RAM: Rapid Alignment Method

Ruben A. Muijrers[1], Jasper G.J. van Woudenberg[2], and Lejla Batina[1,3]

[1] Radboud University Nijmegen, ICIS/Digital Security group
Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands
rubenmuijrers@yahoo.com, lejla@cs.ru.nl
[2] Riscure BV, 2628 XJ Delft, The Netherlands
vanwoudenberg@riscure.com
[3] K.U.Leuven ESAT/SCD-COSIC and IBBT
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
lejla.batina@esat.kuleuven.be

Abstract. Several countermeasures against side-channel analysis result in misalignment of power traces, in order to make DPA more difficult. In this paper we propose a new algorithm to align the measurements after this desynchronizing through the variations of the internal clock, random delays, etc. The algorithm is based on the ideas of SIFT and U-SURF algorithm that were originally proposed for image recognition. The comparison with other known methods favors our solution in terms of efficiency and computational complexity.

Keywords: smart cards, side-channel analysis, DPA, random delay countermeasure, alignment methods.

1 Introduction

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. As algorithms used in modern chips are mathematically rather secure, attackers shift focus to specific implementations and the secret information that leaks through physics.

One of the focus areas is the power consumption of a device. About a decade ago Kocher et al. proposed DPA, a method that allows attackers to extract the secret key used for cryptographic operations from a small device such as a smartcard [7]. Since then many countermeasures have been developed, as well as ways to reduce the effects of these countermeasures.

Kocher's method and its many variants depend on the assumption that during the encryption, the timing of each operation during the cryptographic operation is constant between multiple executions of the algorithm. By adding dummy operations at random points in time or using an unstable clock this assumption is broken, and thereby DPA attacks become less effective. However, there exist algorithms that attempt to recover DPA information even in this case of the traces being misaligned e.g. Static Alignment [9] and Elastic Alignment [10].

E. Prouff (Ed.): CARDIS 2011, LNCS 7079, pp. 266–282, 2011.

The power traces are altered such that the target is susceptible to a DPA attack again. Static alignment is fast, but it only aligns one point of the cryptographic operation and does not take into account differences in the timing of the individual operations. Elastic Alignment performs continuous alignment, but it is computationally intensive.

Our proposed algorithm is designed to perform continuous alignment, without the drawback of the high computational requirements of Elastic Alignment. The algorithm is inspired by U-SURF [2] 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 improves the running time substantially.

This paper is organized as follows. Section 2 describes relevant prior work. In Sect. 3 we explain the main ideas behind wavelets as a starting point for the new method. We present the algorithm and its main components in Sect. 4. Our experiments and results are given in Sect. 5. Finally, Sect. 6 concludes this work.

2 Related Work

Alignment algorithms try to reduce the effects of an unstable clock and dummy operations by aligning all traces in a set to a reference trace, which is a common approach. In this paper we refer to the trace we align to as the *reference trace* and the traces that need to be aligned as the *target traces*.

Static alignment was described by Mangard et al. in 2007 [9], see Ch. 8. The idea is as follows: the attacker chooses a fragment in a 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. 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.

Elastic Alignment [10] attempts to match each sample in a target trace to samples in the reference trace. It can match sequences of samples in the target trace to a single sample in the reference trace, and vice versa. The algorithm minimizes the sum of the different sample values between matched samples. This match is used to stretch and compress samples in the target trace to match the reference trace. The result is an algorithm that can perform a continuous synchronization of any two traces. The authors show Elastic Alignment runs in $O(n)$, with n the number of samples in the trace to align. However, the continuous matching procedure is still quite computationally expensive.

Sliding Window DPA was proposed by Clavier et al. in 2000 [6]. as an algorithm that 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. 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. In [10] 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.

SIFT stands for Scale Invariant Feature Transform. It is a feature generation method proposed by Lowe in 1999 [8]. 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, which are similar problems to those due to the misalignment. U-SURF [2] stands for Upright-SURF where SURF stands for Speeded Up Robust Features. U-SURF does the same as SURF, except that it skips a step where orientations of the points of interest (POI). Our solution is inspired by both, as we translate the ideas of SIFT and U-SURF from the 2 dimensions (images) to one dimension (power traces).

In this work we aim at creating a continuous alignment algorithm, with a strong focus on improving computational complexity.

3 Alignment with Wavelets

To align two power traces one first needs to choose points in these power traces to align upon. In Elastic Alignment, the points used for alignment are all the samples in every resolution of each trace. This results in the calculation being computationally demanding. The proposed algorithm uses far less points for alignment and interpolates in between these points.

