudenberg and Lejla Batina

Thesis 561

June 17, 2011

Differential Power Analysis is a widely used side channel attack that uses leakage in the power signal of a device to extract its key. It requires a large number of power signal measurements of the device while it is encrypting known plaintexts and uses statistics to analyze them.

For this attack to work it is important that power traces are aligned in the time domain. If this is not the case the number of traces required to successfully perform the attack increases with several orders of magnitude. One of the countermeasures against this type of attack is based on this requirement. By using an unstable clock or introducing dummy operations the measurements are misaligned.

We propose an algorithm to align these measurements based on the ideas of SIFT and U-SURF which are algorithms used for object recognition in images. The proposed algorithm consists of four main components: A detector to detect points of interest; A descriptor to generate a feature vector for those points; A matcher to match points between traces using the feature vector; And a warper that aligns the trace on these points and interpolates in between. Each of the components can easily be replaced resulting in an easy to use framework for new alignment algorithms.

We conclude that the proposed algorithm outperforms Static Alignment and SW-DPA and that performs similar to Elastic Alignment but is over 50 times faster.

Acknowledgments

Here I'd like to thank a few people who also contributed to this thesis in one way or another. On the top of my list are my supervisors Lejla Batina and Jasper van Woudenberg. Their guidance and patience was a great help getting this thesis done. Their help was especially appreciated during the last week when e-mail communication with comments, questions and suggestions intensified to several e-mails a day. Also I'd like to thank Jing Pan for filling in for Jasper when he was not available.

I'd like to thank Riscure B.V. for providing the knowledge, software and hardware needed for this thesis. Additionally I'd like to thank Riscure for providing lunch each time I dropped by, these were the most extensive lunches I've seen so far in a company (keep it up!). Also I'd like to thank the employees of Riscure for making my weekly trip to Delft (which took 3 hours one way) a rewarding one. I'd like to thank them for all the funny and interesting discussions during lunchtime.

Furthermore I'd like to thank a couple of my fellow students, namely Albert Gerritsen, Allan van Hulst, Jan de Muijnck-Hughes and Jelle Schuhmacher (not related to the famous racer). They encouraged me and were of great help when I got stuck. I'm also grateful for the daily discussions in the coffee breaks. Although most of these discussions did not make too much sense these were always a welcome intermezzo.

Finally I'd like to thank my girlfriend, Linda, for being very supportive during the writing of this thesis. Also I'd like to thank her for being one of the very few who are able to getting me to work whenever I run out of motivation.

Contents

List of Figures

List of Algorithms

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
CPA	Correlation Power Analysis
DES	Data Encryption Standard
DPA	Differential Power Analysis
DTW	Dynamic Time Warping
EA	Elastic Alignment
FT	Fourier Transform
MHz	Megahertz
POI	Point Of Interest
RAM	Rapid Alignment Method
RFID	Radio Frequency Identification
SIFT	Scale Invariant Feature Transform
SNR	Signal-to-Noise Ratio
SPA	Simple Power Analysis
STFT	Short Term Fourier Transform
SURF	Speeded Up Robust Features
SW-DPA	Sliding Window DPA
U-SURF	Upright SURF
XOR	Exclusive OR

1 Introduction

Nowadays small electronic devices such as smart phones, PDAs and smart cards are becoming increasingly popular. As a side effect of this trend more critical information is stored in these devices and therefore it is crucial that they are secure. When smart cards entered the consumer market they often had major cryptographic weaknesses in encryption algorithms. Because of the constraints on memory and computing time engineers were forced to develop there own encryption methods which were usually not nearly as well tested as the known encryption algorithms for faster platforms such as personal computers. This has changed now, due to more powerful smart cards using proven secure algorithms.

A good example that this can backfire has been shown by Garcia et al. [10]. The MIFARE classic RFID chip was developed by NXP Semiconductors (formerly Phillips Semiconductors) and is in use for over a decade now. At the moment [10] was published NXP claimed that they had sold over 1 billion MIFARE cards and there would be around 200 million MIFARE classic cards in use around the world. Due to the fact that the card had such a successful history it was marketed as *field-proven* secure. However, at the time the MIFARE classic chips came to the market there where no RFID readers available for consumers. Those few who were able to look at the internals of the chip had to sign a non-disclosure agreement. When eventually researchers were able to evaluate the encryption algorithm used in the chip, they found a major flaw which exposed the keys and thereby allowed full read and write access to the cards. Due to the many applications this discovery had quite some impact and was covered by several media. NXP still sells MIFARE classic systems but gives a warning when selling, also they no longer claim it is *field-proven* secure.

Now the algorithms used in modern chips have become secure, attackers shift focus to specific implementations. One of the new focus areas is the power consumption of a device. About a decade ago Kocher et al. proposed a method that allowed attackers to extract the secret key used for encryption operations from a small device such as a smartcard [12]. Since then many countermeasures been developed and ways to negate or reduce these countermeasures as well.

Kocher's proposed method, DPA (Differential Power Analysis), and many variants depend on the assumption that during the encryption the time intervals between the start of the encryption algorithm and each operation during the encryption remain constant between multiple executions of the algorithm (with the same input). Meaning that if it takes 1 millisecond for the algorithm to get to a certain point in the algorithm, it will take 1 millisecond to get to that same point every time the algorithm is executed. In standard algorithms this is usually the case. However by adding bogus operations at random points in time or using an unstable clock it is possible to defend against DPA

attacks. This thesis proposes a method to negate this countermeasure. By finding easy to recognize points in the power signal of a device and aligning those points the power signal is altered so it is susceptible to a DPA attack again.

Other algorithms to do this are Static Alignment and Elastic Alignment. Although Static Alignment can be used for this purpose it does not perform very well. This is due to the fact that it can only handle differences in the starting point of the encryption algorithm but not differences in the timing of the individual operations. By shifting the recorded power signals in such a way that they align around the point of attack it improves susceptibility to DPA attacks not substantially.

Elastic Alignment performs quite well. On the used trace set it was capable of raising the chance of success from 0% to 95%. It works by aligning samples with each other that have the highest correlation. It first does this with the minimal resolution of only two samples. It then iteratively increases the resolution of the recorded power signal and aligns again at each step until the full resolution is reached. Although this algorithm performs well in a lot of cases it has the major drawback that it is slow. Although it has a computational complexity of $O(c \cdot n)$ with n being the number of samples, the constant factor c is quite high. This causes alignment to take several days on large trace sets.

The proposed algorithm is designed to deal with the calculation time issues of Elastic Alignment and it should execute much faster while still producing reasonable results. It aims to bring down the calculation time from days to hours. The algorithm is inspired by U-SURF [3] which, given a reference picture, is used to recognize pictures of the same scene/object taking into account differences in angle and light. The proposed algorithm uses several techniques used in U-SURF, among most notably the use of block wavelets for detection and identifying of specific points in the recorded power signal. The main advantage of block wavelets is that they can be applied in $O(1)$ which makes the algorithm very fast.

The question that this thesis attempts to answer is as follows:

> Can we design an alignment algorithm that is fast and yet provides reasonable results?

Where 'fast' means that the computation of an typical trace set should complete in hours as opposed to Elastic Alignment were it takes days. With 'reasonable' is meant that the resulting aligned traces should be usable to perform a DPA-style attack. To answer the question ideas from the SIFT and U-SURF algorithms are used and adapted to fit this application.

We translate the ideas of SIFT and U-SURF from the 2 dimensions (images) to one dimension (power traces). Our one dimensional version needs to be a lot more robust against noise since power traces contain a lot more noise than the images used for object recognition. Finally we add a step to the algorithm that takes care of the actual alignment of the traces.

This thesis is organized as follows. Chapter 2 describes the basics of side channel attacks such as DPA and several countermeasures. Chapter 3 covers the algorithms currently available, Sliding Window DPA, Static Alignment and Elastic Alignment. Chapter

4 gives some background about signal processing and substantiate the choice for the use of wavelets for this application. Chapter 5 covers the fundamentals of SIFT and U-SURF on which the proposed algorithm is based. The proposed algorithm is described into detail in Chapter 6 and Chapter 7 discusses the setup for our experiments. Last but not least, in Chapter 8 the conclusion and further research is covered.

2 CPA and Alignment

2.1 Power Analysis

Attacking an electronic device by analyzing the algorithm mathematically is considered attacking over the *main channel*. One tries to find a mathematical or logical flaw in the algorithm (or protocol) and tries to exploit that. If this succeeds, every device that implements the same algorithm is vulnerable to the same attack. There are other attacks smart cards and small electronic devices have weaknesses against. Every computer chip emits heat, sound and light during a calculation. Also the electromagnetic field and the power consumption change during a calculation. *Side channel analysis* makes use of the information that is exposed through these side products of the computation. The data that is measured from such a side channel can be combined with a model that gives the relation of the side channel leakage and the internal activity of the chip. With an accurate enough model and an unprotected device it is possible to find out what the device is computing. With side channel analysis the focus is not so much on the algorithm itself but more on a specific implementation. If one succeeds in breaking the security of a device through side channel analysis it does not mean that every other device that uses the same algorithm is also vulnerable to this same attack. Different chips can have different implementations of the same algorithm and can have different countermeasures to prevent side analysis. The type of side channel attack that inspired this thesis is called power analysis, which was invented by Kocher et al. a decade ago [12]. Power analysis focuses on the power usage of a device and tries to map this to the internal calculations.

One of the most notable power analysis attacks is called *Differential Power Analysis*. This works roughly as follows.

1. First the timing of the leakage is determined. This should be a position in the algorithm where some bits of the key k are combined with a known non-constant value v. This is normally a piece of the plaintext when attacking the beginning of the encryption algorithm or a piece of the cyphertext when attacking at the end. We denote the piece of the key and the known value combined as $f(v, k)$.

2. Now we feed the device a large number of random plaintexts and let it encrypt these. How many are necessary depends on the countermeasures present in the device and the amount of power leakage and measurement noise. In practice the number of plaintexts needed varies from thousands to millions.

3. For each of the possible values of k the bit $f(v, k)$ is predicted. It is important that the number of possible values of k is limited. For every possible value of k the trace set is split into two groups. One, where the predicted bit $f(v, k)$ is 0

and the other where the bit is 1. If the guess for k was correct the majority of traces in each group group are traces where the actual bit matches the predicted bit $f(v, k)$. If the guess for k was incorrect the traces are split randomly and the two groups contain traces more or less 50% of traces where the actual bit matches the predicted bit and 50% of the traces where the actual bit does not match the predicted bit.

4. The traces from each group are averaged. Both the averaged traces are subtracted from each other. Since the traces were split on the basis of the predicted value of $f(v, k)$ all sample points not representing this bit average to more or less the same value in both of the groups. Subtracting those results in a value near 0. The same happens for the sample point where $f(v, k)$ is processed if k was wrong. If k was right this sample point averages to power consumption value that is needed to process the value of the predicted bit for each group. Subtracting these two averages does not yield a value near 0. So ideally the result of these four steps is a trace where every sample point is 0 and a clear peak (positive or negative) at the position where $f(v, k)$ was processed.

The above process is repeated for every key bit. The statistical test used to determine the relationship between the hypothetical power consumption and the measured power consumption is called a *side channel distinguisher*. A popular variant of DPA is– *Correlation Power Analysis* (CPA)*[5]*. CPA does not use the difference in means as the side channel distinguisher but uses Pearson correlation. It takes longer to compute but it is more sensitive for the linear dependencies of power in data and thus provides better results in general.

Simple Power Analysis (SPA) *[12]* requires the attacker to analyze the power signal manually by finding patterns in the signal. This is, despite the name, far from simple and requires quite some experience on the attacker side. However, typically only a few traces are needed to perform SPA. Therefore it might be the only option to go with when not many traces are available.

Since the DPA and CPA rely on respectively averaging and correlating traces, it is important that the traces are aligned in the time domain. A specific calculation step of the chip-under-attack should occur at the same position in the time domain for all the measured traces. If this is not the case, combining the traces averages out the measurements and lose the information stored in there. Some of the countermeasures against DPA-style attacks are based on this principle and aim to create variances in the time domain.

2.2 Countermeasures

There is a strong push from certification bodies such as Common Criteria and EMVco on manufacturers to keep up with the latest techniques, while balancing these with their economical constraints. They therefore implement a range of countermeasures to reduce the possibilities of DPA-style attacks. The main goal of the countermeasures against

DPA or CPA is to minimize or hide the dependencies between the power consumption and the intermediate values of the encryption algorithm used. Several countermeasures have already been developed and successfully put in use against DPA or CPA attacks. Although this research is focused on one specific way to counter one specific class of countermeasures (time domain hiding, including random delays and unstable clocks) it is important to know that this is only one of the few types of countermeasures. Most countermeasures fall in one of two categories which are explained in the remainder of this section.

2.2.1 Masking

With masking, one changes the encryption algorithm slightly so that the intermediate values are randomized and do not correspond to the standard but the final result remains the same. Because many attacks depend on the prediction of intermediate values this is an effective way of preventing such attacks. A typical way to do masking for AES (Advanced Encryption Standard) is to XOR all intermediate value with a certain mask. This way at every point in the algorithm the intermediate values are different from what is defined in the standard. Of course one has to make sure that the final result is the same. Before the start of the encryption one picks a random value m, this is the mask. Now the algorithm, including the S-boxes, are changed in such a way that every intermediate result v is replaced by its masked variant $v_m = v \oplus m$. At the end of the encryption algorithm the mask is removed to get the correct result. Because now only v_m is processed there is no dependency between v and the power usage as long as m remains unknown to the attacker. To negate this countermeasure one can use second order power attacks [12, 15] which attack multiple points in the algorithm at the same time and exploit the dependencies within the algorithm. Using more than one mask increases the protection level but decreases the performance which can be unacceptable given the timing constraints of the protocol used.

2.2.2 Hiding

The second category, hiding, is focused on decreasing the so-called signal-to-noise ratio (SNR) [14] which is defined as follows:

$$SNR = \frac{Var(P_{exp})}{Var(P_{noise})}$$

Where P_{exp} is the exploitable component of the power consumption and P_{noise} is the noise component. The lower the SNR is the more traces are needed to successfully perform the side channel attack. Put simply: the more noise there is, the harder it is to get something useful out of the signal. There are multiple ways of decreasing this ratio.

Amplitude related hiding

Possibly the most straightforward way to do this is to increase the noise. This is typically done by adding a parallel calculation (hardware or software) or adding extra hardware

components that generate random noise. This principle shows one of the advantages of a hardware implementation opposed to a software implementation. In hardware implementations, where the algorithm is implemented by linking logical gates, the algorithms are executed much faster and often there exists some parallelism which introduces noise as opposed to software implementations where a general purpose processor needs to execute assembler code. When parts of the algorithm are executed in parallel the power signal does not map to a single point in the computation anymore. Now the power signal consists of the sum of multiple computations. This makes DPA-style attacks much more difficult and increases the number of traces needed considerably.

