Side-Channel Attacks on Fingerprint Matching Algorithms

Markus Dürmuth
Ruhr-University Bochum
Universitätsstr. 150
Bochum, Germany
markus.duermuth@rub.de

David Oswald
The University of Birmingham
School of Computer Science
Birmingham, UK
d.f.oswald@bham.ac.uk

Niklas Pastewka
Ruhr-University Bochum
Universitätsstr. 150
Bochum, Germany
niklas.pastewka@rub.de

ABSTRACT

Biometric authentication schemes are frequently used to establish the identity of a user. Often, a trusted hardware device is used to decide if a provided biometric feature is sufficiently close to the features stored by the legitimate user during enrollment. In this paper, we address the question whether the stored features can be extracted with side-channel attacks. We consider several models for types of leakage that are relevant specifically for fingerprint verification, and show results for attacks against the Bozorth3 and a custom matching algorithm. This work shows an interesting path for future research on the susceptibility of biometric algorithms towards side-channel attacks.

Keywords

biometry; fingerprint matching; side-channel analysis; simple power analysis; Bozorth3

1. INTRODUCTION

Biometric features are frequently used as a convenient method to identify and authenticate individuals. Common use cases of biometric user authentication include fingerprint and face recognition in laptops and smartphones, or access control to buildings using iris scanners and fingerprint readers. The user registers with the reader by providing one or several recordings of the appropriate biometric feature, the *gallery recording* or *gallery image*. When a user authenticates, he provides a fresh recording of the feature, the *probe recording* or *probe image*, and the device checks if both are "sufficiently similar".

In this paper, we address the question whether the stored gallery images can be recovered by observing the side-channel leakage of the respective matching algorithms. While this scenario may seem artificial at first, there is a variety of possible cases where such attacks are realistic: For example, in a so-called match-on-card system [26], a smartcard (without sensor) receives the fingerprint image (or the extracted features) from an external reader and performs the comparison internally. In this case, the adversary can easily send chosen inputs to the card and measure classical side channels like power consumption. Another example are PC software libraries like `libfprint`, which (in many cases) store and process the fingerprint on the PC, not the reader [17]. However, the adversary usually would require super user rights or physical access to directly read the respective files. Yet, a variety of software side channels, e.g. cache timing [4], potentially across VM boundaries [27], or branch prediction [2], may enable an adversary with normal user privileges to obtain the fingerprint.

In this context, it is important to note that, in contrast to a password or a cryptographic key, biometric features cannot be changed by the user. A fingerprint is very likely re-used between several devices, unless the user choses to use one finger per system. In practice, most of the time, the index fingers and thumbs will be preferred. Thus, the same fingerprint that unlocks the user's smartphone may also grant access to his computer as well as an office building, and be used for authentication when using his electronic passport. Hence, attacking a low-security system (e.g., a PC-based matcher or a low-cost fingerprint-enabled door lock) may be sufficient to gain access to high-security areas (e.g., a secure facility).

1.1 Related work

In the literature, attacks on biometric systems are usually based on creating a "similar enough" copy of the respective biometric feature, e.g., a rubber fingerprint [7], a picture or video to defeat face recognition, etc. These methods have in common that they require one-time access to the original biometric property, for example a fingerprint on a glass or a phone surface, a high-resolution photograph, and so on [21]. In contrast, methods that do not require access to the original biometric property have been studied to a smaller extent. [24] gives an overview over the attack surface wrt fingerprint matching, and presents an attack based on hill climbing that reconstructs a fingerprint faster than brute force when the matching algorithm outputs the score (and not just a yes/no decision).

The authors of [5] demonstrate that simple power analysis (SPA) can be used to extract side-channel information from an FPGA matcher implementation [9] running on the SASEBO GII board. This leakage is then utilized in the hill climbing algorithm. In [10], this work is extended by using correlation power analysis (CPA) [6] targeting the registration phase. The authors can distinguish the correct minutia

TrustED'16, October 28 2016, Vienna, Austria

ISBN 978-1-4503-4567-5/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2995289.2995294

hypothesis from the incorrect ones with 800 side-channel traces with chosen input, however, state that the processing requires significant computing time on high-speed systems amounting to several hours per minutia.

In general, there is a wealth of literature with regard to side-channel analysis (SCA) of embedded devices, starting with the initial papers of Kocher et al. [15, 14]. Other notable publications include the proposal of electro-magnetic analysis [3], template attacks [8], and cache-timing attacks [4] (that in particular apply to PC and server systems). Side-channel attacks have been frequently applied to non-biometric authentication devices, including keyless entry systems [11], RFID smartcards [19], and two-factor tokens [20].