The points that are used to align on must be efficient to compute and recognizable in each trace. To find these *points of interest* (POI) in multiple traces the proposed algorithm searches for certain patterns in the traces. These patterns are identified with the use of wavelets. Wavelets give information about a power trace with excellent temporal resolution, as opposed to a Fourier Transform which has a trade-off between frequency resolution and temporal resolution. The benefits of wavelets in getting rid of noise were already presented in [5], but they were not used for re-synchronization. Another important advantage is that the wavelets used in the proposed algorithm can be applied in $O(1)$. This is further explained in the remainder of this section.

3.1 Wavelets

The term "wavelet" means small wave. It is a signal with an amplitude that starts out at zero, than increases, and then decreases back to zero. An example of this can be created by multiplying a sine wave with the Gaussian distribution function. This will result in a Morlet wavelet as can be seen in Fig. 1. Given a wavelet function the wavelet response $WR(t, \psi)$, of a wavelet defined by function $\psi(t)$ at a point t in a signal defined by function $f(t)$, can be computed by

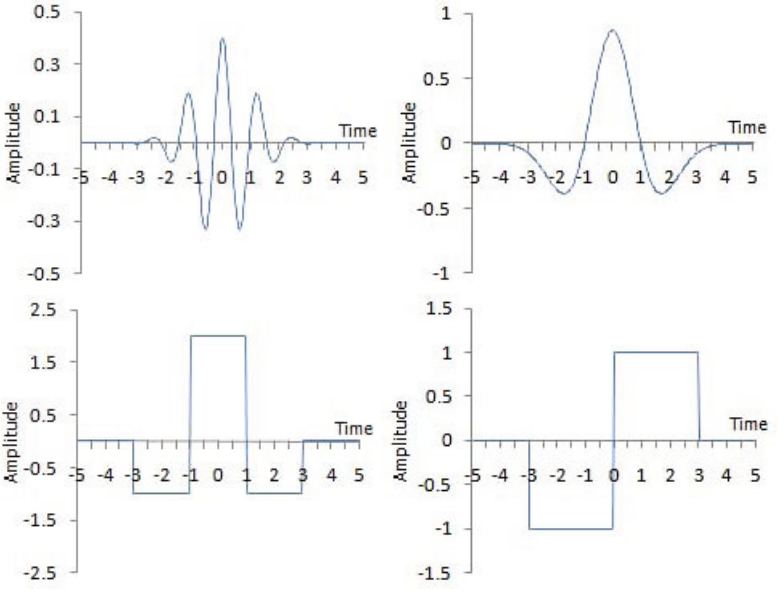

Fig. 1. Examples of a Morlet wavelet (top-left), Mexican Hat wavelet (top-right), Mexican Hat block wavelet (bottom-left) and Haar wavelet (bottom-right)

calculating the convolution of the wavelet and the signal at that point as is shown in equation (1):

$$WR(t, \psi) = \int f(x) \cdot \psi_{\alpha,t}(x) dx, \tag{1}$$

where $\psi_{\alpha,\beta}$ is the wavelet function ψ scaled with α and translated with β as is shown in equation (2):

$$\psi_{\alpha,\beta}(t) = \frac{1}{\sqrt{\alpha}} \psi(\frac{t-\beta}{\alpha}). \tag{2}$$

When working with power traces the signal is not defined by a continuous function. Instead it is defined by a set of discrete data points. The wavelet response function reduces then to a summation which is shown in the following sum:

$$WR(t, \psi) = \sum_{x=0}^{len} f(x) \cdot \psi_{\alpha,t}(x). \tag{3}$$

where *len* is the number of sample points in the power trace.

For detection of POIs, wavelet responses are calculated at several scales and at several sample points in the trace. Two patterns have been tried, the first being a peak pattern which was identified with the Mexican Hat wavelet shown in Fig. 1. The second was the slope pattern which was identified with the Haar wavelet also shown in Fig. 1. To be able to compare wavelet responses of different scales the wavelet responses are divided by $\sqrt{s}$ where s is the scale of the wavelet.

3.2 Block Wavelets

The calculation of a wavelet response as defined in equation (3) takes $O(n)$ to compute, where n is the length of the power trace. By using the 1-dimensional variant of summed-area tables [1] this can be reduced to $O(1)$ when using block wavelets such as the Haar wavelet or the approximation of the Mexican Hat wavelet. First, the power traces that need to be aligned are transformed in a *summed trace*. Each element at position t in the summed trace is the sum of all the samples before and including sample t in the power trace. The formal definition of the summed trace S of power trace P is given in equation (4):