One can of course also reduce the usable signal. By ensuring that every operation uses the same amount of power it reduces the amount of leakage. An example of this is shown below:

```
1: if b then
2:   a = f()
3: else
4:   a = 1
```

The problem with the above code is that calling a function typically (in line 2) uses more power than assigning a constant (in line 4). This could show up in the power signal. To solve this issue we introduce a dummy variable d which can be used in the following way:

```
1: if b then
2:   d = 1
3:   a = f()
4: else
5:   a = 1
6:   d = f()
```

Now the branches of the if-statement do the same calculation except they assign the results to a different address. They both use the same amount of power and are therefore much harder to distinguish when looking at the power signal. A disadvantage of this type of countermeasures is that adding bogus instructions usually comes with a significant performance penalty.

There are several logic styles which aim to have a constant power usage throughout the program and keep any information hidden from view. A few examples of such styles are Sense Amplifier Based Logic [20], Wave Dynamic Differential Logic [21] and Dual-rail Transition Logic [17].

Time related hiding

Another set of countermeasures in this category is based on making the computations happen at a non-consistent time during the execution of the encryption algorithm. Although this is not a distinct category, technically it falls under hiding, it is the most important set of countermeasures for this thesis. To do this one can introduce dummy instructions in the algorithm or process interrupts at randomized intervals. This causes

changes in the time dimension. Both DPA and CPA combine traces to enhance the relation between the power trace and the computation in the device. When this countermeasure is applied it becomes much harder to perform the attack successfully. Note that it is still possible to perform the attack but the number of traces needed increases with several orders of magnitude to average out the effects of the shifted time dimension. A disadvantage is that there is a performance penalty due to the fact that the algorithm contains dummy instructions and that the values for the randomized intervals need to be computed.

Changes in the time dimension can also be achieved by using an unstable clock. This causes operations to have a non-constant time, even if they are the same [14].

This thesis focuses on this last type of countermeasures. By aligning the traces we reduce the changes in the time dimension so fewer traces are needed to successfully attack the implementation. This is particular interesting due to the fact that modern cards usually have an unstable clock, added noise and added dummy instructions as countermeasures.

3 Dealing with time shifted traces

3.1 Sliding window DPA

This method was proposed by Clavier et al. in 2000 [7]. Sliding window DPA (or SW-DPA) is specifically designed to counter *random process interrupts* (RPI). When RPI are used as a countermeasure the position of the leakage that is exploited by DPA can shift a few clock cycles. SW-DPA is based on this assumption, Clavier et al. describes it as follows.

> "Suppose the spike on the differential trace should be seen after n cycles. If RPIs occurred, a spike appears after $n + C_n$ cycles, where the delay $C_n = \sum_{i=1}^{n} c_i$, c_i being the i-th cycle, with $c_i = 1$ if an RPI occurred and $c_i = 0$ if not."[7]

When traces are averaged during the DPA the resulting spike follows a Gaussian distribution. The top of this distribution is usually to low to be useful in a DPA attack. Clavier et al. proposes to reconstruct this spike by averaging consecutive clock cycles. This way, when looking at multiple traces, the spike information that was spread over several clock cycles is now combined and put back in one place.

In practice this means that each clock cycle in a power trace is replaced by the average of itself and a number of previous clock cycles. The two main parameters of this method are the number of cycles to average (the window size) and the number of samples per one clock cycle. The correct number of cycles to average depends on how many RPI have occurred before the spike. If the number is way off, the samples that contain spike information are averaged with other samples reducing the height of the spike again. There is no way of exactly knowing how many RPI occur on average before the spike so several values have to be tried. The number of samples per clock cycle depends on the settings during acquisition of the power traces. This is known by the attacker.

In [22] it was shown that SW-DPA performs fairly well. However when an unstable clock is used the performance drops drastically. This is not surprising since the algorithm assumes a stable clock of which the frequency is set in parameters of the algorithm.

3.2 Static Alignment

This algorithm is proposed by Mangard et al. in 2007 [14]. The basic idea is as follows: the attacker selects an fragment in the reference trace close to the area where the attack takes place. Then the algorithm aims to find this same fragment in the other traces and shifts the other trace so that the reference fragments are aligned. The attacker should pick

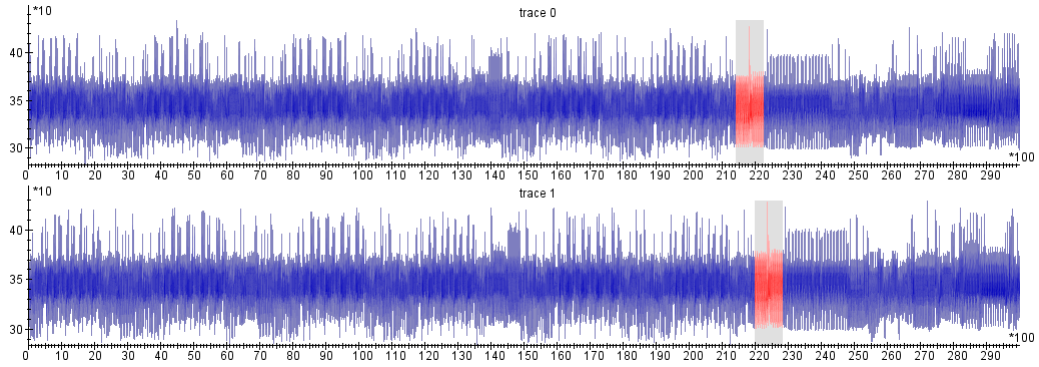

Figure 3.2.1: Static Alignment example. The two unaligned traces have the fragments selected that will be used for Static Align

a fragment with a unique pattern, since it reduces confusion of the algorithm. Although this does not fully counter an unstable clock or random delays it often does reduce the number of traces needed to successfully perform a DPA attack. This is due to the fact that aligning at a certain point in a trace reduces the noise and thereby increases the SNR around that point.

The Static Alignment implementation used for this thesis uses normalized cross-correlation to identify the fragment where the traces should be aligned. The fragment with the highest cross-correlation with the reference fragment is selected as the point where to align. Usually this fragment can be found near the position where the reference is found in the reference trace to use this as heuristic the attacker can specify a maximum shift so the algorithm only has to search within that window. If the best cross-correlation found is below a threshold set by the attacker the entire trace is excluded from the result. In figure 3.2.1 two traces can be seen. Trace 0 is the reference trace here where the attacker selects a fragment for the algorithm to align on. Trace 1 is the only target trace here, the selection here shows the same fragment as in the reference trace.

3.3 Elastic Alignment

Elastic Alignment by van Woudenberg et al. in 2011 [22] is conceptually more difficult than Static Alignment. It aims to stretch and shrink the trace in such a way that the time dimension is fully restored and the unstable clock and/or random delays are fully negated. It does this by calculating the distance (absolute difference) between every sample of the target trace with every sample in the target trace. This results in a matrix T of size $N \cdot N$. The goal is to find a path from the lower left corner (the start of both traces) to the upper right corner (the end of both traces). This path is called the *warppath*, it is used to warp the target trace so is aligned with the reference trace. Using dynamic programming techniques the path is selected where the sum of all the distances is the lowest. The cells in the matrix that are selected represent which sample points of

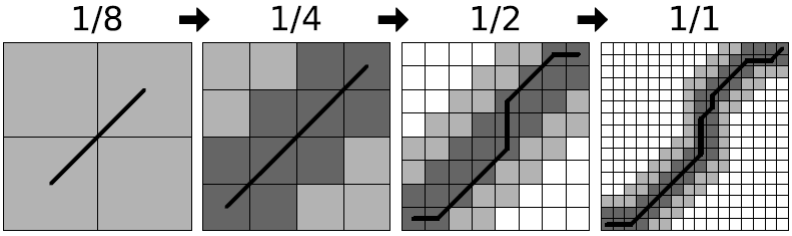

Figure 3.3.1: The FastDTW process with four iterations. Every iteration the resolution for both traces is doubled. Figure by [22].

the target trace should be aligned with which sample points of the reference trace. A hard constraint is that every sample point k that comes before sample point p the after being aligned should still come before p and every sample point q that comes after p should also come after p when the traces are aligned. In other words: one can only stretch and shrink the trace but one is not allowed to fold the trace. This constraint is based on the fact that typically for encryption algorithms it is important that the computations happen in a certain order. Lifting this constrained would assume that the device under attack could change the order in which calculations are done. Although it is possible that this happens, it is not common and it increases the search space exponentially which makes it much harder to traverse.

The algorithm described above takes $O(n^2)$ to compute where n is the length of the traces. This is problematic given the fact that some trace sets have a few million samples per trace. To overcome this problem Elastic Alignment makes use of FastDTW proposed by Salvador et al. in 2004 [19]. The idea here is to reduce the number of samples that are related to each other by starting the calculation on a low resolution of the trace and iteratively increasing the resolution. Each iteration gives a hint on where the lowest distances can be found for the next iteration. This way the number of sample points being related can be greatly reduced.

This process is shown in figure 3.3.1. Both axes represent the traces at different resolutions. The light gray squares represent the sample points where the algorithms look for good matches. This algorithm runs in $O(n)$. By only allocating the areas that are of interest instead of the entire table, it is possible to reduce the memory consumption from $O(n^2)$ to $O(n)$ as well.

The aim of this thesis is not to reduce the order of complexity but to reduce the constant c in $O(c \cdot n)$. Instead of relating every sample of the reference trace to every sample in the target trace the proposed algorithm only relates specific samples and interpolates everything in between. The aim is to reduce c with several orders of magnitude compared to Elastic Alignment.

4 Alignment with wavelets

The proposed algorithm does the same as Elastic Alignment but in a different way. The aim is to create an algorithm that is significantly faster than Elastic Alignment. Despite the linear complexity of Elastic Alignment, it can still take several days to align a typical trace set for modern cards. The proposed algorithm aims to reduce the running time with several orders of magnitude. Although the aligned trace quality is very important (the better the quality the less traces are needed), sacrificing some quality in exchange for speed is acceptable.

The proposed algorithm tries to find *points of interest* (POI) in the reference trace and then tries to find the same points of interest in the target trace. Matching points are used for the alignment. A point of interest can be a peak or a slope or anything else that is fast to detect and can be detected repeatedly in other traces from the same set. To match the POIs from the reference trace and the target trace for each POI a feature vector is generated based on the samples around the POI. This is feature vector is called the *description* of the POI. The algorithm aims for a unique description per POI so there won't be any confusion when matching the POIs.

There are many ways to detect POIs and create a description of their context. A good detection algorithm is fast, robust to noise and finds the same POIs in the reference trace and in the target trace. A good description algorithm is fast and creates a description that is unique for every POI in a trace but is the same for the same POI in the reference and in the target trace. For both these tasks we choose to use wavelets because they are very fast and perform well when it comes to catching the characteristics of a signal. In the remainder of we explain why wavelets are preferred over the more traditional way of analyzing a signal by using a Fourier Transform.

4.1 Fourier Transform

In the 19th century Fourier showed that any periodic function can be expressed as an infinite sum of periodic complex exponential functions [9]. In other words any periodic function can be expressed through much simpler functions. This idea was later generalized to non-periodic functions and discretised so it was suitable for fast computation. Even up to now this has been arguably the most popular tool to analyze signals. In this subsection we explain briefly what it does and what the disadvantages are in the context of this thesis.

A *Fourier transform* (FT) gives us an idea of what frequencies appear in the signal and with what strength. This can be illustrated by the example shown in figure 4.1.1. Figure 4.1.1 shows a periodic signal and the Fourier transform of that signal.

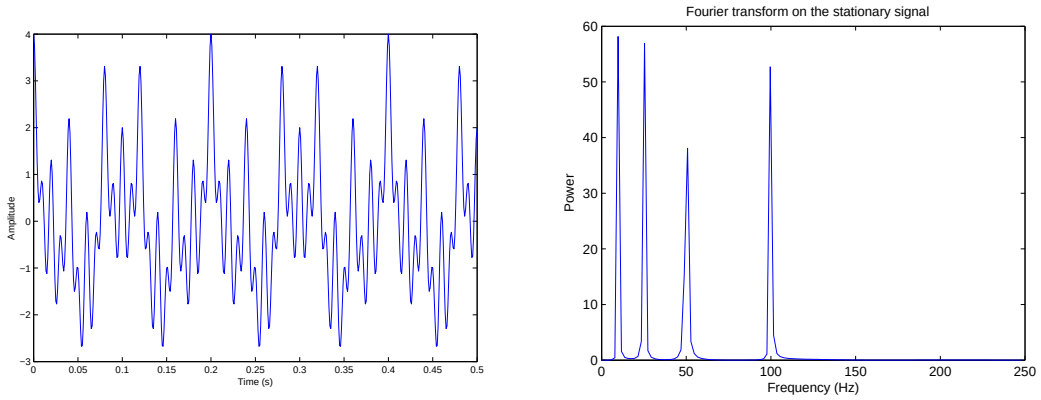

Figure 4.1.1: A stationary signal (left) consisting out of a 10Hz, 25Hz, 50Hz and a 100Hz wave and its Fourier transform (right).

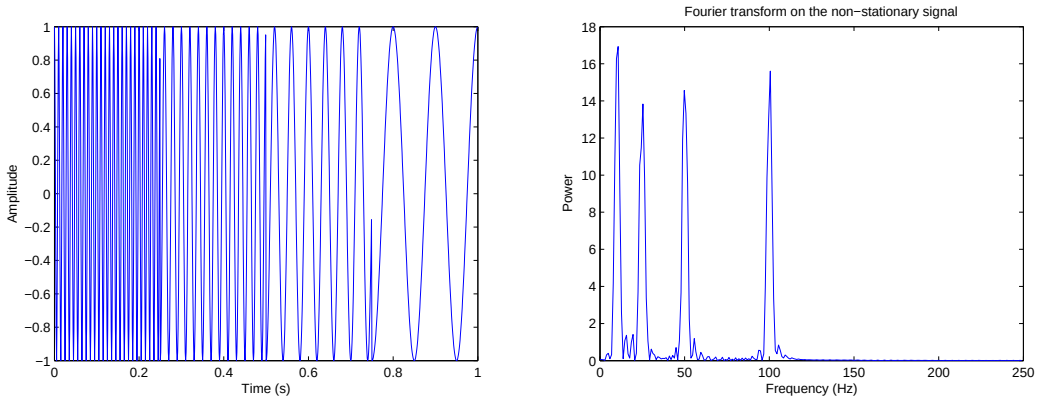

Figure 4.1.2: A non-stationary signal(left) consisting out of a 10Hz, 25Hz, 50Hz and a 100Hz wave and its Fourier transform (right).