1.2 Outline

The remainder of this paper is organized as follows: In Section 2, we describe and discuss different models of side-channel leakage that we consider in this work. Then, in Section 3, we exemplify the attack using a textbook fingerprint matching algorithm, illustrating the techniques. In Section 4, we show an attack against the Bozorth3 fingerprint matching algorithm. Subsequently, in Section 5, we demonstrate that the attack is feasible in principle against a microcontroller implementation.

2. SIDE-CHANNEL ANALYSIS AND FINGERPRINT MATCHING

As shown in [10], applying techniques of differential power analysis (DPA) to fingerprint matching is challenging due to the relatively large candidate space and the linearity of matching algorithms. Furthermore, the resulting attacks are highly algorithm-specific. In this paper, we therefore focus on side channels in fingerprint matching stemming from the control flow of the targeted algorithms. To this end, we describe models that capture the behavior of larger classes of matching algorithms. In most cases, the respective leakage can be observed using SPA, i.e., by utilizing a single or few traces. However, more sophisticated approaches based on templates would also be applicable, potentially down to the single-instruction level [12, 23], and could further improve the success rate.

We explicitly do not address the problem of how to present chosen input fingerprints to the sensor or matching algorithm in practice. However, note that for match-on-card systems, one can send the (constructed) input fingerprint to the smartcard carrying out the matching. Sensors relying on visual input only could be made to accept high-resolution images (for example on a smartphone screen). In the case of more advanced sensors, the adversary may have to create a series of rubber fingerprints, e.g., using the techniques from [7].

2.1 Models for side channels in fingerprint matching algorithms

In the following, we describe general models for side channels in typical matching algorithms. We believe that these models cover a large amount of the exploitable leakage. This assumption is motivated by the practical case study in Section 5.

Timing model.

In the simplest case, the adversary measures the overall runtime of the algorithm. This may leak information on internal parameters of the algorithm. Consider the example of Alg. 2.1, which reflects a typical construct for matching fingerprints: If the algorithm takes longer to execute for a specific *input*, we can assume that *input* was greater than *internal constant*. However, a short duration indicates that *input* was smaller or equal to the constant. Hence, *internal constant* can be found with binary search.

Algorithm 2.1: Example algorithm for the timing and score model

```
1 Initialize: Set score to 0
2 begin
3 |  while input > internal constant do
4 |  |  input = input − 1
5 |  |  score = score + 1
6 |  end
7 end
```

Note that timing leakage can occur in various, potentially remotely detectable forms in practice. For example, a fingerprint reader may output a visual or acoustic signal to indicate that a fingerprint was rejected. The adversary may measure the time between the fingerprint being presented and the respective signal to obtain timing information.

Score model.

In a minutiae-based matching algorithm, the score determines whether an input fingerprint matches one in the gallery. A match is detected if the score exceeds a threshold. In the example of Alg. 2.1, the score is proportional to the number of loop iterations and hence could be retrieved via the timing side channel. In other cases, the score may leak through error messages or debug functionality.

Exact branches model.

The exact branches (and the approximate branches) model applies to the specific case of two nested loops with an if-condition in the inner loop. This reflects a common structure of minutiae-based fingerprint matching, where one loop iterates over the features in the gallery fingerprint, while the second loop is over the fingerprint to be matched (cf. Alg. 2.2).

Algorithm 2.2: Example algorithm for branches models

```
1  Initialize: Set score to 0
2  begin
3  |  for i = 1 to M do
4  |  |  for j = 1 to N do
5  |  |  |  if input[i] == unknown_array[j] then
6  |  |  |  |  score = score+1
7  |  |  |  end
8  |  |  end
9  |  end
10 end
```

The assumption is that additional instructions are executed when the if-condition is fulfilled, which can then be observed in the side-channel signal. In addition, we as-

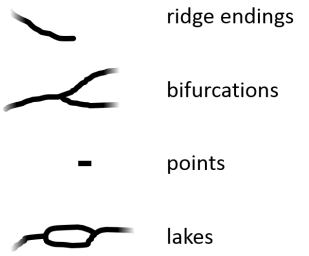

Figure 1: Examples for commonly considered types of minutiae.

sume that we obtain the exact values of the loop counters (by counting the iterations before the match). In other words, the exact branches model provides all pairs (i, j) with $input[i] = unknown\_array[j]$. We can then analyze these pairs to obtain the entries of the unknown array. Note that as an additional information, we can calculate the score by counting the number of pairs, i.e., this model encompasses the score model.

Approximate branches model.

The approximate branches model provides information similar to the exact branches model in that we can observe when the if-condition is true. However, the approximate branches model assumes that we only obtain the exact counter value of the outer loop, while the state of the inner loop counter is only approximately known. Looking back at Algorithm 2.2, we can identify the values of the unknown array, but the exact position within the array remains uncertain.

3. ATTACKING A SIMPLE MATCHING ALGORITHM

As a first illustrative example, we consider a straightforward textbook algorithm for minutiae matching [13].