$$\begin{aligned} S[0] &= P[0] \\ S[t] &= S[t-1] + P[t]. \end{aligned} \tag{4}$$

The convolution of the Haar wavelet and the block version of the Mexican Hat wavelet with scale s at position t in summed trace S can now be calculated efficiently: the Haar wavelet now requires only 3 read operation regardless of its scale and the Mexican Hat requires 4. This follows directly from their definitions applied to summed traces, the result of which is shown in equations (5) and (6):

$$\begin{aligned} Haar(S,t,s) &= \frac{-(S[t] - S[t-\frac{s}{2}]) + (S[t+\frac{s}{2}] - S[t])}{\sqrt{s}} \\ &= \frac{-2 \cdot S[t] + S[t-\frac{s}{2}] + S[t+\frac{s}{2}]}{\sqrt{s}}, \end{aligned} \tag{5}$$

$$\begin{aligned} MexicanHat(S,t,s) &= \frac{-(S[t-\frac{s}{6}] - S[t-\frac{s}{2}]) + 2 \cdot (S[t+\frac{s}{6}] - S[t-\frac{s}{6}]) - (A[t+\frac{s}{2}] - S[t+\frac{s}{6}])}{\sqrt{s}} \\ &= \frac{3 \cdot (S[t+\frac{s}{6}] - S[t-\frac{s}{6}]) - (S[t+\frac{s}{2}] - S[t-\frac{s}{2}])}{\sqrt{s}}. \end{aligned} \tag{6}$$

4 New Algorithm

The proposed algorithm is inspired by U-SURF [2]. It has several tunable parameters but in order to ease the tuning attackers need to do, every parameter in the algorithm is related to properties of the trace.

The algorithm consists out of four components: Detector, Descriptor, Matcher and Warper. First the Detector finds *points of interest* (POI) in the reference trace and the target traces. The Descriptor then generates a feature vector for each POI based on their context. Using these vectors the Matcher will match POIs in the reference trace with POIs in the target trace. Finally, the Warper uses the matched points to stretch and shrink the target trace and to align them with the reference trace. This is a recursive process starting with POIs detected with large wavelets. Each recursive step the algorithm repeats itself on trace parts in between the POIs from the previous step with a reduced wavelet scale.

A *POI* in this paper is a data structure containing its position in a trace, the wavelet scale which was used to detect it and a feature vector. The position and scale are set by the Detector and the feature vector is generated by the Descriptor. An *area* in the code consists of two positions marking the start (inclusive) and end (inclusive) of the area. A *match* consists of a position in the reference trace, a position in the target trace and a confidence value.

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 1.

Algorithm 1. The main loop of RAM

```
procedure RAMINNER(m:matches, ref:sumtrace, tar:sumtrace, aref:area,
atar:area, ws:wavsize)
    if ws < Det_mw ∨ atar.size = 0 ∨ aref.size = 0 then
        return
    end if

    refpois = DETECT(ref, aref, ws)
    tarpois = DETECT(tar, atar, ws)
    DESCRIBE(ref, refpois)
    DESCRIBE(tar, tarpois)
    mat = MATCH(refpois, tarpois)

    if ISEMPTY(mat) then
        RAMINNER(m, ref, tar, aref, atar, ws/2)
        return
    end if
    for All areas (mref, mtar) between neighbouring matches in mat do
        RAMINNER(m, ref, tar, mref, mtar, ws/2)
        ADDMATCH(m, mref.end, mtar.end)
    end for
end procedure

procedure RAM(ref:trace, tar:trace)
    mat = (0,0)
    ws = argmax_x (x = Det_mw · 2^k | k ∈ ℕ ∧ x < ref.size)
    RAMINNER(mat, MakeSumTrace(ref), MakeSumTrace(tar), {0, ref.size −
1}, {0, tar.size − 1}, ws)
    return WARP(tar, mat)
end procedure
```

4.1 Detector

The Detector detects POIs in the reference and target trace. A POI is a pattern in the trace that can be quickly and repeatedly recognized in other traces of the same set. The entire Detector procedure is shown in Algorithm 2.

POIs are found by performing a wavelet transform. The response for wavelets of various scales is calculated for the trace. The algorithm starts with the largest wavelet scale that satisfies this predicate $Det_{mw} \cdot 2^k < l$ with k being a natural number, l being the trace length and Det_{mw} being the minimum wavelet scale. For each iteration the scale is halved until 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 wavelet responses are not calculated for every sample but with a step size of 10% of the wavelet scale. A small step size slows down the algorithm, whereas a big step size does not detect sufficient POIs. Preliminary testing shows a value of 10% yields a good tradeoff. To be able to compare the responses to a constant 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. The constant Det_{th} will be a parameter of the algorithm.

A *best-of-its-neighbors* filter is then 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 defined here 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, as otherwise too many POIs survive the filter, whereby the matcher will not be able to differentiate them properly.