In the Fourier transform we see four peaks, one peak at each of the frequencies the original signal consisted of. The peaks should be the same height but because the FT was computed here on a sample set instead of the actual function some peaks are higher than others. The peak heights vary depending on the size of the sample set and on the sample frequency. Nonetheless, the Fourier plot gives us a very clear description about the signal. Whereas the original signal seems somewhat chaotic to an inexperienced eye, the Fourier transform can easily be read by anyone.

Now a slightly different signal, it contains the same frequencies as before (10Hz, 25Hz, 50Hz and 100Hz) but instead of all the frequencies at the same time (as it was in the previous example), here the frequencies are consecutive so there exists only one frequency at a time. The resulting plot is shown in figure 4.1.2.

In the same figure we see that although the original signal was quite different from

the periodic signal, the FT looks very similar. In the FT we see four peaks again, one for each frequency. We also see some noise here, this is because of the transitions from one frequency to another. Note that for real signals noise is very common and cannot be used to identify or describe signals. Except for the noise the FT looks the same as for the first example. The amplitudes of the Fourier transformed signals are indistinguishable.

This is because an implicit assumption of FT. It assumes that the signal contains the same frequencies during the entire signal (from minus infinity to infinity). These types of signals are called stationary signals. The second example however, is a non-stationary signal as its frequencies change over time. Most signals that appear in real life are non-stationary. The power traces we want to analyze are never stationary since they are based on the computations in a computer chip. So Fourier transforms are not the best option for analyzing them.

Note that depending on the implementation an FT transform can also give information about the phase of a frequency (whether it is shifted to the left or the right). Although this information does help at differentiating the examples used here, one can easily construct an example (by shifting the frequencies slightly until they match in both the stationary and the non-stationary example) where an FT would fail to differentiate the signals again. For simplicity's sake we ignore the phase information here as it does not fully counter the arguments presented here.

4.2 Short Term Fourier Transform

To be able to deal with non-stationary signals *Short Term Fourier Transform* (STFT) was designed [1]. This is basically the same as FT but uses a window. Instead of transforming the entire signal STFT only transforms the signal within the window. The window has a predefined width and starts at the beginning of the signal. After each transform the window shifts over the signal until it reaches the end performing an FT at every step. This gives us information about the frequency domain and about the time domain of the signal. Figure 4.2.1 shows the results of the STFT on the two examples used earlier in this section.

Now we can clearly distinguish the signals. However, a significant problem with STFT is the width of the window. A large window gives us poor time-resolution (the larger the window the more it STFT looks like a normal FT). On the other hand if the window is small it has less resolution for frequencies. In the case of a small window the peaks of the FT widen so it becomes less clear what the actual frequency was.

The appropriate window size for STFT is application dependent. If the frequency components are well separated one can sacrifice some frequency resolution to get a better time resolution and vice versa. The window size comes down to a choice between the a good resolution in the frequency dimension and the time dimension.

Note that this problem is inherent to a physical phenomenon and is known as the Heisenberg Uncertainty Principle. Published by W. Heisenberg in 1927 this principle states that is not possible to measure certain pairs of physical properties simultaneously with arbitrary precision. The more precisely one property is measured the less precisely

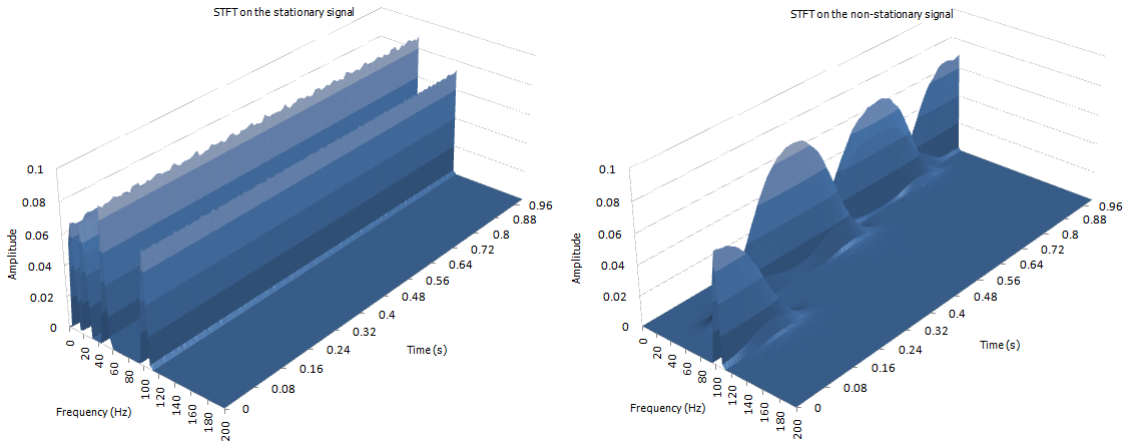

Figure 4.2.1: The results of the STFT on the stationary (left) and the non-stationary (right) signal

the other can be measured. A common example of such a pair is position and momentum of a particle. Here the frequency and the position of a wave are such a pair. It basically means that one cannot know what spectral components exist at what instances of time. One can only know the time intervals in which certain bands of frequencies exist. This is a trade-of which appears very clearly when using STFT. According Heisenberg's principle it is impossible to solve window problem of STFT perfectly, however one can make choice that works well based on the application.

4.3 Wavelet Transform

Another technique that tries to deal with the ineffectiveness in certain applications of FT is wavelet transform. The term 'wavelet' used in digital signal processing dates back several decades [18] and means 'small wave'. A wavelet can be visualized as a brief oscillation[1]. It is a non-stationary signal with an amplitude that starts out at zero, increases, and then decreases back to zero. A few examples of wavelets are shown in figure 4.3.1.

By calculating the convolution of a wavelet at a sample in the signal one gets a response that tells how well the wavelet matches the signal at that point. If this is done for every sample in the signal the series responses shows where and how well the signal matches the wavelet. Often the wavelet is applied multiple times with different scales. By doing this it is possible to describe a signal at low frequencies (large wavelets) and high frequencies (small wavelets). A wavelet can be scaled with the following formula:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})$$

[1]The source of this quote is unknown

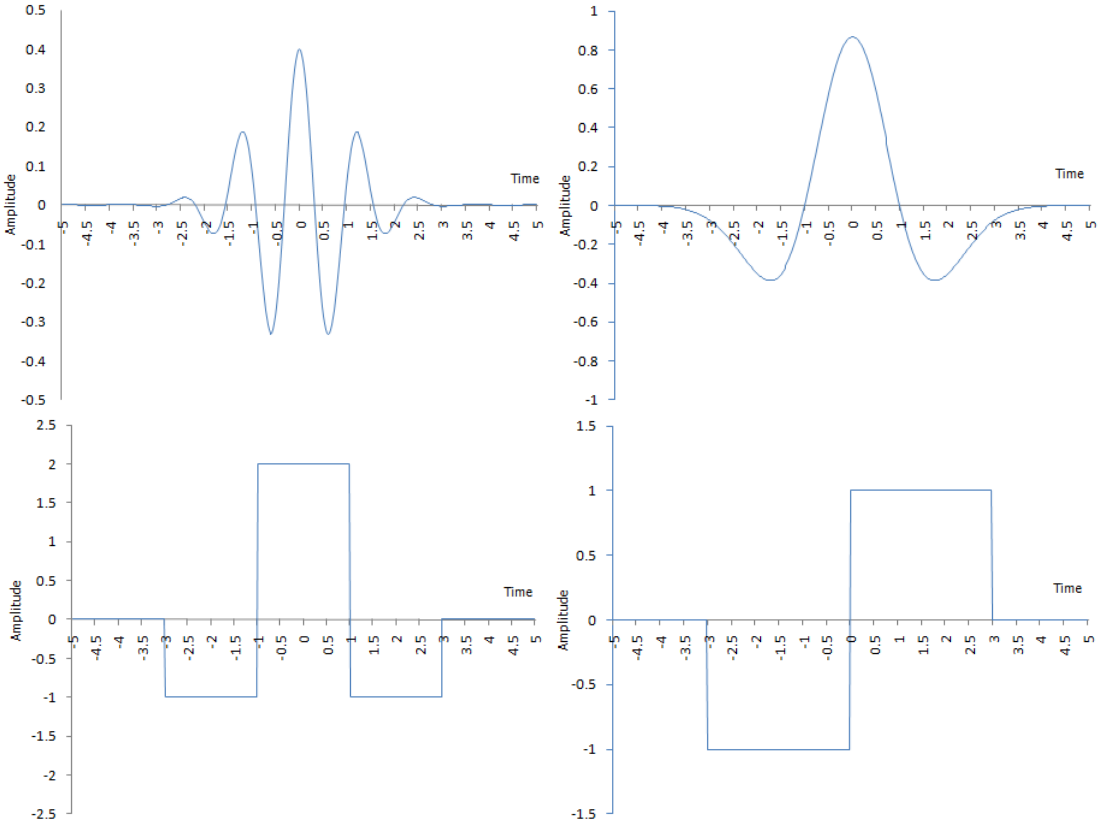

Figure 4.3.1: Examples of a Morlet wavelet (top-left), Mexicanhat wavelet (top-right), Mexicanhat block wavelet (bottom-left) and Haar wavelet (bottom-right)

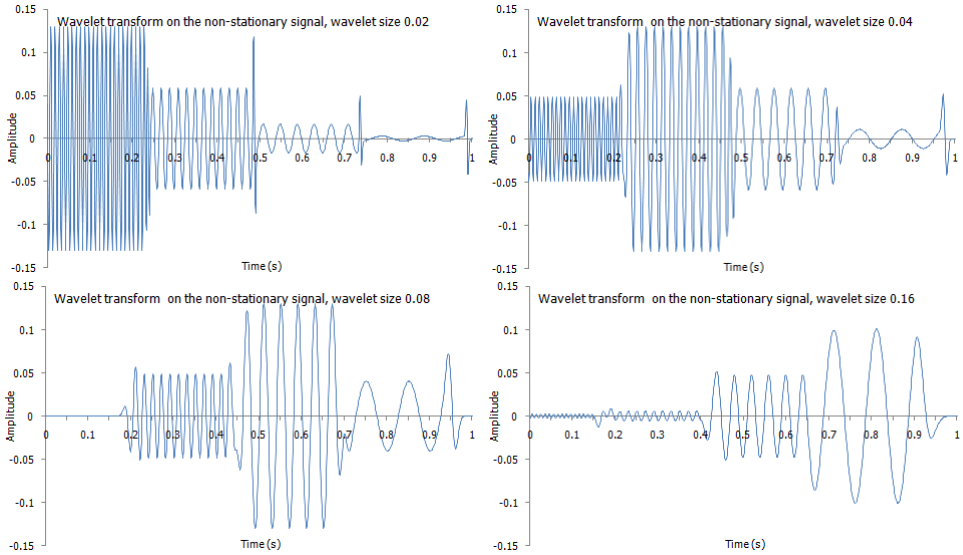

Figure 4.3.2: The response of 4 wavelets on the non-stationary signal

Where ψ is the wavelet function with (positive) scale a at offset b. A wavelet transform can mathematically be expressed in the following way:

$$WT_{\psi}\{x\}(a,b) = \int x(t)\psi_{a,b}(t)dt$$

Where $\psi_{a,b}$ is the scaled wavelet function and $x(t)$ is here the signal over time t. This simply means that you first stretch the wavelet by using a and place it on the signal that you want to analyze at b. Then pass over the samples of the signal and multiply every sample with the value of the wavelet at that position. Because most values of ψ are zero only the part of the signal around the center of the wavelet give nonzero values. The sum of those values is called the *response* for the wavelet at position b. Calculating this for every possible (or useful) b results in a wavelet transform at scale a. By doing this for several values of a we get a 2 dimensional matrix with wavelet responses. The result of a wavelet transform on the consecutive frequencies example from figure 4.1.2 is shown in figure 4.3.2.

Figure 4.3.2 shows at what time which wavelet responses are the highest in the signal. The four graphs show a clear distinction for every frequency. Note that the choice of the wavelet matters, here the Mexicanhat wavelet is used with 4 different scales. Depending on what patterns need to be found different wavelets can be used. For example when searching for a specific frequency one could use a sine (or cosine) with that frequency. To turn this into a wavelet the sine is multiplied with a Gaussian distribution curve defined by the following formula:

$$Gauss_{\sigma,\mu}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

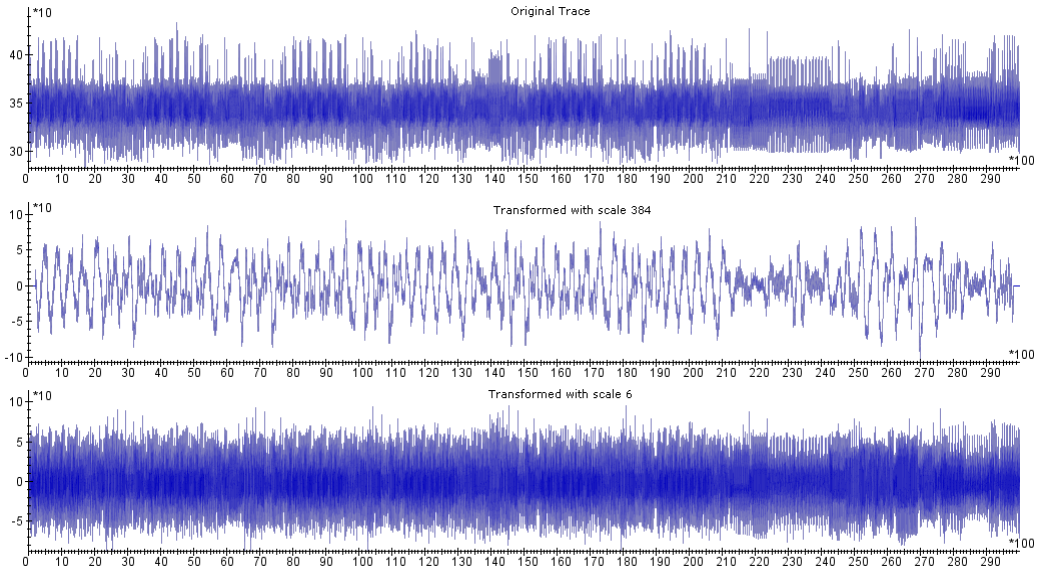

Figure 4.3.3: A power trace (top) with two wavelet transforms at different scales (middle, bottom)

Where σ is the standard deviation of the distribution this curve describes and μ is the average. One can use $\sigma$to tune the width of the bell curve and $\mu$to shift it over the x-axis. The result of the multiplication of the sine and the Gaussian is a Morlet wavelet as is shown in figure 4.3.1. When only searching for slopes in a signal one can use Haar wavelets. In the wavelet transform example used here only 4 different scales are used, depending on the application this could suffice or many more could be needed.

Figure 4.3.3 shows a wavelet transform for a power trace with wavelet scale 384 and 6. Scale 384 responds nicely to the big peaks which are visible in the original trace while scale 6 responds to every little peak in the trace.