3.1 The matching algorithm

The algorithm performs three steps: extraction of minutiae, optionally compensating for rotation and translation using the Hough transformation, and finally the matching step.

Minutiae extraction.

Many fingerprint recognition algorithms are based on so-called *minutiae*, characteristic points in the ridge structure of the fingerprint. One commonly considers *ridge endings* and *bifurcations*, and sometimes *points*, *lakes*, and others (cf. Figure 1). Minutiae extraction is relatively straightforward, and we used the MindTCT tool [25] to extract minutiae. We will not further consider minutiae extraction in the following.

General Hough transformation.

The positions of the recorded minutiae are not always identical, but can be "warped" for a number of reasons. The described algorithm considers translation and rotation of the fingerprints, and uses the Hough transformation to determine the most likely parameters.

Algorithm 3.1: General Hough transformation

input : Two minutiae sets $\left\{x_i^T, y_i^T, \theta_i^T\right\}_{i=1}^{M}$ and $\left\{x_j^Q, y_j^Q, \theta_j^Q\right\}_{j=1}^{N}$
output: Transformation parameters

```
1  Initialize accumulator array A to 0
2  begin
3      for i = 1 to M do
4          for j = 1 to N do
5              Δθ = θ_i^T − θ_j^Q
6              Δx = x_i^T − x_j^Q · cos(Δθ) − y_j^Q · sin(Δθ)
7              Δy = y_i^T + x_j^Q · sin(Δθ) + y_j^Q · cos(Δθ)
8              A[Δx][Δy][Δθ] = A[Δx][Δy][Δθ] + 1
9          end
10     end
11     return location of maximum in A
12 end
```

The algorithm iterates over all pairs of minutiae, one from each fingerprint. For each pair, it computes the difference of the angles $\Delta\theta$ and the translation between the two $\Delta x, \Delta y$ (after rotation), and then registers the triple $(\Delta\theta, \Delta x, \Delta y)$ in a three-dimensional array A. The position of the maximum value in this array gives the parameters of the optimal transformation, which is applied to the minutia in the second fingerprint. Pseudo-code of the Hough transformation is given in Algorithm 3.1.

Algorithm 3.2: Minutiae pairing algorithm

input : Two minutiae sets $\left\{x_i^T, y_i^T, \theta_i^T\right\}_{i=1}^{M}$ and $\left\{x_j^Q, y_j^Q, \theta_j^Q\right\}_{j=1}^{N}$
output: List of paired minutiae

```
1  Initialize: Set flag arrays f^T, f^Q and count as 0
2  begin
3      for i = 1 to M do
4          for j = 1 to N do
5              if f^T[i] == 0 and f^Q[j] == 0 and distance
               between i and j < t_d and rotation between i
               and j < t_r then
6                  f^T[i] = 1
7                  f^Q[j] = 1
8                  count = count + 1
9                  list[count] = {i, j}
10             end
11         end
12     end
13     return list
14 end
```

Matching.

The described matching algorithm performs a straightforward matching on the transformed fingerprint. It finds pairs of minutiae from both fingerprints that are more similar than a set threshold. Each matching pair that is found is marked as used and not considered further. The final score

is given by the number of matching minutiae. The algorithm is shown in Algorithm 3.2.

Performance.

For determining reasonable threshold values for the scores that determine when two fingerprints are considered "equal", and also for determining the accuracy of the described algorithm, we tested the algorithm on the Fingerprint Verification Challenge 2004 (FVC04) dataset [1]. We used database 3 set B, which contains 80 fingerprints from 10 different fingers, captured under controlled conditions with a resolution of 300×480 pixel. The minutiae were extracted using the MindTCT algorithm. Figure 2 shows two of these images from the same finger.

Figure 2: This figure shows two images of the same finger from the used fingerprint database.

We tested different parameters and report the resulting false acceptance rates (FAR) and false rejection rates (FRR) in Table 1.

T_d	T_r	T_t	T_{Score}	FAR	FRR
13	19	18	18	0.031	0.088
13	19	18	19	0.025	0.113
13	18	20	18	0.028	0.1
13	23	19	18	0.057	0.063
13	23	20	18	0.047	0.075
14	18	18	22	0.013	0.1
14	18	20	22	0.011	0.113
15	18	18	24	0.008	0.113

Table 1: Comparing FAR and FRR for different thresholds.

3.2 The attack

The attacker's goal is to find a fingerprint that is accepted by a fingerprint matcher against a fixed gallery fingerprint. He is given access to the side-channel information from the matcher on the same gallery fingerprint. Here, we are concerned with fingerprints represented as a list of minutiae and do not consider the problem of translating lists of minutiae to graphical representations of fingerprints.

We first present an attack basically guessing one minutiae after the other, then a (much) more efficient attack guessing multiple minutiae at once. In both cases, we consider the case that the Hough transform is applied as described above, or is not applied and the matching is directly performed on the given sets of minutiae.