The samples that remain after the filter will be the POIs. Since these are the points that will be used for alignment later on it is important that they are located as accurate as possible. To achieve this the remaining responses will be pinpointed by searching for the highest wavelet response in the vicinity (the step size) of the point.

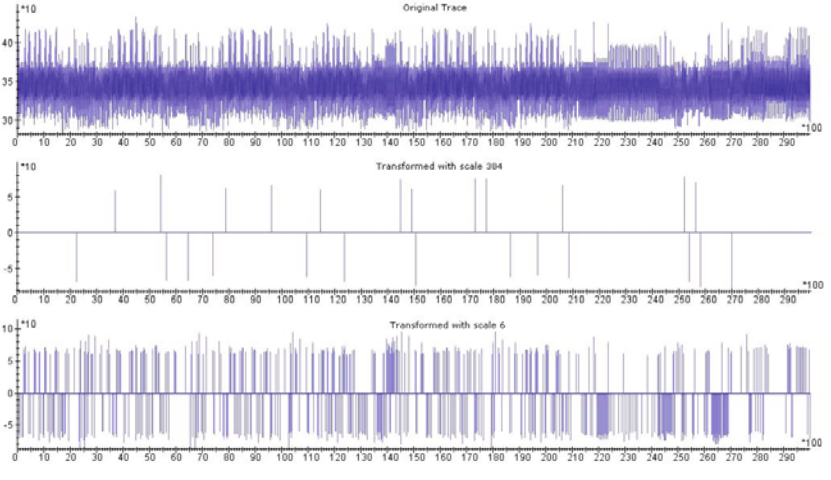

Fig. 2. A power trace (top) with its points of interest for two scales (middle=384, bottom=6)

Figure 2 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 more efficient to match the POIs from large scale wavelets to POIs from another trace. On the other hand, smaller wavelets give more information on how to align the traces.

The type of wavelet used here defines on which 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 tried. Both wavelets are shown in Fig. 1. We found that in the traces we analyzed that the Mexican Hat block wavelet outperformed the Haar wavelet in finding the same POIs in similar traces.

Algorithm 2. The main loop of the Detector

```
procedure DETECT(S:sumtrace, a:area, ws:wavsize)
    B = buffer trace
    step = max(1, ws/10)
    pois = empty list
    for i = a.start; i < a.end; i+ = step do
        B[i] = MEXICANHAT(S, i, ws)
        if abs(B[i]) < Det_th * stddev(RefTrace) then        ▷ Below threshold?
            B[i] = 0
        end if
    end for

    for i = a.start; i < a.end; i+ = step do
        if B[i] ≠ 0 ∧ BESTOFNEIGHBORS(B, i) then             ▷ If best, add
            ADDPOI(pois, PinPoint(S, i), ws)
        end if
    end for
    return pois
end procedure
```

4.2 Descriptor

The descriptor (Algorithm 3) aims to uniquely describe each POI by its context. It generates a feature vector which will be 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 single trace, the features must be fast to calculate. We use the same approach as in U-SURF: an area of Des_{as} times the scale around each POI is divided in several sections as can be seen in Fig. 3. Des_{as} is a tunable parameter of the algorithm. For each of the sections a few simple features are calculated. The sections are used to retain temporal information.

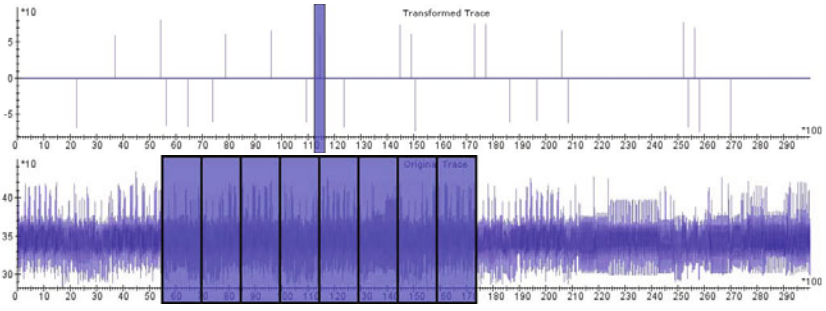

Fig. 3. A selected POI (top) and 8 sections which are used for the features (bottom)

For each section a number Des_{ss} of Haar wavelet responses are calculated (using the summed trace) at constant distance from each other with a wavelet scale of Des_{hw} times the scale of the POI. In order to focus on the area directly around the POI, a Gaussian curve centered at the POI and with a standard deviation of Des_{ga} is used to normalize the Haar wavelet response.