In this thesis wavelets are used for the detection of interesting points in a trace and for generating a feature vector of such points. The advantage of wavelets for the detection of these points is obvious since wavelets give us a position of such a point whereas an FT discards the time domain. Another advantage, which is the main reason wavelets are used for the feature vector, is speed. When using block wavelets, a wavelet convolution can be calculated in $O(1)$. The details on how wavelets are used in the proposed algorithm are elaborated in chapter 6.

5 SIFT and U-SURF

The proposed algorithm is inspired by SIFT and U-SURF. The purpose of these algorithms is object recognition in images. However the techniques used in these algorithms can be translated into recognizing specific points in a power trace. The proposed finds points in two traces and aligns these points. The difference is that here we only work in 1 dimension (which reduces computational complexity) but we have to deal with a lot more noise (which can confuse the algorithm). The remainder of this section describes the original SIFT and U-SURF algorithm.

5.1 SIFT

SIFT stands for Scale Invariant Feature Transform. It is a feature generation method proposed by Lowe in 1999 [13]. The features generated are used to recognize objects in images. Object recognition algorithms have to be robust against scaling, translation, rotation and noise in the images. SIFT aims to achieve a certain robustness by using a multiple filter approach. The algorithm has three phases.

Detection phase
The algorithm starts by identifying key points in a gray scale image by using a *difference-of-Gaussian* function. The key points are minima or maxima of this function. Invariance to noise and minor distortions is achieved by blurring the image. The idea of these key points is that they will be detected in other images of this object as well. In the next phases these key points are used to recognize the object.

The difference-of-Gaussian function works as follows. First the image is blurred with a Gaussian distribution curve with $\sigma = \sqrt{2}$, this is image A. Image A is blurred again with the same Gaussian, this is image B. Image B is subtracted from A (the difference of the Gaussians) resulting in the first layer of an image pyramid. Each layer of this pyramid represents a certain scale of detection. The lowest layer is allows for the detection of very small key points while the top layer allows for detection of large key points. The next layers of the image pyramid are created by repeatedly resampling image B. This allows to detect key points of different scales. Each layer in the pyramid represents a such a scale.

Then at the first level of the pyramid a *best-of-neighbors* filter is applied. If a pixel has a higher (or lower) value than all of its 8 neighbors it passes, else it is discarded. If a pixel passes, the closest pixel at the next level in the pyramid is calculated and the same filter is applied. The level at which the calculation stops defines the scale of the key point. For a typical 512 x 512 pixel image this results in 1000 key points.

For each pixel A_{ij} in image A the gradient magnitude, M_{ij}, and the orientation, R_{ij}, are calculated using pixel differences:

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2}$$

$$R_{ij} = \text{atan2}(A_{ij} - A_{i+1,j}, A_{i,j+1} - A_{i,j})$$

Where the atan2(x, y) function is a variation on the standard arctan(x) function. The atan2 function is common in programming languages (C, C++, Java, .NET...) for calculating an angle between the x-axis and a point given the coordinates x and y. It is defined as follows:

$$\text{atan2}(x, y) = \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \pi + \arctan(\frac{y}{x}) & y \geq 0, x < 0 \\ -\pi + \arctan(\frac{y}{x}) & y < 0, x < 0 \\ \frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

Since SIFT aims to be robust against rotation the orientation of each key point is important. For each key point a canonical orientation is selected by finding a peak in a histogram. The histogram consists of 36 bins covering the 360 degrees of orientation. For every pixel around a key point with coordinates (i, j), M_{ij} is weighted against a Gaussian ($\sigma = 3 \cdot scale$, where *scale* is the scale of the key point) window and then accumulated in the bin corresponding with R_{ij}. The histogram is smoothed and then the peak is selected which defines the orientation of the specific key point.

Description Phase

The SIFT algorithm then calculates some features which bare some similarities to the inferior temporal cortex in primate vision. The area around each key point is downsampled to a 4 x 4 area and is represented with 8 *orientation planes*. The planes are small gradient matrices that each represent 45 degrees of the 360 degrees of orientation. Only the gradients (M_{ij}) from the pixels that have the an orientation (R_{ij})that is represented by the specific plane are copied to the plane. The rest of the values in the plane are filled using linear interpolation. The orientation planes are blurred to allow for shifts in positions of the gradients. These planes are the feature set for the specific key point.

Matching Phase

SIFT now looks for key point matches in the object-to-be-found-image and the search-image. If enough key points from the object-to-recognize match in the image where the object needs to be found. By using a variant of the kd-tree structure (k-dimensional tree, a data structure that, given an element, allows for fast retrieval of its nearest neighbors) called the *best-bin-first* search algorithm proposed by Beis et al. [4], SIFT is able to quickly look up and compare key points. The best-bin-first algorithm is a approximation

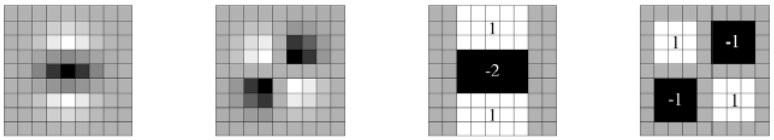

Figure 5.2.1: From left to right: Gaussian 2nd order partial derivative in y-direction and xy-direction; and the approximations of these functions; figure by [16]

algorithm. It returns the nearest neighbor for most queries and a very close neighbor otherwise.

SIFT also proposes to use a Hough transform proposed by Ballard et al. [2] to identify the orientation of the object-to-recognize. The Hough transform uses a voting scheme to decide on a canonical orientation for the object-to-recognize. By searching for key points that match this orientation SIFT and clustering these in a hash table SIFT only considers plausible key point combinations as the object-to-recognize.

5.2 U-SURF

U-SURF stands for Upright-SURF where SURF stands for Speeded Up Robust Features. U-SURF is exactly the same as SURF except that it skips a step where orientations of the *points of interest* (POI, this is same as the *key point* in SIFT) are calculated. Since, when handling power traces, rotation is not a problem U-SURF is closer related. Therefore this section covers U-SURF and not SURF. Just as SIFT U-SURF uses three phases. The algorithm is split up into three components each responsible for one phase.

The Detector

U-SURF takes the ideas of SIFT and simplifies them resulting in greatly boosted speed of the algorithm. Instead of using the blurred image pyramid from SIFT it uses a speeded up version of the Hessian-Laplace detector first proposed by Mikolajczyk et al. in 2001 [16]. This detector is based on a Hessian matrix defined by the convolutions of several Gaussian second order derivatives. POIs are identified by the response of the Hessian filter applied on the image. However instead of using a continuous Gaussian curve it uses discretised approximation as can be seen in figure 5.2.1.

The response of the Hessian filter then simplifies to:

$$Hessian(approx) = D_{xx}D_{yy} - (0.9 \cdot D_{xy})^2$$

Where D is the return value of the convolution of the Gaussian derivatives (D_{xx} is x direction, D_{yy} is y direction and D_{xy} is diagonal see figure 5.2.1). Using the approximated filters brings another advantage. Filter responses can be calculated in $O(1)$ instead of $O(n^2)$ where n is the scale of the filter. For a normal discretised filter one would need to multiply every pixel of the filter with each corresponding pixel in the image. Because the approximation filters only consist of a few different areas one can use an integral image [8] to do it in constant time.

An integral image works as follows. Every pixel in the the integral image, I_{xy}, is the sum of all the pixels to the left and above of this pixel in the original image, i_{xy}. The value of each pixel in the integral image is defined as follows:

$$I_{xy} = i_{xy} + I_{x-1,y} + I_{x,y-1} - I_{x-1,y-1}$$

Now the sum of an area in the original image can easily be calculated:

$$\sum_{A<x'\leq B;C<y'\leq D} i_{x',y'} = I_{A,C} + I_{B,D} - I_{B,C} - I_{A,D}$$

One can do the convolution of a box area in the filter by multiplying value of that area, as is shown in 5.2.1, with the sum of the corresponding area in the original image. This takes constant time regardless of the size of the filter. The Hessian filter is applied at different scales, just like SIFT a best-of-neighbor filter is applied. The resulting pixels are the POIs.

The Descriptor

For the generation of the features Haar wavelets are used. They consist of box areas as well and are therefore also fast with the use of the integral image. A window of 20 times the scale of the POI is split up into smaller 4 x 4 subregions. For every pixel in a subregion a Haar wavelet response in y-direction, d_y, and in x-direction, d_x, are calculated. These responses are weighted with a Gaussian ($\sigma = 3.3 \cdot scale$) centered at the POI. Then per subregion all the responses d_x and d_y are summed and form the first set of features for the subregion. The responses $|d_x|$ and $|d_y|$ are also summed and form the second set. At four features per subregion and 16 subregions each POI is describe by 64 features.

The Matcher

The matching works the same as with SIFT. It also uses the *best-bin-first* search algorithm proposed by Beis et al. [4] and Hough transforms proposed by Ballard et al. [2] to define a canonical orientation for the object-to-detect. In addition to this U-SURF adds an additional indexing step based on the sign of the Hessian response.

6 The Algorithm

6.1 Outline

Our proposed algorithm tries to reduce the effects of an unstable clock and dummy operations by aligning all traces in a set to a reference trace. This is a common approach which is also used by Static Alignment and Elastic Alignment. The reference trace can be the result of a calculation (such as averaging several traces) or just be one trace of the set. For simplicity's sake in this thesis we use the first trace in the set. In this thesis we refer to the trace we align to as the *reference trace* and the traces that need to be aligned as the *target traces.*

In order to limit the amount of tunable settings for attackers every parameter in the algorithm was related to properties of the trace. This way the constants used in the algorithm are not specific for one trace set causing the algorithm to be easier to use.

The algorithm discussed in this thesis is based on the ideas used in the SIFT [13] algorithm and variants on that. To be more specific the proposed algorithm is inspired by U-SURF [3]. It consists out of four components. First the Detector finds points of interest (POI) in the reference trace and the target traces. The Descriptor then describes the POIs based on their context. These descriptions then are matched against the descriptions in the reference trace by the Matcher. Finally the Warper uses the matched points to stretch and shrink the target traces to align them with the reference trace. This process starts with a large wavelet size and is recursively repeated with smaller wavelet sizes.

In the following sub-sections a detailed description is given for each of the four components. The main procedure of the algorithm is shown in algorithm 6.1. The constant Det_{mw} used in the code is the minimum wavelet scale. This is described further in the next section.

6.2 Detector

The detector's job is to detect points-of-interest (POI). A POI is nothing more than a point in the trace which can quickly and repeatedly be recognized in other traces of the same set. Later on the descriptor aims to describe the context of these points in a unique way. The entire detector procedure is shown in algorithm 6.3.

POIs are found by doing a wavelet transform. The response for wavelets of various scales is calculated for the trace. The algorithm starts with the largest wavelet scale $< Det_{mw} \cdot 2^k$ with k being a natural number. For each iteration the scale is halved until

Algorithm 6.1 The main loop for the proposed algorithm

```
01: // 'mat' the match array that is being built
02: // 'ref' is the reference trace
03: // 'tar' is the target trace
04: // 'aref' the area in the reference trace where is scanned for POIs
05: // 'aref' the area in the target trace where is scanned for POIs
06: // 'ws' is the wavelet size
07: void doAlignRecurse(match[] mat, trace ref, trace tar,
                       area aref, area atar, int ws)
08:   if(ws < Det_mw || aref.size <= 0 || atar.size <= 0)
09:     return;
10:   POI[] rp = Detector.detectWithScale(aref, ref, ws);
11:   POI[] tp = Detector.detectWithScale(atar, tar, ws);
12:   Descriptor.describe(rp);
13:   Descriptor.describe(tp);
14:   match[] m = Matcher.match(rp,tp);
15:   if(m.isEmpty())
16:     doAlignRecurse(mat, ref, tar, aref, atar, ws/2);
17:     return;

18:   doAlignRecurse(mat, ref, tar, new area(aref.start, m[0].ref),
                                    new area(atar.start, m[0].tar),
                                    ws/2);
19:   mat.add(m[0]);
20:   for(int i = 1; i < m.size(); i++)
21:     doAlignRecurse(mat, ref, tar, new area(m[i-1].ref, m[i].ref),
                                      new area(m[i-1].tar, m[i].tar),
                                      ws/2);
22:     mat.add(m[i]);
23:   doAlignRecurse(mat, ref, tar, new area(m[m.size-1].ref, aref.end),
                                    new area(m[m.size-1].tar, atar.end),
                                    ws/2);
24: // 'ref' is the reference trace
25: // 'tar' is the target trace
26: trace doAlign(trace ref, trace tar)
27:   match[] mat = new match[];     //match list
28:   doAlignRecurse(mat, ref, tar, new area(ref.start, ref.end),
                                    new area(tar.start, tar.end),
                                    getMaxWaveSize(ref.length);
29:   return Warper.warp(mat, tar);
```

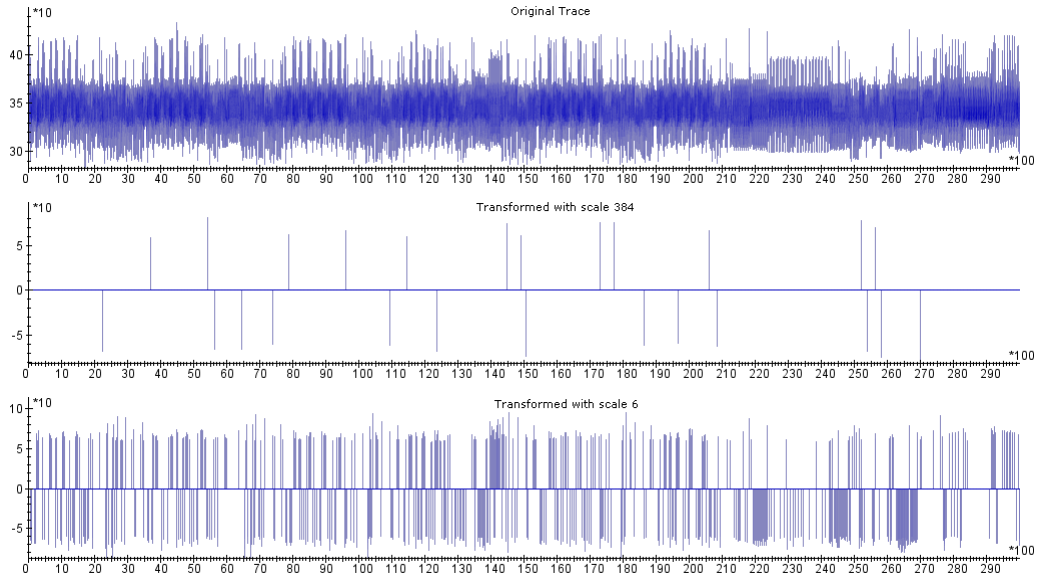

Figure 6.2.1: A power trace (top) with its points of interest for two scales (middle, bottom)

the minimum wavelet scale Det_{mw} is reached. Due to the type of wavelet used (Mexican hat) Det_{mw} should be a multiple of 3.