3.2.1 *Guessing a single minutia at a time*

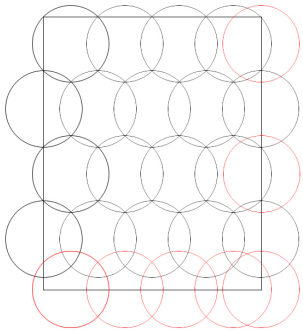

Figure 3: This figure illustrates how the points are chosen. The whole area is covered by the circles which represents the thresholds. The center points of the red circles are moved to fit in the area.

We assume the attacker can determine the score from the side-channel information available to him. If no Hough transform is used, the attacker can simply guess one minutiae after another. Assuming that the used threshold values are known (if they are unknown, they can be estimated easily), he can test several minutiae, where it is sufficient to use locations that are spaced so that they cover (including thresholds) the entire range (see Figure 3). Furthermore, the attacker can take the density of minutiae in different parts of the image into account, in particular the density in the center being much higher than in the surroundings (cf. Figure 4).

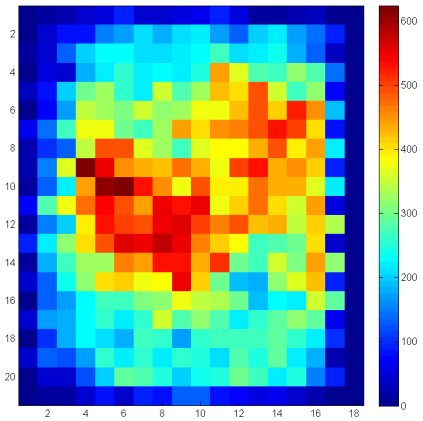

Figure 4: This heat map shows the distribution of all minutiae of the 80 test images.

The attack then works as follows: the adversary sends a single minutia (chosen according to the above restrictions) to the oracle and keeps the minutia if the returned score is one (which is determined with the side-channel leakage). Otherwise, the minutia is exchanged for a new one until the score reaches one. After a matched minutia is found, the algorithm keeps the correct minutia and adds a second. This

step is repeated until the score is equal to the acceptance rate, or all possible minutiae are tested.

Algorithm 3.3: SINGLE MATCHING ATTACK

```
input  : Gallery image
output: Fitting probe image
Initialize: Set minutiae set oldprobe to empty set and
oldscore to 0
begin
    for m = 1 to max possible minutiae do
        for t = 1 to max possible angles do
            probe = oldprobe
            addMinutia(probe,m,t)
            score = oracle(probe,gallery)
            if score > oldscore then
                oldprobe = probe
                oldscore = score
            end
        end
    end
    return probe
end
```

When we add the general Hough transformation, the values after the transformation changes often due to the small number of minutiae. A trick can be used to make sure that the transformation remains stable: If the first matched minutia is found, a second one is added two pixels aside. These two minutiae will have approximately the same distance and difference of angles to one minutia of the gallery image. The transformation will then remain stable even with more minutiae added.

3.2.2 Guessing multiple minutiae at the same time

Next, we assume that the adversary is able to access more fine-grained information from the side-channel analysis. Here we assume the *exact branches model*, i.e., he can identify the exact pairs of minutiae that are matched.

First, in the absence of the Hough transformation, it is sufficient to add more than one minutia at a time. Specifically, we add 100 minutiae at once, which is only slightly more than the average number of minutiae per fingerprint that we see in our database. From the exact branches model, we can directly determine which minutiae matched, replace the incorrect minutiae, and repeat this process until we find a sufficiently large number of matches.

If we consider the algorithm including the Hough transformation, we keep the five first false minutiae in addition. This modification usually ensures that the values for the translation and rotation do not change.

3.3 Results

The following results are based on the 80 fingerprints of the database 3, set B used before. The results are summarized in Table 2.

For the single matching attack without the Hough transform, the average number of requests was 443, with a success rate of 100%, i.e., all fingerprints that we found had a score higher than 22. If the matching algorithm uses the Hough transform, then the information the attacker can extract is slightly less reliable, resulting in an increase of requests made

Algorithm 3.4: MULTIPLE MATCHING ATTACK

```
input  : Gallery image
output: Fitting probe image
Initialize: Set minutiae set probe to NULL and
oldscore to 0
begin
    for m = 1 to 100 do
        addMinutia(probe,m,0)
    end
    currentMinutia=101
    while currentMinutiae < max possible minutiae
    do
        (branches[ ][ ], score) = oracle(probe,gallery)
        if score > acceptancethreshold then
            break
        end
        for every wrong minutia do
            increase angle t
            if t ≥ 360 then
                removeMinutia(wrong minutia)
                addMinutia(probe,currentMinutiae,0)
                currentMinutia++
            end
        end
    end
    return probe
end
```

(1028 requests), and a slightly worse success rate of 97.5%. For one fingerprint, the initial step that tries to "lock" the transformation failed; for another fingeprint, the algorithm did not find enough minutiae.