To include information about the polarity of the section all the wavelet responses are summed, and to include information about the intensity of the section all the absolute wavelet responses are summed. The number of sections is based on the feature count Des_{fc} (2 features per section). The constants Des_{hw}, Des_{ss}, Des_{fc}, Des_{ga} are tunable parameters of the algorithm.

4.3 Matcher

The Matcher (Algorithm 4) creates a mapping between the POI set of the reference trace and the POI set of the target trace based on the feature vectors. The Matcher has to be robust to the fact that not every POI appears in both trace and that descriptions of different operations may be similar.

For every POI from the reference trace a distance is calculated to every POI from the target trace with the same scale. To allow comparison against a threshold the normalized Euclidean distance is used, which is a special case of the Mahalanobis distance and is defined in equation (7).

$$d(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{\sum_{i=1}^{N} \frac{(x_i - y_i)^2}{\sigma_i^2}} \tag{7}$$

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. To do this the all the POIs have to be calculated before the matching starts (note that this is not shown in the algorithms), this will also speed up the algorithm. In some cases, especially those with large scales, there are not enough POIs to give a proper estimate for σ_i, but we find in practice this does not cause problems.

Algorithm 3. The main loop of the Descriptor

```
procedure DESCRIBE(S:sumtrace, pois:POIlist)
    for All POI p from pois do
        v = ZeroFilledVector(Des_fc)
        da = Des_as * p.scale                          ▷ size of the description area
        sc = Des_fc/2                                  ▷ section count
        ss = da/sc                                     ▷ section size
        o = p.pos − sc/2 * ss                          ▷ offset, most left sample point
        step = max(1, p.scale/Des_ss);

        for s = 0; s < sc; s++ do                      ▷ For each section
            for i = 0; i < ss; i+ = step do            ▷ For each sample in the section
                t = HAAR(S, o + s * ss + i, Des_hw * p.scale)      ▷ Calc Haar
                t* = GAUSSIAN(o + s * ss + i − p.pos, Des_ga)      ▷ Normalize
            end for
            v[2 * s]+ = t                              ▷ Add response for section
            v[2 * s + 1]+ = abs(t)                     ▷ Add absolute response for section
        end for
        SETFEATUREVECTOR(p, v)
    end for
end procedure
```

Each POI from the reference trace is matched with the POI with the smallest distance in the target trace. For efficiency, if the distance is greater than the threshold Mat_{md} the match is removed. Testing shows that this removes most of the mismatches as can be seen in Fig. 4.

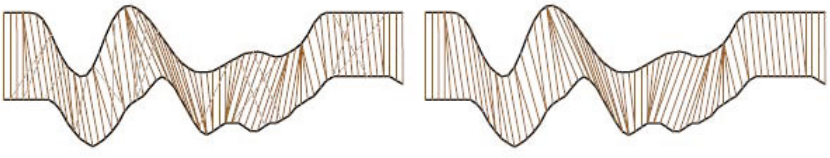

Fig. 4. Two matched POI sets, with $Mat_{md} = 1000$ (left) and $Mat_{md} = 4$(right)

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 is not allowed because it would violate the temporal behavior of the trace.

To resolve these cross matches the confidence of a match and a penalty function are 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 will than be removed from the set. This is repeated until all cross matches are resolved.

Algorithm 4. The main loop of the Matcher

```
procedure MATCH(ref:POIlist, tar:POIlist)
    matches = emptylist

    for All POI p from ref do                                   ▷ Match POIs
        q1 = argmin_q(DIST(p.feats, q.feats)) | q ∈ tar)        ▷ Closest
        q2 = argmin_q(DIST(p.feats, q.feats)) | q ∈ tar ∧ q ≠ q1)   ▷ Second closest
        d1 = DIST(p.feats, q1.feats)

        conf = 1 − d1/DIST(p.feats, q2.feats)
        if d1 < Mat_md then
            ADDMATCH(matches, {p, q1, conf})
        end if
    end for

    while crossmatches exist in matches do                      ▷ Remove crossmatches
        worst = argmin_m (m.conf ∗ 0.5^NUMCROSSES(m,matches) | m ∈ matches)
        REMOVEMATCH(matches, worst)
    end while
    return matches
end procedure
```

4.4 Warper

The Warper (Algorithm 5) takes a list of matched POIs from two traces and shrinks and stretches sections of the target trace so it will be 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 [4]. Hermite

Algorithm 5. The main loop of the Warper

```
procedure WARP(tar:trace, mat:Matchlist)
    t = TraceOfSize(tar.size)
    for m = 1; m < mat.size; m++ do
        ddest = mat[m].ref.pos − mat[m − 1].ref.pos      ▷ delta in reference trace
        dsrc = mat[m].tar.pos − mat[m − 1].tar.pos       ▷ delta in target trace
        r = dsrc/ddest                                   ▷ stretch/shrink factor

        for i = 0; i < ddest; i++ do
            t[mat[m].ref.pos + i] = INTERPOLATEDPOINT(tar, mat[m].tar.pos + i ∗ r)
        end for
    end for
    t[t.size-1] = tar[tar.size-1]                        ▷ Copy last sample
    return t
end procedure
```

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. Testing shows that differences were minimal but slightly in favor of the Cubic interpolation scheme. The other schemes have not been researched further.