For performance reasons the wavelets are not applied for every sample but with a step size of 10% of the wavelet scale. A small step size slows down the algorithm where as a big step size does not detect enough POIs to work with. Early testing showed promising results for a value of 10% but given the nature of the constant other values may work fine as well.

To be able to compare the responses to a threshold the responses are normalized with respect to the wavelet scale. Samples with an absolute wavelet response less than Det_{th} times the standard deviation of the trace, are discarded.

A best-of-its-neighbors filter is than applied to the remaining samples. If the absolute wavelet response of a sample is not greater than the absolute wavelet responses of its neighbors the sample is discarded. Neighbors are here defined as every sample within 3 times the step size from the sample under evaluation. Early testing showed that at least 3 times the step size is needed here, otherwise too many POIs come through the filter and the matcher will not be able to differentiate them anymore. A greater window here could also work but has not been researched.

The samples that remain are the points of interest. Since these are the points that are used for alignment later on it is important that they are located as accurate as possible. To achieve this the remaining responses are pin pointed by searching for the highest wavelet response in the vicinity (1 times the step size, otherwise this POI would not have been the best of its neighbors and would not have passed the filter) of the point.

Figure 6.2.1 shows the result of the detector. The first image is the original trace

followed by two images that show the POIs found by the detector at different scales. As is expected, large wavelets result in much fewer POIs than small wavelets. Therefore it is easier to match the POIs from large scale wavelets to POIs from another trace. However smaller wavelets give more information on how to align the traces.

The type of wavelet used here defines on what patterns the responses will be high. Searching for slope patterns (by using Haar wavelets) and for peak patterns (by using Mexican hat block wavelets) was used. Both wavelets are shown in figure 4.3.1. The Mexican hat block wavelet outperformed the Haar wavelets in repeatability (it found the same points in similar traces).

Because the wavelet transform on a sample set comes down to multiplying every sample point of the wavelet with a sample point of the signal that is being analyzed, the complexity of applying a wavelet is usually $O(n)$ with n being the scale of the wavelet. However when using block wavelets such as the Haar wavelet or the Mexican hat block wavelet as shown in figure 4.3.1 it can be done in $O(1)$. First the trace is converted to a summed trace. Every sample is now the sum of itself and its predecessors. This is based on the idea of integral images by Crow [8]. By subtracting two samples it is now possible to obtain the sum of all the samples in between. A convolution with a wavelet is now calculated by multiplying the sum samples with the value of the block. A Haar wavelet can be applied with only 3 read operations on the trace regardless of its scale. The Mexican hat block wavelet needs only 4 read operations. Using the wavelets shown in 4.3.1 the mathematical formulas for getting a wavelet response on a sample array A at position i with scale s look as follows:

$$\begin{aligned} Haar(A,i,s) &= \frac{-(A[i]-A[i-\frac{s}{2}])+(A[i+\frac{s}{2}]-A[i])}{\sqrt{s}} \\ &= \frac{-2\cdot A[i]+A[i-\frac{s}{2}]+A[i+\frac{s}{2}]}{\sqrt{s}} \end{aligned}$$

$$\begin{aligned} Mexicanhat(A,i,s) &= \frac{-(A[i-\frac{s}{6}]-A[i-\frac{s}{2}])+2\cdot(A[i+\frac{s}{6}]-A[i-\frac{s}{6}])-(A[i+\frac{s}{2}]-A[i+\frac{s}{6}])}{\sqrt{s}} \\ &= \frac{3\cdot(A[i+\frac{s}{6}]-A[i-\frac{s}{6}])-(A[i+\frac{s}{2}]-A[i-\frac{s}{2}])}{\sqrt{s}} \end{aligned}$$

Note that the scaling with $\sqrt{s}$ is not mandatory in general but is used here to compare wavelet responses of different scales.

The code for a convolution with a Mexican hat wavelet can be seen in algorithm 6.2.

6.3 Descriptor

The descriptor aims to uniquely describe each POI by its context. It generates a feature vector which is used by the matcher to calculate the distance between two POIs. These features must be robust to noise and because there are many POIs coming from one

Algorithm 6.2 Calculates the convolution of the Mexican hat block wavelet

```
01: // 'sa'  is the sum array, sa[k] = sum_i^k sample[i]
02: // 'idx' is the position in the signal where the wavelet is applied
03: // 'ws'  is the scale of the wavelet, which should be a multiple of 3
04: // return value is the response of the wavelet
05: double convoluteMexicanHatWavelet(double sa[], int idx, int ws)
06:   int ts = ws / 3;
07:   idx    -= (ws+1) / 2; //add 1 to round upwards for odd ws
08:   if(idx < 0 || idx+ws >= a.length)
09:     return 0;
10:   return (3.0*(sa[idx+2*ts]-sa[idx+ts])-(sa[idx+ws]-sa[idx])) / sqrt(ws);
```

Algorithm 6.3 The main procedure for the Detector.

```
01: // 'samples' is the sample array, which contains all sample points
02: // 'ws'  is the scale of the wavelet, which must be a multiple of 6
03: return value is a list of POI positions
04: int[] doDetect(double samples[], int ws)
05:   double sa[] = createSumArray(samples); //sum array
06:   double ra[] = new double[sa.length];   //response array
07:   int s = max(1, ws / 10);               //stepsize
08:   int[] ret = new int[];                 //result variable
09:   for(int i = 0; i < sa.length; i += s)
10:     ra[i] = convoluteMexicanHatWavelet(sa, i, ws);
11:     if(abs(ra[i]) < Det_th*samples.sdev)
12:       ra[i] = 0;
13:   for(int i = 0; i < sa.length; i += s)
14:     if(ra[i] != 0 && isHighestOfItsNeighborPeaks(ra[i]))
15:       ret.add(pinPointPeak(sa, i));
16:   return ret;
```

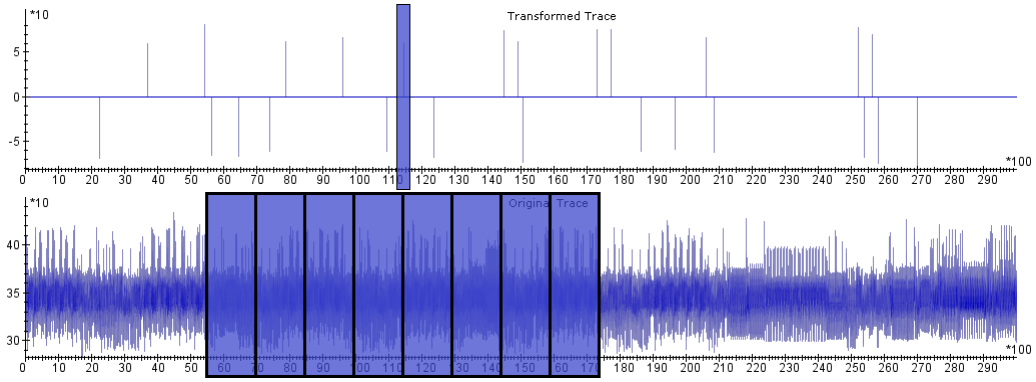

Figure 6.3.1: A selected point of interest (top) and 8 sections which are used for the features (bottom)

single trace, the features must be fast to calculate. Given these constraints it is clear that there are many possible features that can be chosen here. One of the first options that came to mind was using an FT to describe the the area around each POI. However, on our test data it quickly showed that this was not feasible for POIs with a small scale since there are not enough data points for a proper FT. The current approach is based on the approach found in U-SURF.

An area of Des_{as} times the scale around each POI is divided in several sections as can be seen in figure 6.3.1. The constant Des_{as} is intended to have the same value for different trace sets. Its value will be empirically determined. For each of the sections a few simple features are calculated. The sections are used to keep some temporal information in.

A Haar wavelet response is calculated (again using the summed trace described in the previous sub-section) with wavelet scale Des_{hw} times the scale of the POI. The wavelet responses are weighted with a Gaussian centered at the POI with σ equal to Des_{ga} times the scale of the POI. This ensures that the area close to the POI is of greater importance when comparing two feature vectors than the areas further away. To include information about the polarity of the section all the wavelet responses are summed. To include information about the intensity of the section all the absolute wavelet responses are summed. A fourth constant is used to determine the number of sections. This number is based on the feature count Des_{fc}. There are always be twice as much features as sections. How the whole procedure looks in code can be seen in algorithm 6.4. All the constants (Des_{as}, Des_{hw}, Des_{ga}, Des_{fc}) are intended to remain the same for different trace sets. They are chosen in such a way that they depend on properties of the trace.

6.3.1 The Fast Haar Descriptor

An altered descriptor has been used. The idea of this descriptor is to use less sample points to calculate the features. Instead of using every sample point in a section only a number Des_{ss} of points are selected at constant intervals from each other.

The algorithm for this descriptor is very similar to algorithm 6.4. The only differences

Algorithm 6.4 The main procedure for the Descriptor

```
01: // 'sa'  is the sum array, sa[k] = Σ_i^k sample[i]
02: // 'idx' is the position in the signal where the POI is found
03: // 'ws'  is the scale of the wavelet
04: // return value is a list of features for the specified POI
05: double[] doDescribe(double sa[], int idx, int ws)
06:   double f[] = new double[16]; //the feature vector
07:   int da = Des_as * ws; //the size of the description area
08:   int sc = Des_fc / 2;          //section count
09:   int ss = da / sc;    //section size
10:   int o  = idx - sc/2 * ss; // offset, most left sample point
11:   for(int s = 0; s < sc; s++)
12:     f[2*s]   = 0;
13:     f[2*s+1] = 0;
14:     for(int i = 0; i < ss; i++)
15:       double t = convoluteHaarWavelet(sa, o+s*ss+i, Des_hw*ws) *
                     Gauss(o+s*ss+i - idx, Des_ga);
16:       f[2*s]   += t;
17:       f[2*s+1] += abs(t);
18:   return f;
```

are that after line 10 the following line is added:

int stepsize = max(1, ws / Des_{ss});

And line 14 is replaced with:

for(int i = 0; i < ss; i+=stepsize)

6.3.2 The Super Fast Haar Descriptor

To further speed up the descriptor we assume that the Gaussian used in the descriptor does not need to be continuous. By using a discretised version of the Gaussian function we get a block pattern which means we can apply the summed array trick here. For every scale two sum arrays of the trace are calculated. One for the Haar wavelet responses and one for the absolute Haar wavelet responses. By subtracting two elements from these sum arrays it is possible to calculate the sum of the wavelet responses in between in O(1). Such a summed block is then multiplied with the average of the Gaussian curve that would be used in the normal descriptor for that block. Each block takes 2 read operations and a multiplication with the Gaussian value to calculate. The number of blocks used per section is defined by Des_{bs}. More blocks means a more accurate Gaussian upto the value where Des_{bs} equals the sample count per section. Then the Gaussian has the same accuracy as in the normal descriptor.

Algorithm 6.5 The main procedure for the Super Fast Haar Descriptor

```
01: // 'asr' is the sum array of the absolute wavelet responses
02: // 'sr'  is the sum array of the wavelet responses
03: // 'idx' is the position in the signal where the POI is found
04: // return value is a list of features for the specified POI
05: double[] doDescribe(double asr[], double sr[], int idx)
06:   double f[] = new double[16]; //the feature vector
07:   int da = Des_as * ws; //the size of the description area
08:   int sc = Des_fc / 2;          //section count
09:   int ss = da / sc;    //section size
10:   int bs = ss / Des_bs;    //block size
11:   int o  = idx - da/2; // offset, most left sample point
12:   for(int s = 0; s < sc; s++)
13:     f[2*s]   = 0;
14:     f[2*s+1] = 0;
15:     for(int i = 0; i < Des_bs; i++)
16:       f[2*s]   += sumArea(sr, o+s*ss+i*bs, o+s*ss+i*bs+bs) *
                      averageGuass(o+s*ss+i*bs, o+s*ss+i*bs+bs, Des_ga);
17:       f[2*s+1] += sumArea(asr, o+s*ss+i*bs, o+s*ss+i*bs+bs) *
                      averageGuass(o+s*ss+i*bs, o+s*ss+i*bs+bs, Des_ga);
18:   return f;
```

The code for this algorithm is shown in algorithm 6.5, it assumes that the two summed arrays for the wavelet responses are already calculated.

6.4 Matcher

The matcher creates a mapping between two sets of POIs. Because the results of the detector and the descriptor are not perfect the matcher has to be robust to the fact that not every POI appears in every trace and that descriptions may be similar.

For every POI from the reference trace a distance is calculated to every POI from the target trace. To allow comparison against a threshold the normalized Euclidean distance is used, which is a special case of the Mahalanobis distance and is defined by the following formula:

$$d(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}}$$

Where σ_i is the standard deviation of the x_i over the sample set. In our case σ_i is calculated using every possible POI with the same scale. In some cases, especially those with large scales, there are not enough POIs to give a proper estimate for σ_i but in practice this does not cause problems.

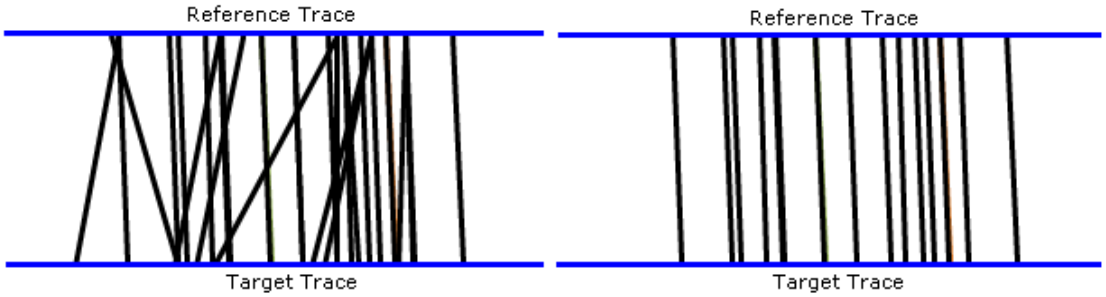

Figure 6.4.1: Two matched POI sets, with $Mat_{md} = 1000$ (left) and $Mat_{md} = 4$(right)

Each POI from the reference trace is matched with the point with the lowest distance of the target trace. If the distance is greater than Mat_{md} the match is removed. In these cases the distance is so large that one could state that the matched points are not similar at all. Early testing showed that this removed most of the mismatches as can be seen in figure 6.4.1. The traces used here were from a different trace set than the one used in chapter 7 due to the fact that this was the only trace set available at the time.