As expected, the attack matching multiple minutiae performs much better. Without the Hough transform, we achieve a 100% success rate with 6 requests on average, and with the Hough transform, a success rate of 80% for 20 requests. The number of requests is lower by a factor of approximately 50 to 70, due to the much higher number of minutiae tested at once.

	Average requests	Success rate
Single matching attack		
- without Hough	443	100 %
- with Hough	1028	97.5 %
Multiple matching attack		
- without Hough	6	100 %
- with Hough	20	80 %

Table 2: Results of the attacks against the simple matching algorithm.

4. ATTACKING THE BOZORTH3 MATCHING ALGORITHM

The text-book fingerprint matching algorithm from the previous section illustrated the principles of our attack, but is of limited use in practice. In the following, we consider

the Bozorth3 algorithm [25], which is used in the `libfprint` library [16] and is publicly available.

4.1 Algorithm description

The Bozorth3 algorithm operates in three phases: First, it builds a table of pairs of minutiae (the *intra-fingerprint comparison table*). Second, it builds a table of matching pairs for two different fingerprints (the *inter-fingerprint compatibility table*). Third, it computes a score by *traversing the Inter-Fingerprint Compatibility Table*.

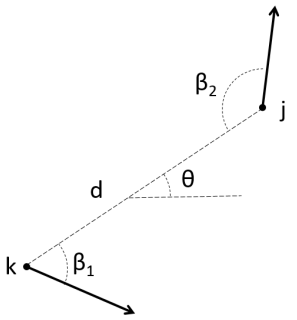

Figure 5: Intra-fingerprint minutiae comparison.

Step 1: Construct intra-fingerprint comparison tables.

In a first step, intra-fingerprint comparison tables are constructed for each fingerprint individually. The entries of the table consist of relative measurements between all pairs of minutiae for a fingerprint; a pseudo-code representation is given in Algorithm 4.1. More precisely, for two minutiae j and k, it computes the following quantities (cf. Figure 5): the distance d_{kj} between two minutiae k and j; the angles β_1 and β_2 calculated relative to the connection line between k and j; and the angle between the horizontal and the connecting line θ. Note that apart from θ, all values are translation and rotation-invariant. Also, for efficiency reasons, only pairs with a distance less than 125 are considered.

Algorithm 4.1: CONSTRUCT INTRA-FINGERPRINT TABLES

input : Minutiae set with M minutiae
output: Intra-fingerprint table

begin
  for $k = 1$ **to** $M - 1$ **do**
    for $j = 2$ **to** M **do**
      $d_{kj} = Distance\,(k, j)$
      if $d_{kj} \leq 125$ **then**
        $\beta_1 = \min(\beta_k, \beta_j)$
        $\beta_2 = \max(\beta_k, \beta_j)$
        $\theta_{kj} =$ Rotation of $\overrightarrow{kj}$
        Create *table* entry $(d_{kj}, \beta_1, \beta_2, k, j, \theta_{kj})$
      end
    end
  end
  return *table*
end

Step 2: Construct inter-fingerprint compatibility table.

In a second step, the generated tables for a probe image and a gallery image are compared and an inter-fingerprint compatibility table is generated, indicating pairs of pairs that are "compatible", i.e., similar.

Let P_m be the m-th entry of the table of the probe table and G_n the n-th entry of the gallery table. P_m and G_n are said to be compatible iff the following conditions are met:

1. The difference of their distances is smaller than a threshold
$$|d(P_m) - d(G_n)| \leq t_d.$$
2. Similarly, the differences in the angles are smaller than a threshold
$$|\beta_1(P_m) - \beta_1(G_n)| \leq t_\beta$$
$$|\beta_2(P_m) - \beta_2(G_n)| \leq t_\beta$$

If these conditions are fulfilled, P_m and G_n represent a possible matching and an entry in the inter-fingerprint table is created. This entry consists of the difference of $\theta(P_m)$ and $\theta(G_n)$, which represents the relative rotation between those two pairs, and the four involved minutiae $k(P_m)$, $j(P_m)$, $k(G_n)$, $j(G_n)$. This algorithm is given in Algorithm 4.2.