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. We fill this with the values of the nearest sample.

5 Experiments and Results

In our experiments we intend to compare RAM to other well-known methods for dealing with misalignment: Static Alignment [9], SW-DPA [6] and Elastic Alignment [10]. We compare first-order success rates of CPA after performing the different alignment techniques applied to measurements taken from a software DES implementation.

The hardware used for our measurements consists of the standard side channel equipment: a smart card reader, a LeCroy $104Xi$ oscilloscope and our acquisition and analysis software. We also employed a 48 MHz analog lowpass filter. The traces were acquired from a programmable smart card, which contains a software implementation of DES, including software countermeasures that can be turned on and off. For these measurements we enabled the random delays countermeasure, which introduces random wait states during execution of the DES algorithm. The card runs at an external clock of 4 MHz.

Timing measurements for the comparison of different alignment methods are performed on a computer with an E6750 2.66 GHz processor and 2 Gb of RAM and running Windows 7. We sampled the traces at 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 compressed right after acquisition by averaging samples until we effectively obtained one sample per clock. In total we acquired 8248 consecutive clock cycles per DES execution, providing us with several rounds of DES. This was performed with random inputs for a total of 500k traces.

A typical trace from the resulting set is shown in Fig. 5. Note that the y-axis shows negative mVolts. This is due to an offset introduced during acquisition and has no consequences for our research.

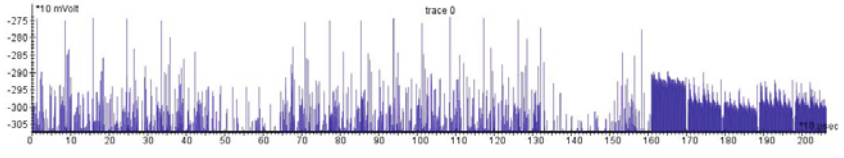

Fig. 5. A typical trace from the data set, showing the last round of the encryption

5.1 Settings

Here we elaborate on different settings used for alignment algorithms we compare our algorithm with.

Static alignment settings: For static alignment we selected a trace fragment of 600 samples at the end of the encryption. The allowed shift for this fragment was set to 250 samples.

Sliding window settings: We preprocessed the traces for a SW-DPA attack [6], by averaging the samples as described previously. The number of samples per clock cycle was set to 1 and the size of the sliding window that we used was 50, 100 and 200.

Elastic alignment settings: We also compare our proposed algorithm to Elastic Alignment. We set the radius in which FastDTW (as in [10]) will search for the optimal warppath. A radius of 70 provided a good trade-off between alignment quality and speed.

RAM align: For the new algorithm the initial values of the constants were chosen close to the values of U-SURF [2] whenever possible. The other baselines for the constants were tuned on the basis of previous experimenting. Using the correlations of the target traces with the reference trace it was decided whether increasing or decreasing the constant yields better results. Various values for the constants that we tested are given in Table 1.

Table 1. Different values for the constants that were used for tuning the algorithms

Constant	Notation	Values tested	Baseline value
Minimum Wavelet Size	Det_{mw}	3, 6, 12	6
Response Threshold	Det_{th}	2, 2.5, 3	3
Area Size	Des_{as}	12, 16, 20	20
Feature Count	Des_{fc}	16, 24, 32	32
Haar Wavelet Size	Des_{hw}	1.5, 2.5	-
Standard Deviation Gaussian	Des_{ga}	2.8, 3.3, 3.8	-
Samples Per Section	Des_{ss}	10, 20	-
Maximum Distance	Mat_{md}	2.3, 2.5, 2.7	2.5

Some additional explanations for constants and their values are given below.

Minimum Wavelet Size (Det_{mw}) This constant specifies the stopping criterion for the detector. The detector starts with large wavelets and decreases them after every iteration. If this wavelet size is reached then the detector is finished. In addition, lowering this value increases computation time, as usually there are much more possible POIs at low wavelet sizes, which possibly complicates correct matching. Lowering this value provides more precision for alignment and could increase the overall score.