In the remaining matches there can still occur cross matches. By this we mean that sample point ref_n of the reference trace is matched against sample point tar_n of the target trace while at the same time sample ref_{n+p} is matched against sample tar_{n+q} with $p \cdot q < 0$. This cannot be allowed because it would violate the temporal behavior of the trace. This is the same constraint that is given with Elastic Alignment: the trace can be stretched and shrunk but not folded. To resolve these cross matches the confidence of a match and a penalty function is used. The confidence of a match is defined by the distance of the best match divided by the distance for the second best match. The penalty function halves the confidence of a match for every other match it crosses. The conflicting match with the lowest confidence is then removed from the set. This is repeated until all cross matches are resolved. A pseudo code implementation is shown in algorithm 6.6 and 6.7.

6.5 Warper

The Warper takes a list of matched POIs from two traces and shrinks and stretches the target trace so it is aligned with the reference trace. In the result trace the values of the samples that are included in the match list are set to the values of the target-trace. Values in between the matched points are interpolated. Various interpolation schemes have been tried such as Nearest Neighbor, Linear, Cosine, Cubic and Hermite [6]. Hermite interpolation is similar to cubic but has tension and biasing parameters. The tension controls tightens the curve at known points whereas bias twists the curve towards one of the two points. Early testing showed that differences were minimal but slightly in favor of the Cubic interpolation scheme. The other schemes have not been researched further.

Algorithm 6.6 The violator function

```
01: // 'mat' is the list of matches
02: // 'idx' is the index where the conflict was detected
03: int findViolator(match[] mat, int idx)
04:   match[] v = selectMatchesCrossedBy(mat, idx); //matches that cause problems
05:   double mc = infinity; //lowest confidence thusfar
06:   int mm  = 0
07:   for(int i = 0; i < v.length; i++)
08:     double c = v[i].confidence;
09:     c *= pow(0.5, countCrossMatches(mat, i));
10:     if(c < mc)
11:       mc = c;
12:       mm = i;
13:   return i;
```

The start and the end of the traces are not necessarily aligned. The samples before the first matched sample are interpolated with the same parameters as the samples between the first and the second matched samples. The same is done at the end of the trace. Sometimes a gap remains at the start or end of an aligned trace. There are multiple values that can be used to fill such a gap. One can use zeroes, copy parts of the reference trace or use the value of the nearest sample. The best results were achieved by using the values of the nearest sample. This is due to the fact that it does not disturb the continuity of the trace as zeroes would do and it does not add patterns that should not be there as copying from the reference trace would do. The pseudo code is shown in algorithm 6.8.

Algorithm 6.7 The main procedure for the Matcher

```
01: // 'ref' is the list of POIs found in the reference trace
02: // 'tar' is the list of POIs found in the target trace
03: match[] doMatch(POI[] ref, POI[] tar)
04:   double d1   = infinity; //shortest distance
05:   double d2   = infinity; //second shortest distance
06:   POI pm      = null;     //POI with distance d1
07:   double c    = 1;        //confidence for the match
08:   match[] ret = new match[];
09:   forall(POI p in ref)
10:     forall(POI q in tar)
11:       double d = getNormalizedEuclideanDistance(p, q);
12:       if(d < d1)
13:         d2 = d1;
14:         d1 = d;
15:         pm = q;
16:       elseif(d < d2)
17:         d2 = d;
18:   if(d2 < infinity)
19:     c -= d1 / d2;
20:   if(d1 < Mat_md)
21:     ret.add(new match(p, pm, c));

22:   //resolve conflicts
23:   for(int i = 0; i < ret.length; i++)
24:     if(crossesAnyOtherMatch(ret[i])
25:       ret.remove(findViolator(ret, i));
26:       i = 0;
27:   return ret;
```

Algorithm 6.8 The main procedure for the Warper

```
01: // 'mat' is the list of matches
02: // 't' is the trace currently being aligned
03: trace doWarp(match[] mat, trace t)
04:   trace ret = new trace();
05:   addArtificialMatchesStartEnd(mat);
06:   for(int m = 1; m < mat.length; m++)
07:     int ddest = mat[m].refpos - mat[m-1].refpos; //delta in refrence trace
08:     int dsrc  = mat[m].tarpos - mat[m-1].tarpos; //delta in target trace
09:     double r  = dsrc / ddest; // stretch/shrink factor
10:     for(int i = 0; i < ddest; i++)
11:       ret[mat[m].refpos + i] = interpolatedPoint(t, mat[m].tarpos + i*r);
12:   ret[ret.length-1] = t[t.length-1];
13:   return ret;
```

7 Experiments

The performance of the proposed algorithm was measured through several experiments. The set up for these experiments is similar to what was used by Hogenboom in [11], this set up can be seen in figure 7.0.1. The main differences are that here a different oscilloscope is used and the device-under-attack is different. The detailed hardware and software requirements as well as the general setup are explained in the remainder of this chapter.

7.1 Hardware

The hardware used for our measurements consists of several components which are listed below.

Power tracer This device is an advanced smartcard reader which can be connected to an oscilloscope. It is developed by Riscure B.V. and is controlled via the software tool Inspector. The power tracer can trigger the oscilloscope so the amount of unusable samples is reduced. It also has a very low noise power supply for the smart card which improves the signal to noise ratio.

Oscilloscope To measure the power usage of a device one needs an oscilloscope. For this project the LeCroy 104Xi was used. This is a Windows based oscilloscope that can measure up to 10 gigasamples per second and has a low noise figure.

Filters An analog filter was used to filter out frequencies that would not contribute to the CPA. In this case we used a lowpass 48Mhz filter which reduces the amplitude of the high frequencies and allows the low frequencies to pass through.

Smartcard The traces that were used for this project were acquired from a cards provided by Riscure. They contained a software implementation of Triple DES (used here with $k_1 = k_2$, so it becomes a normal DES) with countermeasures which could be turned on and off. For this project the only countermeasure that was turned on was random delays.

Computer Finally there is a computer needed to link everything together. Although the traces and calculations were performed on several computers the calculation time measurement was done on a computer with an E6750 2.66Ghz processor and 2Gb of RAM memory.

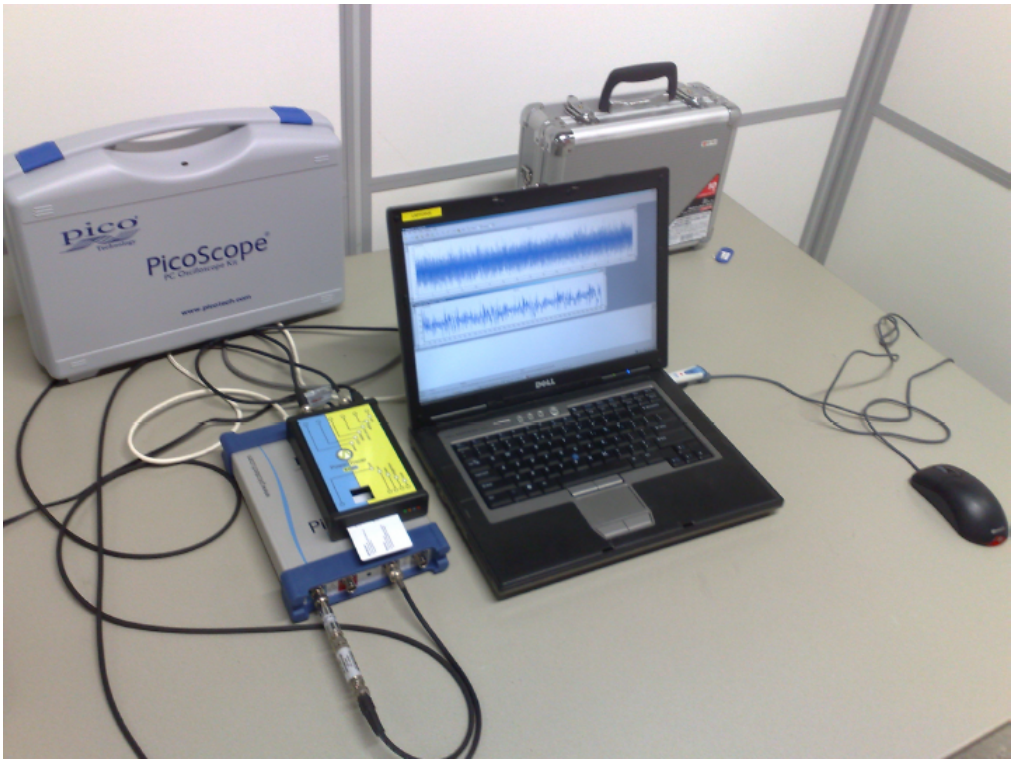

Figure 7.0.1: A measurement set up similar to the one used for this thesis. Next to the laptop is the power tracer placed on top of the oscilloscope.

7.2 Software and Settings

The main tool used for this project is an application called 'Inspector'. This is a tool developed by Riscure B.V. specifically designed to handle any operations needed for side channel analysis. When this tool is combined with the right hardware it can be used to acquire samples for analysis. These samples can be power measurements or electromagnetic measurements. Samples can also be inspected and analyzed with various modules that come with the product. The version used for this project and also currently the latest version is 4.3. Several modules where used to filter the data set before aligning. The following modules were used for this thesis:

DESAcquisition Module

This module is used in combination with the oscilloscope and powertracer to gather traces from an electronic device, in our case a smartcard. The module dialog shows various settings for the oscilloscope and the power tracer. The most notable settings for the oscilloscope are: the sample frequency, number of samples and delay.

The sample frequency specifies how many samples per second should be taken. Higher frequencies give more detail but use more disc space to store. The frequency should be set to at least the clock speed of the smartcard, 4Mhz in our case. If the sample rate is lower than the clock speed, different clock cycles end up in one sample and valuable leaked information might get lost. This setting was set to 250 megasamples per second. We chose for a much higher frequency than the clock speed to get a more accurate reading per clock cycle. The acquired traces were resampled right after acquisition by the Resample module.

The number of samples specifies how many samples are acquired per trace. It is important that this enough to cover one round of the encryption procedure. If it is too low and critical parts are missing it is impossible to do a successful CPA attack. We chose a number much greater than we anticipated we would need. Afterwards we cropped the part of the trace that was actually useful. This resulted in traces of 8248 samples long.

If a delay is set the oscilloscope waits with sampling until it receives a trigger from the powertracer. In our case the trigger was sent right after the input bytes for the encryption algorithm were transmitted.

There are also some settings for the powertracer. The most notable here is the number of traces that should be acquired. In order to generate graphs based on statistics a large number of traces is required. We set this setting to 500,000 (five hundred thousand) traces.

Resample Module

Right after the acquisition before writing them to the disc the traces were resampled in order to save disc (and memory) space. This module resamples a trace by averaging consecutive samples. We resampled to 4 megasamples per second (the clock speed of the smartcard). This resulted in a data set of 16 gigabytes. We set this module to align to a peak (clock pulse) before starting to average. This ensures that for every trace in the set the resample window is more or less positioned the same way. Otherwise one could have

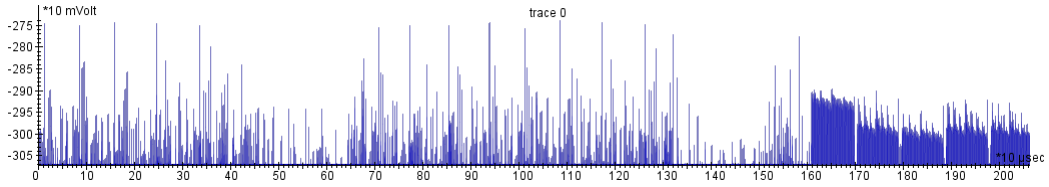

Figure 7.2.1: A typical trace from the data set, showing the last round of the encryption

the clock pulse in the beginning of the window in one trace and at the end in another. When traces are combined leakage information is spread over multiple clock cycles an become much harder to retrieve. A typical trace from the resulting set is shown in figure 7.2.1. Note that the y-axis shows negative mVolts. This is due to an offset error during acquisition and has no consequences for this thesis.

StaticAlign Module

This module was used to align the traces in the set so we could compare our proposed algorithm to Static Alignment. This module implements Static Alignment as is described in chapter 3. We selected a trace fragment of 600 samples at the end of the encryption where the module should align on. The allowed shift for this fragment was set to 250 samples.

CombAverage Module

This module pre-processes the traces for a SW-DPA attack. It averages the samples as described previously after which a normal DPA attack can be performed. The module has two settings: the number of samples per clock cycle which is set to 1 and the size of the sliding window for which we used the values 50, 100 and 200.

ElasticAlign Module

This module was used to align the traces in the set so we could compare our proposed algorithm to Elastic Alignment. This module implements Elastic Alignment as is described in chapter 3. This module has a settings to set the radius in which FastDTW searches for the optimal warppath. We set this setting to auto configure. The module then selects the optimal settings based on the first few traces. The optimal radius found was 70.

RAMAlign

This is a custom module that implements our proposed algorithm. In the settings dialog all the constants introduced in chapter 6 are listed and can be changed. As a baseline the initial values of the constants were chosen close to the values of U-SURF [3] where possible. The other baselines for the constants were based on early experimentation. Then the values where changed and used for a fraction of the data set. Based on the average Pearson correlation of the target traces with the reference trace it was decided whether increasing the value or decreasing it would show more promise. If the traces would be good aligned they would show a higher correlation than the baseline and if the traces would not be aligned properly the correlation would be lower than the baseline.

However since high correlation does not directly indicate a proper alignment in some cases it was decided that both a higher value and a lower value for the constants should be analyzed.

The analysis was done by the FirstOrderAnalysis module. Which was used to give a more founded argument for the specific values for the constants. Here are the constants listed with a short description and the values that were tested. The baseline values are in bold.

Minimum Wavelet Size (Det_{mw}) This constant specifies the stopping criterion for the detector. The detector starts with large wavelets and decreases them every iteration. If this wavelet size is reached the detector is done. Besides that lowering this value increases computation time, usually there are much more possible POIs at low wavelet sizes. This could confuse the matcher. On the other hand, lowering this value provides more precision for alignment and could increase the overall score. Tested values: 3, **6** and 12.

Response Threshold (Det_{th}) The detector only selects samples to become potentially a POI if the wavelet response at that point is greater than Det_{th} times the standard deviation of the trace. A higher value causes less confusion for the matcher but may result in too few POIs to do proper alignment. A lower value has also increases the running time of the algorithm. The matching algorithm is quadratic in the number of POIs detected so which makes this very computationally very demanding. Tested values: 2, 2.5 and **3**.

Area Size (Des_{as}) This specifies the size of the area around the POI which is used for the generation of the feature vector for this POI. The area of this POI is Des_{as} times the scale of the POI. Low values mean that POIs are related to each other based on the samples close to the POI (which could have patterns that exist multiple times in the trace). High values take samples further away into account, but could cause confusion that due to the fact that areas of POIs overlap too much. Tested values: 12, 16 and **20**.