Algorithm 4.2: CONSTRUCT INTER-FINGERPRINT TABLES

input : Intra-fingerprint tables with entries $\{P_m\}_{m=1}^{M}$ and $\{G_n\}_{n=1}^{N}$
output: Inter-fingerprint table

begin
  for $h = 1$ **to** M **do**
    for $i = 1$ **to** N **do**
      $\Delta_d = Distance\,(d(P_m), d(G_n))$
      $\Delta_{\beta_1} = Distance\,(\beta_1(P_m), \beta_1(G_n))$
      $\Delta_{\beta_2} = Distance\,(\beta_2(P_m), \beta_2(G_n))$
      if $\Delta_d < T_d$ **and** $\Delta_{\beta_1} < T_\beta$ **and** $\Delta_{\beta_1} < T_\beta$ **then**
        $\Delta_\theta = Distance\,(\theta(P_m), \theta(G_n))$
        Create *table* entry $(\Delta_\theta, k(P_m), j(P_m), k(G_n), j(G_n))$
      end
    end
  end
  return *table*
end

Step 3: Traversing the table.

In a final step, a similarity score is computed from the compatibility table. The algorithm searches for clusters of matching pairs from the compatibility table. The size of the biggest matching cluster gives the similarity score. The inner workings of this algorithm are not documented. The details of this process are not relevant for the remainder of this work, and thus we omit the description.

4.2 The attack

We consider an attack that targets the construction of the inter-fingerprint compatibility table. On a high level, the attack has two main steps (cf. Algorithm 4.3): First, we

calculate sets of minutiae with a high score, and second, we combine several such sets for reaching even higher scores. This is repeated until the score is high enough or no more sets are found.

Algorithm 4.3: ATTACK ON BOZORTH3

```
input  : Gallery image
output: Fitting probe image
begin
    probe = FindSet()
    while score(probe) < acceptance threshold do
        set2 = FindSet()
        probe = MergeSets(probe,set2)
```

4.2.1 Finding a set of matching minutiae

In this first step, we try to find a small set of minutiae with a high score, where the score can be approximated by the number of "hits" in the intra-fingerprint table. The maximum score with n minutiae is $\frac{n\cdot(n-1)}{2}$, i.e., the number of edges in a complete graph with n vertices. This is achieved if all minutiae are arranged identical to a subset in the gallery image (within the error tolerance), and the pairwise distance of all minutiae is smaller than the threshold 125 (as otherwise they do not show up in the intra-fingerprint table).

We start by creating two minutiae $(0, 0, 0)$ and $(0, \Delta, 0)$ with a given distance Δ. Fifty copies of the two are created with different angles and shifted by 126 pixel, which is exactly one more than the maximum distance for the intra-fingerprint table. This is done to ensure that the oracle will provide only links with the distance Δ. Also, it assures that there are enough minutiae for Bozorth3 to start, and it also decreases the number of oracle requests, as different minutiae are tested at the same time. The algorithm needs at least ten minutiae to work.

If a pair was found, the involved minutiae are saved as the first two hits. Since Bozorth3 is translation-invariant, the matching minutiae can be moved to $(0, 0)$ and $(0, \Delta)$.

Next, we add the third and fourth minutia one by one. Given a distance d and an angle t, the position of the next minutia is calculated relative to the first minutia. If the position of the first minutia is (x_0, y_0), the new minutiae is added at $\left(x_0 + \sin\left(\frac{t\cdot\Pi}{180}\right)\cdot d, y_0 + \cos\left(\frac{t\cdot\Pi}{180}\right)\cdot d\right)$.

As done before, copies of the available minutiae are created with different positions and another angle for the current minutia. The number of copies is chosen in a way that the total number is less than 100. Since the copied minutiae only differ in the angle of one minutia, we focus on the original ones. Analyzing the results of the oracle request, we first check if the score equals $\frac{n\cdot(n-1)}{2}$ for $n = 3$ and $n = 4$, respectively. If this is true, one of the copies added is a fitting minutia. We can check which one matched by looking at the branches in the oracle. The matching minutia is the one that has entries in the inter-fingerprint table with every remaining minutia. The same technique is used for more than four minutiae. But instead of adding one minutia at a time, we add twelve, which increases the chance that one of them fits. The advantage is that this will lower the number of oracle requests.

After ten fitting minutiae have been added or if no fitting minutia is found, the set is returned. With ten proper minutiae the score of the set is 45, which is higher than our used acceptance threshold. However, usually the sets are much smaller than that. The following Algorithm 4.4 computes a set with as much minutiae as possible.

Algorithm 4.4: FINDSET

```
input  : Gallery image and startdistance
output: Set and the score of the set
Initialize: Set oldscore to 1 and jnew to 25
begin
    for i = startdistance to 125 do
        set = add2Minutiae(i)
        (branches[ ][ ], score) = oracle(set,gallery)
        if branches != NULL then
            set = findCorrectTwo(set)
        end
    end
    for i = 3 to 4 do
        for j = jnew to 125 do
            for k = 0 to 360 do
                set = addMinutiae(i,j,k)
                (branches[ ][ ], score) =
                oracle(set,gallery)
                if score != oldscore + i − 1 then
                    continue
                end
                ...
            end
        end
    end
    return set
end
```

4.2.2 Merging sets

We observed that the above routine does not always result in a cluster of minutiae with score high enough to pass verification. The most likely reason is that we only can observe if two entries in the intra-fingerprint tables are similar within the error bounds, and these errors add up and make combining multiple pairs hard. Thus, we perform a second step of combining multiple clusters to further increase the score. This is done by combining both sets with different translations and rotations of the added set. Ideally, we would expect the score of the combined set of minutiae to be greater than the sum of both individual scores, but this is not always achieved, probably due to the inaccuracies adding up as explained before.