Response Threshold (Det_{th}) The detector will only select samples to become a POI if the wavelet response at that point is greater than Det_{th} times

the standard deviation of the trace. A higher value input less points to the matcher, but may result in too few POIs to do proper alignment. A lower value also increases the running time of the algorithm. The matching algorithm is quadratic in the number of POIs detected, which makes this very computationally demanding.

Area Size (Des_{as}) This specifies the size of the area around the POI which will be used for the generation of the feature vector for this POI. The area of this POI will be 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.

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 for relating POIs.

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 at lower signal frequencies. The wavelet becomes bigger and is less sensitive for higher frequencies.

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. Lowering the value of this constant means that the description is focused on samples closer to the POI.

Samples Per Section (Des_{ss}) This is the number of samples used per section. The more samples used, the more accurate the descriptor can describe that section. If significantly less samples are used it speeds up the algorithm.

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 will be discarded. Lowering this value will prevent cross matching, but lowering it too much will discard usable matches.

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. More detailed graphical representation of the constants tuning is given in the appendix. After tuning the best results were obtained with the following values for the constants: $Det_{mw} = 3, Det_{th} = 2.5, Des_{as} = 16, Des_{fc} = 32, Des_{hw} = 2, Des_{ga} = 3.3, Des_{ss} = 10, Mat_{md} = 2.3$.

Result Analysis. The final results are the calculated first order success rates. After alignment, the trace set is split into subsets. For each subset a number of traces N_s is selected and a CPA [3] 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:

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

.

5.2 Comparison Results

The running times of the different algorithms are listed in table 2. The proposed algorithm clearly outperforms Elastic Alignment in computation time. Although Static Alignment and SW-DPA are faster they are not able to align the traces in such a way that a successful DPA attack can be performed as can be seen in Fig. 6. When comparing the success rate to the number of traces used the proposed algorithm performs similar to Elastic Alignment. However, when comparing the success rate to the time it took to align the traces, the proposed algorithm outperforms Elastic Alignment with an order of magnitude.

Table 2. Timing results for the various alignment methods. The time listed for SW-DPA is the additional time it took to perform the DPA attack.

Algorithm	Run Time	Time Per Trace
Static Alignment	12 minutes	1.44 ms
SW-DPA	18 minutes	2.16 ms
RAM	76 minutes	9.1 ms
Elastic Alignment	3115 minutes	373.8 ms

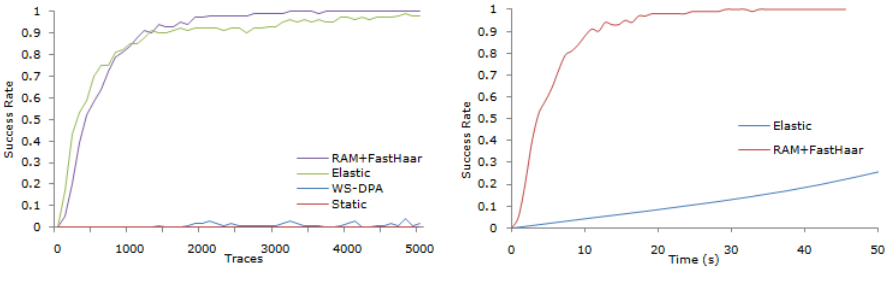

Fig. 6. Success To Trace Ratio (Left) and Success to Time Ratio (Right). Note that data points in the right graph are not measurements but translated and interpolated points from other experiments.

6 Conclusions and Future Work

Although several algorithms to align the measurements exist today they are all limited in either success-to-number-of-traces-ratio or computation time. In this work we introduce a new algorithm that obtains the best performances of previous works in terms of both the success rate and computation time. 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.

We illustrate our results by experiments on 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 the

parameters are dependent on properties of the traces to be aligned. This results in an algorithm which is easy to use. However, it is possible that further experimenting (with other implementations and platforms) could result in other values for the parameters.

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.

Acknowledgments. We would like to thank Riscure for providing an environment for fruitful discussion during the research, and for providing the side-channel analysis platform that was used for this work (Inspector). This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State and by the European Commission under contract number ICT-2007-216676 ECRYPT NoE phase II and by the K.U.Leuven-BOF (OT/06/40).