Feature Count (Des_{fc}) This constant is related to the area size. It specifies how many features should be calculated in the specified area. More features means longer computation time but more accuracy when relating POIs. Tested values: 16, 24 and **32**.

Haar Wavelet Size (Des_{hw}) The descriptor uses Haar wavelets of size Des_{hw} times the scale of the POI to generate the feature vectors. Higher values means searching for patterns (slopes in this case) in lower signal frequencies. The wavelet becomes bigger and is less sensitive for small frequencies. Tested values: 1.5, and 2.5.

Standard Deviation Gaussian (Des_{ga}) The descriptor weighs the Haar wavelet responses with a Gaussian with an standard deviation of Des_{ga} times the scale of the POI. The Gaussian is centered at the POI. Lowering the value of this constant

means that the description is focused on samples closer to the POI. Increasing the value widens the focus window of the description. Tested values: 2.8, **3.3** and 3.8.

Maximum Distance (Mat_{md}) This constant specifies the maximum allowed distance between two matched POIs. If a match has a distance of more than Mat_{md} it is discarded. Lowering this value prevents cross matching. Lowering it too much discards usable matches. Tested values: 2.3, **2.5** and 2.7.

Due to the fact that some of the constants influence each other, changing the setting on one constant may change the optimal settings for the others, it is hard to find good values. A full grid search over all plausible combinations would take too much time so we chose to pick a base line and tune one constant at the time.

FirstOrderStatisticalAnalysis Module

This is a statistical custom module by Jasper van Woudenberg (supervisor) it is used to measure the performance of an pre-processing algorithm in combination with a CPA attack. After pre-processing is done (in our case this is the aligning) it splits up the trace set into subsets. For each subset a number of traces N_s is selected and a CPA attack is performed on these traces. The module counts the number of successful attacks S_+ and the number of unsuccessful attacks S_- and calculates the *first order success rate* R_s:

$$R_s = \frac{S_+}{S_+ + S_-}$$

R_s is then reported as a percentage of success for this specific number of traces. N_s is then increased and the process is repeated. This results in a climbing percentage of success the more traces are added. This is an indication of, given the pre-processing method, how many traces would be needed to get a certain percentage of success.

We set the start number of traces to 100 and the end to 5000 with a step size of 100. This means the data set is split into $500,000/5000 = 100$ subsets and that the module returns 50 data points per full execution. A full execution on takes about 9 hours on this data set pre-processing not included.

7.3 Tuning Results

Every execution of the FirstOrderStatisticalAnalysis module results in a series of data points. For every constant these data points were plotted in a graph as can be seen in figure 7.3.1. In every graph the baseline is plotted with a black dotted line.

When looking at the resulting graphs one of the first things that can be concluded is that the chosen baseline is not a particularly high performer. This is not a problem though, since it is here only used to tune the constants. Another important thing to note is that one should expect the graphs to be rather smooth and monotonically increasing. This is because each added trace contains leaked information that, when successfully extracted, should contribute to a higher success ratio. However the graphs are not

smooth and some start to decrease after a number of traces. This is due to the fact that if alignment for a trace fails completely it takes many successfully aligned traces to average out the errors. When more traces are used the chance of such a 'bad' trace increases and causes the success rate to drop.

There are two ways to counter this. The first is to improve the algorithm so it produces less (or none) of these bad traces. This solution is implemented by using tuned values for the constants. The second way is to build in a detection mechanism that discards bad traces. This can be implemented by using Pearson correlation after the trace is aligned. If the correlation result is below a certain threshold the trace is discarded. However, it is possible to use filter for any alignment algorithm therefore it is not considered part of the algorithm here. The correlation filter is not used during the experiments here since it would mask the performance of the actual algorithm.

The results from the graph are discussed per graph.

Minimum Wavelet Size This graph shows a clear increase in performance when using the smallest possible wavelet of scale 3.

Response Threshold Here there is clear improvement when the number of POIs increases. When too much POIs are allowed the matcher gets confused and performance drops drastically. This can be seen with the threshold value of 2.

Area Size This graph suggests that there is too much overlap in the descriptions of the POI and that the area size should be lowered.

Feature Count The results here are inconclusive. There is a very small difference in favor of less features. However, this could be noise. In early testing we were able to make a CPA attack work by increasing the number of features and since this graph does not give enough reason to lower the value of this constant remains at the baseline setting of 32.

Haar Wavelet Size Here the graphs show differences slightly in favor of the baseline setting of 2.

Gaussian Standard Deviation Here we were too careful with changing the value. The performance differences are very minimal but it is slightly in favor of a wider Gaussian curve. This suggests that the actual description is focused too much on samples close to the POI.

Maximum Distance Here we have been too careful with changing the value as well. From early testing it was very clear that a high maximum distance would cause a drop in performance. However this does not show up in the graph. Since the Threshold graph suggests that more POIs would be better for alignment it will probably not hurt to lower this setting slightly so only the best matches are used.

After we set the parameters to the new values the success ratio increased greatly as can be seen in figure 7.3.2. This performance is sufficient enough to be useful in practice.

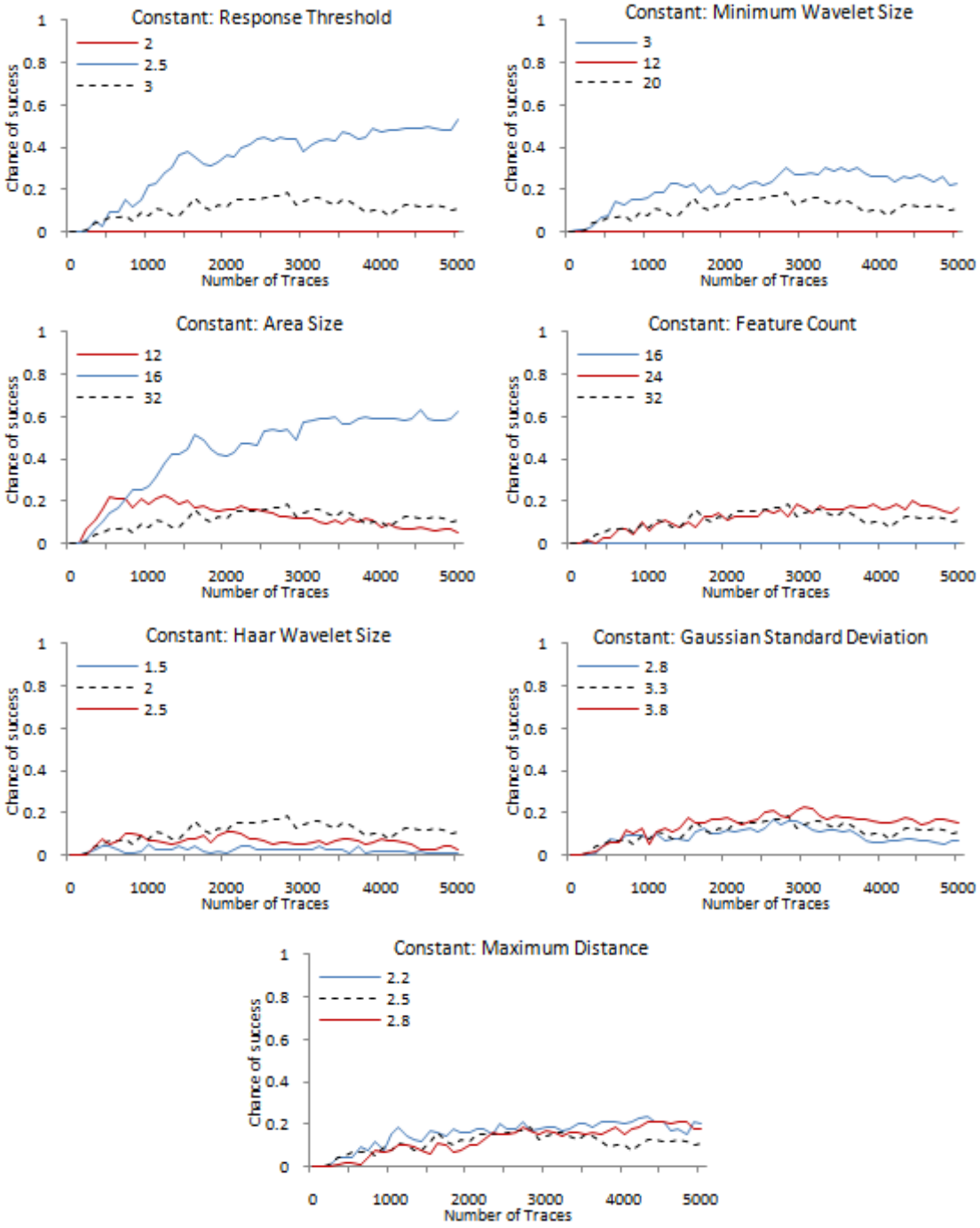

Figure 7.3.1: The results for tuning the constants

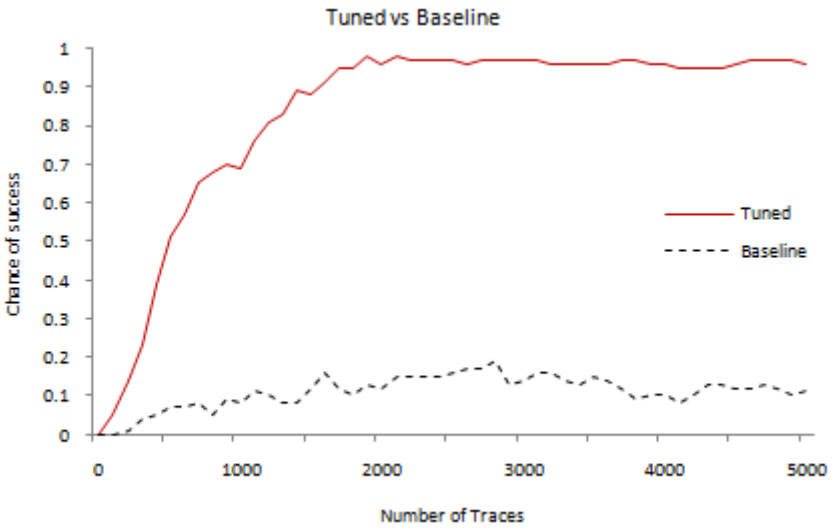

Figure 7.3.2: The performance of the tuned version

Component	Time of Calculation (%)
generateFeatures()	32
getEucledianDistance()	21.6
convoluteHaarWavelet()	20.4
getFeature()	15.8
getGaussian()	4.3
...	5.9

Table 7.1: Top 5 CPU intensive functions

Although further tuning might further increase the success ratio the focus was shifted to speed.

7.4 Speed Boosting Results

To find out where the algorithm spent the most time during execution it was passed through the Netbeans profiler. The top five most demanding functions are shown in table 7.1. The first, third and fifth function in the table are both used in the descriptor. This means that the descriptor consumes 56.7% of the CPU cycles. Our first attempt to speed up the algorithm is by using a faster descriptor.

7.4.1 The Fast Haar Descriptor

The Fast Haar descriptor introduces a new constant:

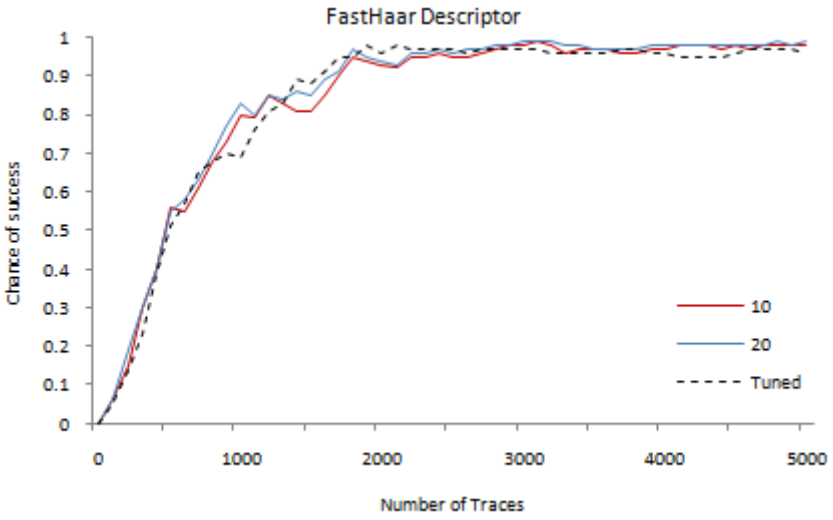

Figure 7.4.1: Results of the Fast Haar Descriptor

Samples Per Section (Des_{ss}) This is the number of samples used per section. The more are used the better the descriptor describes that section. If significantly less are used it speeds up the algorithm. Tested values: 10 and 20.

The results from this run are shown in figure 7.4.1. The number of selected samples per section does not seem to influence the success rate with the chosen values. Although this might be different on noisier trace sets, for this trace set it means that this descriptor performs well enough for use in practice. The run time on our test computer is shown in table 7.2. Although the speed increase is significant there is more to gain as is shown in the next subsection.

7.4.2 The Super Fast Haar Descriptor

The Super Fast Haar descriptor introduces a new constant:

Blocks Per Section (Des_{bs}) This is the number of blocks to calculate per sample. A higher value means that the result comes closer to that of a continuous Gaussian curve. Lower values speed up the calculation. Tested values: 1 and 2

From the results shown in figure 7.4.2 it is clear that the performance is decreased. Although the algorithm gained a great speed boost as is shown in table 7.2 the Fast Haar Descriptor may be preferred on more difficult trace sets.

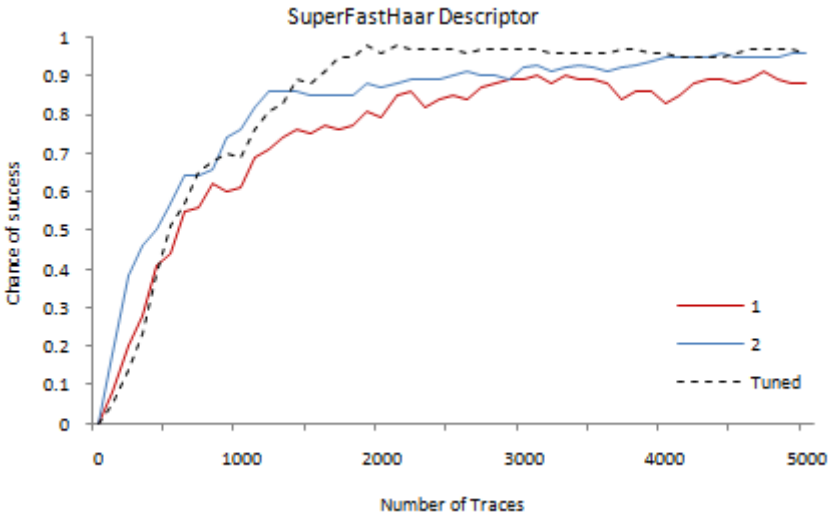

Figure 7.4.2: Results of the Super Fast Haar Descriptor

Algorithm	Run Time	Time Per Trace
RAM	169 minutes	20.28 ms
RAM + Fast Haar 10	123 minutes	14.76 ms
RAM + Fast Haar 20	140 minutes	16.8 ms
RAM + Super Fast Haar 1	55 minutes	6.6 ms
RAM + Super Fast Haar 2	63 minutes	7.6 ms

Table 7.2: Timing results for the various descriptors

7.4.3 Matching Heuristics

The second and fourth entry in table 7.1 are both functions used in the matcher. To decrease the number of POIs related to each other the POIs of the reference trace were split into two groups based on the sign of the Mexican Hat wavelet response. When matching a target trace to the reference trace based on the sign of the response of the POI in the target trace it is only compared with the POIs in the group with the same sign. This should reduce the amount of POI comparisons with 50% without affecting the success rate.