4.3 Results

The attack was performed on the same 80 images of the Fingerprint Verification Challenge 2004 used before. We report the performance of our attack for different target matching scores, i.e., similarity scores as reported by the Bozorth3 algorithm. (i) We consider a threshold of 40 as an (unrealistic) upper-bound, as we reached an FRR of 22.5% on our dataset, which is unacceptable for most use cases. (ii) For a threshold of 26, we have an FAR of 0% on our dataset, at a FRR of 10%; and (iii) 16 is the threshold that leads to equal error rate (EER).

Table 3 shows the achieved results with different target

Algorithm 4.5: MERGESETS

input : Gallery image and two minutiae sets
output: Set and the score of the set

```
1  Initialize: Set maxscore to 1
2  begin
3      for i = 90 downto −90 do
4          for j = 90 downto −90 do
5              set2 = old_set2
6              set2 = shift(set2,i,j)
7              for k = 0 to 3 do
8                  set = combine(set1,set2)
9                  (branches[ ][ ], score) =
                   oracle(set,gallery)
10                 if score < max(n, m) + min(n,m)/2 + 1
                   then
11                     return set
12                 end
13                 else if score > maxscore then
14                     maxset = set
15                     maxscore = score
16                 end
17                 set2 = rotate90(set2)
18             end
19         end
20     end
21     return maxset
22 end
```

scores. A score of 16 or more is reached in 71.25 % of the cases. With a score in this area, the attack can be successful as the score is chosen with regard to the EER. One third of the performed attacks reached a score of 26 or higher, and at least 11.25 % passed a score of 40. For the attack on the Bozorth3 algorithm, on average 61278 oracle requests were made to recover one fingerprint.

Target score	Successful attacks	Success rate
40	9	11.25 %
26	27	33.75 %
16	57	71.25 %

Table 3: Results of the attack against the Bozorth3 algorithm.

5. PRACTICAL RESULTS: ATTACKING A MICROCONTROLLER IMPLEMENTATION

In this section, we apply the attacks theoretically developed in the previous sections to real implementations running on a 32-bit ARM microcontroller. To this end, we investigate which side-channel models (introduced in Section 2.1) appropriately describe the leakage of the studied implementation, and evaluate the practical applicability of our attacks.

Please note that these experiment only constitute a starting point for future work, and further refinement and evaluation of these techniques is needed.

5.1 Setup

We implemented both the simple matching algorithm of Section 3 and the Bozorth3 algorithm (Section 4) on the STM32F407VGT6, a 32-bit ARM Cortex M4 processor comprising a floating point unit, 1 MB of flash and 192 KB of RAM; with the CPU running at 168 MHz [22]. The processor is integrated on a STM32F4 Discovery board, which provides the power supply, access to the IO pins, and USB ports for programming and communication.

We measured the electro-magnetic (EM) emanation using a near-field probe placed approximately in the middle of the microcontroller, as shown in Figure 6. This position was experimentally found to lead to the maximum signal amplitude. The side-channel signal was then digitized using a Picoscope 6000 series digital oscilloscope. The trigger signal (indicating the start of the computation) to start the data acquisition was generated by the microcontroller.

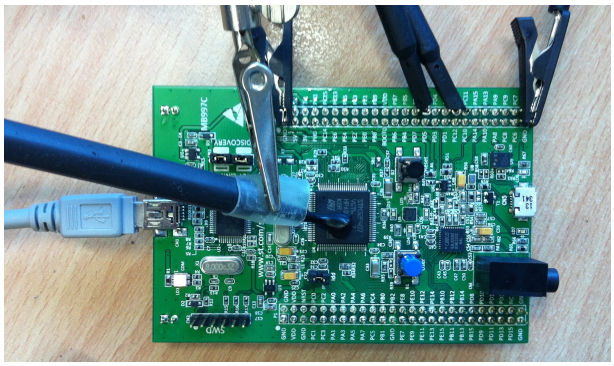

Figure 6: Measurement setup for side-channel analysis of matching algorithms running on the STM32F407. Left / center: EM probe on microcontroller. Top: trigger signal.

5.2 Results for the simple matching algorithm

Our `C` code for the simple matching algorithm (Alg. 3.2) runs without changes on the microcontroller. The general Hough transformation (Alg. 3.1), however, required a significant amount of RAM and was therefore not implemented.