References

1. Ballard, D.: Generalizing the Hough transform to detect arbitrary shapes. Pattern Recognition 13(2), 111–122 (1981)
2. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-Up Robust Features (SURF). Comput. Vis. Image Underst. 110, 346–359 (2008)
3. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
4. Burden, R., Faires, J.: Numerical analysis, Paciific Grove, California, United States (2004)
5. Charvet, X., Pelletier, H.: Improving the DPA attack using Wavelet, transform. In: NIST Physical Security Testing Workshop (2005)
6. Clavier, C., Coron, J.-S., Dabbous, N.: Differential Power Analysis in the Presence of Hardware Countermeasures. In: Koç, Ç.K., Paar, C. (eds.) CHES 2000. LNCS, vol. 1965, pp. 252–263. Springer, Heidelberg (2000)
7. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
8. Lowe, D.G.: Object recognition from local scale-invariant features. In: Proceedings of the International Conference on Computer Vision, ICCV 1999, vol. 2. IEEE Computer Society, Washington, DC, USA (1999)
9. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security). Springer-Verlag New York, Inc., Secaucus (2007)
10. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving Differential Power Analysis by Elastic Alignment. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 104–119. Springer, Heidelberg (2011)

Appendix

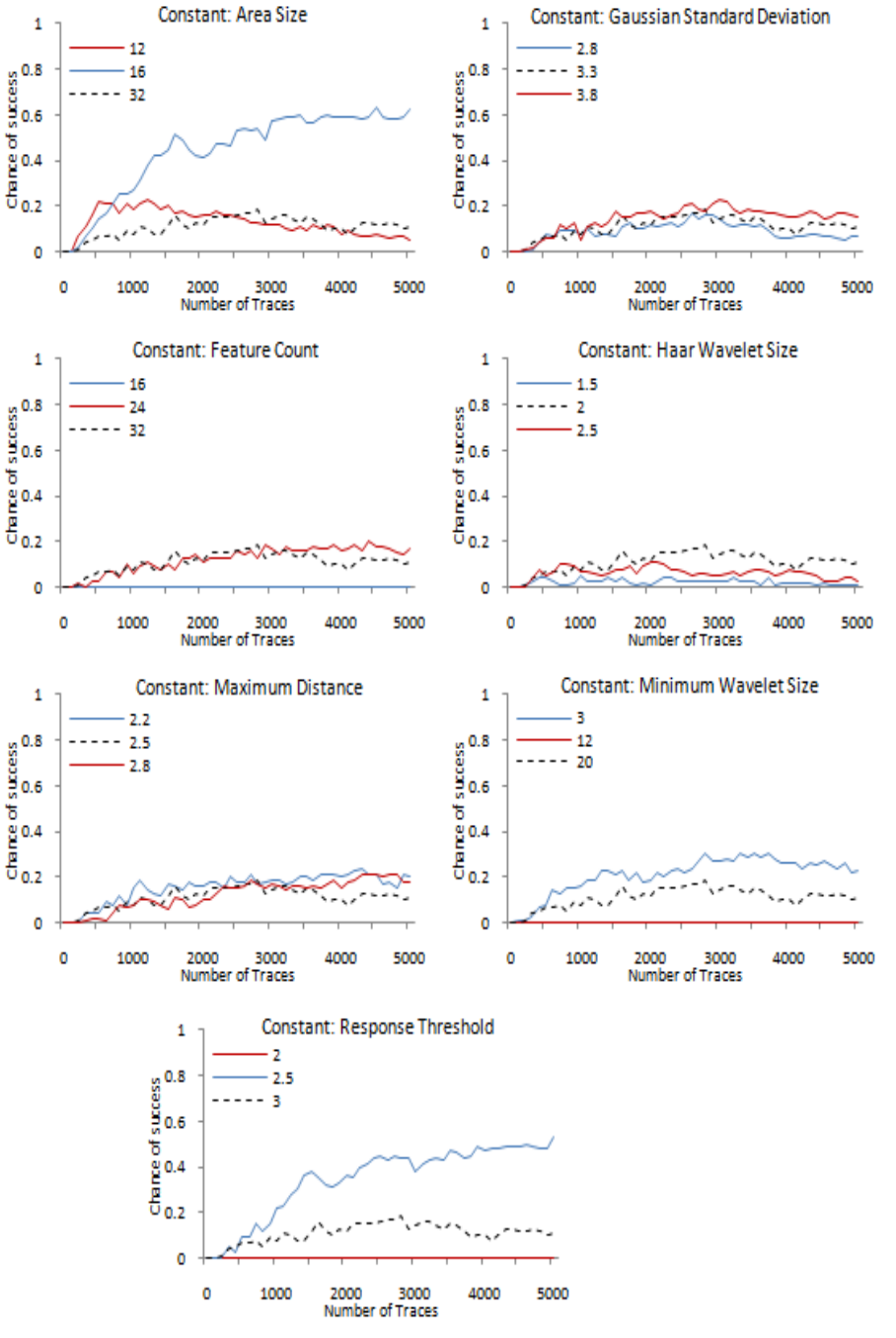

Fig. 7. The results for tuning the constants