Unfortunately, our tests did not show a significant increase in speed. This may be due to recursive character of the overall algorithm which already reduces the number of POIs comparisons greatly (only does in close vicinity are used). The overhead of selecting the group and possible cache misses may negate the expected gain in speed. Due to time constrains we were not able to pursue this option further but we recommend it should definitely be revisited.

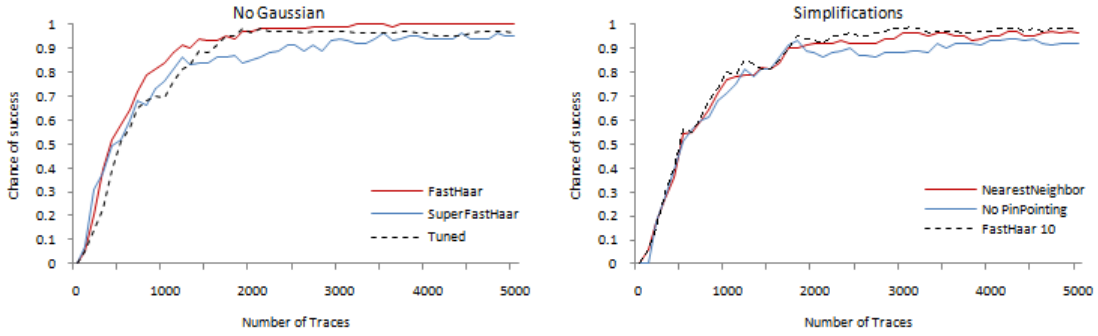

Figure 7.4.3: The algorithm without Gaussian (Left) without pinpointing (Right) and with Nearest Neighbor interpolation (Right)

7.4.4 Simplifications

When the first version of the algorithm was designed a lot of design choices were based on the U-SURF paper and on early testing. Here more extensive testing is used to see whether all computation steps are necessary. Time constrains limited the number of tests that could be performed here but some interesting results were found nonetheless. Other tests suggested that increasing the standard deviation of Gaussian used in the descriptor increases performance of the algorithm. This means that looking at samples close to the POI may not be the best way to match them to POIs in other traces. The Gaussian weighting function was removed and the results are shown in the left graph in figure 7.4.3. The graph shows an increase in performance for both the Fast and Super Fast Haar descriptors. This means that either the values that have been used thus far for the standard deviation of the Gaussian were way too low or the Gaussian is not needed at all. The gain in success rate also comes with a speed boost for the Fast Haar descriptor as can be seen in table 7.3.

Other choices that were based on early testing were the use of a pinpointing method and the choice of the interpolation scheme of the warper. Instead of using pinpointing to find the precise position of a POI one may assume that the precision of 10% of the wavelet scale would suffice. The right graph in figure 7.4.3 shows that there is a slight but significant performance drop when not using pinpointing. The timing results in table 7.3 show that instead of reducing the time to process the traces it took longer. This could be due to the fact that the feature vectors of the POIs are of lesser quality causing the more cross matching to solve for the matcher.

From the same figure and table can be seen that choosing a nearest neighbor interpolation scheme for the warper slightly lowers performance. The reduction in processing time also slightly decreases but since in practice it is more important to gain a little higher chance of success than a few minutes in time Cubic interpolation scheme is preferred.

Algorithm	Run Time	Time Per Trace
RAM + Fast Haar 10 + No Gauss	76 minutes	9.1 ms
RAM + Super Fast Haar 1 + No Gauss	54 minutes	6.5 ms
RAM + Fast Haar 10 + Nearest Neighbor	121 minutes	14.6 ms
RAM + Fast Haar 10 + No Pinpointing	116 minutes	13.9 ms

Table 7.3: Timing results for the simplification attempts

Algorithm	Run Time	Time Per Trace
Static Alignment	12 minutes	1.44 ms
SW-DPA	18 minutes	2.16 ms
RAM + Super Fast Haar + No Gauss	63 minutes	7.6 ms
RAM + Fast Haar + No Gauss	76 minutes	9.1 ms
Elastic Alignment	3115 minutes	373.8 ms

Table 7.4: Timing results for the various alignment methods. The time listed for SW-DPA is the additional time it took to perform the DPA attack

7.5 Comparison Results

The algorithms that resulted from the experiments was compared to Elastic Alignment, Static Alignment and SW-DPA. The results are shown in figure 7.5.1. The graph on the left shows that our proposed algorithm performs slightly worse than Elastic Alignment for low number of traces. Static Alignment and SW-DPA are not able to overcome the countermeasures within 5000 traces.

Since our main goal was to design an algorithm that was much faster than Elastic Alignment we compared the running times of the algorithms. The results are shown in table 7.4. Our proposed algorithm performs much faster than Elastic Alignment. So when time is a factor this clearly is the algorithm of choice.

To further express the potential of this algorithm the success-to-minute ratio was calculated from the already available data. To give a clear picture of the advantage of RAM over Elastic Alignment the data needed to be transformed and interpolated. The the chance of success after a certain number of minutes is plotted in figure 7.5.1. In the graph on the right can be seen that RAM achieves success rates of over 95% in 30 minutes where Elastic Alignment would take over 32 hours to do the same.

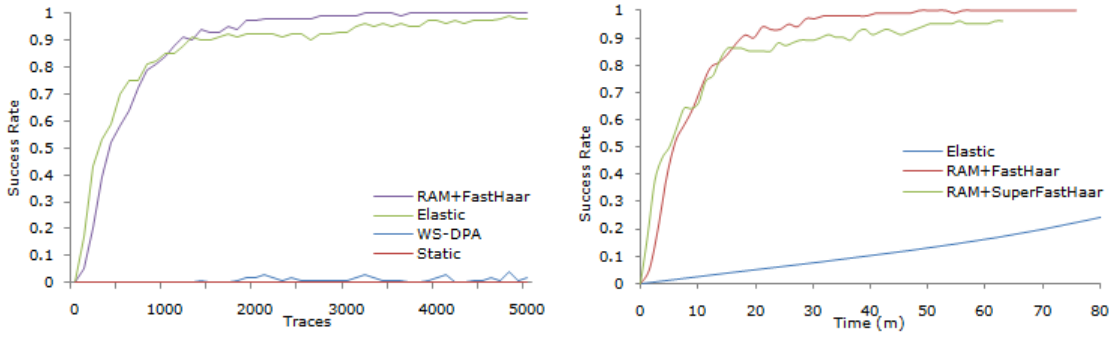

Figure 7.5.1: Success To Trace Ratio (Left) and Success to Time Ratio (Right). Note that data points in the right graph are not measurements but translated and interpolated points from other experiments.

8 Conclusion

8.1 Conclusion

Differential Power Analysis is a widely used side channel attack that uses leakage in the power signal of a device to extract its secret key. It requires a large number of power signal measurements of the device while it is encrypting known plaintexts and uses statistics to analyze them.

For this attack to work it is important that power traces are aligned in the time domain. If this is not the case the number of traces required to successfully perform the attack increases with several orders of magnitude. One of the countermeasures against this type of attacks is based on this requirement. By using an unstable clock or introducing dummy operations the measurements are misaligned.

Although several algorithms exist to align the measurements they are all limited in either success-to-number-of-traces-ratio or computation time. We introduce a fast algorithm with a reasonable success ratio. We used a smartcard with a software implementation of triple DES to measure the performance of the algorithm.

We used several experiments to tune the parameters of the algorithm. All of these parameters are dependent on properties of the traces to be aligned. This results in an algorithm which is easy to use for an attacker since there are no parameters that need tuning. Although our tuning efforts resulted in reasonable results it is very well possible that further tuning increases performance even more.

We compared our proposed algorithm with Static Alignment, Sliding Window DPA and Elastic Alignment. While Static Alignment and Sliding Window DPA are not capable of properly aligning the used trace set, Elastic Alignment showed excellent performance but was relatively slow. It took almost 52 hours to process the 500,000 traces from our data set. Our proposed algorithm performed similarly in terms of success ratio compared to Elastic Alignment but took only 76 minutes to process the data set.

The proposed algorithm consists of four components. Each of these components can be replaced so that new approaches can easily be tested. This provides an easy to use framework for alignment algorithms.

At his moment the matcher component is the weakest link. The algorithm detects points of interest of different scales to align the traces with. If a mismatch occurs between two traces with points of a large scale the resulting trace is not be usable. A better matching scheme may prevent this.

We conclude that RAM Alignment outperforms all of its competitors: WS-DPA, Static Alignment and Elastic Alignment[1].

[1] Near the end of writing this thesis, Riscure reported they developed a new version of Elastic Alignment

8.2 Further research and suggestions

Although our proposed algorithm already outperforms it competitors, there are still some open ends. In this section is briefly discussed what topics could be interesting for further research.

One of the most important differences between modern smartcards and the smartcard used for testing is that modern smartcards have significantly more noise in there power signal. It would be interesting to know how well the proposed algorithm reacts to such noise. To do this similar tests could be run on smartcards with different noise profiles and other countermeasures enabled.

To boost the success rate of the algorithm it would definitely be interesting to further tune the constants. The tuning so far has shown an increase in the success rate from about 20% to 95-100%. With more extensive tuning it might be possible to achieve these success rates with less traces.

To boost the speed of the algorithm the focus should be on the matcher as this is currently the most demanding component in the algorithm. To lower the $O(n^2)$ in that component one could sort the POIs selected for matching based on their distance to the origin, the distance to a description vector with only zeroes. In that way it is possible to discard of POIs that definitely fall outside the maximum distance range in $O(n\log(n))$. This lowers the amount of POIs that are related to each other in the $O(n^2)$ part of the algorithm.

Splitting the POIs of the reference trace into two groups based on whether the Mexican Hat wavelet response was positive or negative could decrease the number of POIs related to each other with 50%. We did some experiments with this but it the results were inconclusive. It is recommended to revisit this option.

After the sumtrace is calculated most operations are done independent to each other. This allows for parallel computing which could greatly improve the calculation time. Also the application of the algorithm allows for easy use of parallel computing. Since the traces are processed independently only being compared with the reference trace (read operations only) it is possible to assign different parts of the trace set to different threads without much overhead.

Now, for matching the POIs are selected based on their position in the trace (they are selected based on the POIs found on the previous (larger) scale). Instead, the POIs of the reference trace could be stored in a kd-tree-like structure. This returns POIs in nearest-neighbor-first order. To built such a tree takes $O(n\log(n))$. This will cause a lot more cross matches that need to be dealt with, on the other hand it might be possible to recover from mismatches at higher scales since they are no longer used to select potential good POI matches. Right now if a mismatch occurs at a large scale all lower scales go wrong too.

In order to make the algorithm easy to use all of the parameters are depending on properties of the trace. However, some (the detection of POIs) depend on the standard deviation of the trace. This is not ideal since some traces contain patterns that are

which is about 3 times faster than the implementation used in this thesis.

not part of the encryption, often due to communication, that significantly increase the standard deviation. To overcome this properly one could select a part of the trace over which the standard deviation should be calculated instead of the full trace.

Last but not least, different detector and description schemes could be used to increase the aligned trace quality. This fundamentally changes the algorithm but the detector-descriptor-matcher-warper-framework proposed here could be useful in many configurations.

Bibliography

[1] J. Allen. Short term spectral analysis, synthesis, and modification by discrete Fourier transform. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 25(3):235–238, 2003.

[2] D. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[3] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[4] J. Beis and D. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1000–1006. IEEE, 1997.

[5] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 135–152, 2004.

[6] R. Burden and J. Faires. Numerical analysis. 2004. *Brooks Cole, Pacific Grove, California, United States.*

[7] C. Clavier, J. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In *Cryptographic Hardware and Embedded Systems-CHES 2000*, pages 13–48. Springer, 2000.

[8] F. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, 1984.

[9] J. Fourier. Théorie Analytique de la Chaleur, 1822. *Firmin Didot–Père et Fils (now public domain, scanned by Google) Fourier-Théorie Analytique de la Chaleur. pdf.*

[10] F. Garcia, G. de Koning Gans, R. Muijrers, P. Van Rossum, R. Verdult, R. Schreur, and B. Jacobs. Dismantling MIFARE classic. *Computer Security-ESORICS 2008*, pages 97–114, 2008.

[11] J. Hogenboom. Principal Component Analysis and Side-Channel Attacks, MSc Thesis Nr 634. 2010.

[12] P. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in cryptology-CRYPTO'99: 19th Annual International Cryptology Conference, Santa*

Barbara, California, USA August 15-19, 1999 proceedings, page 388. Springer Berlin Heidelberg, 1999.

[13] D. Lowe. Object recognition from local scale-invariant features. In *ICCV*, page 1150. Published by the IEEE Computer Society, 1999.

[14] S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Springer Verlag, 2007.

[15] T. Messerges. Using second-order power analysis to attack DPA resistant software. In *Cryptographic Hardware and Embedded Systems-CHES 2000*, pages 27–78. Springer, 2000.

[16] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 525–531. IEEE, 2001.

[17] A. Moradi, M. Shalmani, and M. Salmasizadeh. Dual-rail transition logic: A logic style for counteracting power analysis attacks. *Computers & Electrical Engineering*, 35(2):359–369, 2009.

[18] N. Ricker. Wavelet contraction, wavelet expansion, and the control of seismic resolution. *Geophysics*, 18:769, 1953.

[19] S. Salvador and P. Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *KDD Workshop on Mining Temporal and Sequential Data*, pages 70–80, 2004.

[20] K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Solid-State Circuits Conference, 2002. ESSCIRC 2002. Proceedings of the 28th European*, pages 403–406. IEEE, 2002.

[21] K. Tiri and I. Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *Proceedings of the conference on Design, automation and test in Europe-Volume 1*, page 10246. IEEE Computer Society, 2004.

[22] J. van Woudenberg, M. Witteman, and B. Bakker. Improving differential power analysis by elastic alignment. *Topics in Cryptology–CT-RSA 2011*, pages 104–119, 2011.