Figure 7 depicts several iterations of the inner loop ($j = 1 \dots N$) in Alg. 3.2, recorded at a sample rate of 1.25 GHz. The darker areas belong to single iterations of the actual code in the inner loop, while the lighter parts represent the increment of j, the loop control logic, and the jump back to the beginning of the loop.

In Figure 7, the fifth iteration is significantly longer than the others. This is due to the fact that in this loop iteration, the inner if-condition is fulfilled, which leads to the execution of lines 6–9 in Alg. 3.2. Hence, for the first match, we can observe the total number $n_1 = (i_1 - 1) \cdot N + j_1$ of iterations until a matching occurs. However, this does not provide the exact position (i_1, j_1) yet:

When we observe that a matching occurred after n_1 repetitions of the code in the inner loop, different matchings are possible. For example, for $n_1 = 10$, a possible matching is the first and the tenth minutia. It could also be a matching between the second and the third minutia, if the image for the inner loop consists of only $N = 7$ minutiae ($n_0 = 1 \cdot 7 + 3$). The same holds for subsequent iterations, where only obtain

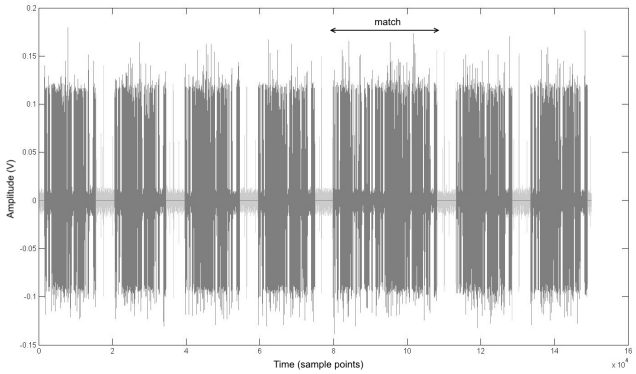

Figure 7: EM trace encompassing several inner loop iterations of the simple matching algorithm. Sample rate 1.25 GHz.

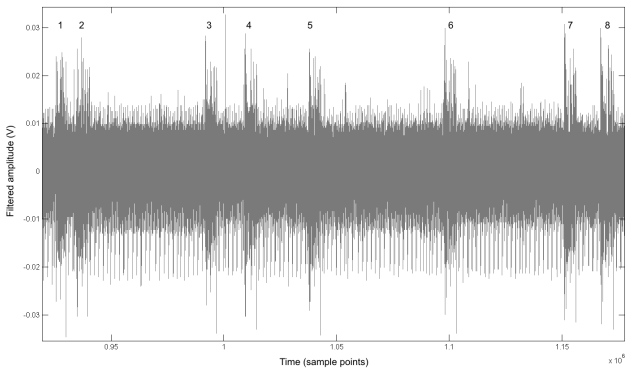

Figure 8: EM trace of the creation of the inter-fingerprint table, encompassing the writing of eight table entries to RAM. Sample rate 625 MHz.

the total number of executions n_1, n_2, ... following the previous match.

Note that the number of minutiae in the probe image is known. Hence, in the case that the inner loop iterates over the minutiae in the probe image, we know N and can compute i and j. In the opposite case, if the inner loop is over the minutiae in the gallery image, we select a random probe image with $M = 1$ minutia and count the number of iterations of the inner loop to obtain N. If we detect that the inner loop found a match (long pattern in side-channel trace), this process is repeated with a different probe image, until no match occurs and we get the correct value for N.

Hence, knowing the pairs (i, j), we are in the exact branch model of Section 7 and can directly apply the attack of Section 3.

5.3 Results for Bozorth3

We implemented Step 1 (intra-fingerprint compatibility table) and Step 2 (inter-fingerprint compatibility table) of the Bozorth3 algorithm on the microcontroller, based on the reference implementation of NIST [18]. Where possible, we left the original `C` code unchanged—we only made the following changes: As explained in Section 4 the attack targets the creation of the inter-fingerprint table, thus, we left out Step 3. In addition, for the purposes of our experiments, we reduced the size of the arrays in the algorithm to fit the available RAM. In the original algorithm, the arrays are preallocated with 20000 entries; for testing, we reduced the size to 1000.

Recall that the assumption for the oracle used in the attack is that one can observe exact branches when an entry is saved to the inter-fingerprint table. Indeed, the creation of such an entry can be clearly seen in the side-channel trace: Figure 8 depicts the respective EM trace (sample rate 625 MHz), exhibiting several high-amplitude patterns (numbered in Figure 8) that each correspond to the creation of a new entry in the inter-fingerprint table.

The corresponding (simplified) `C` code implementing the targeted Step 2 in the Bozorth3 algorithm is shown in Listing 1. When the control flow reaches line 27, a table entry is created and the respective pattern occurs in the trace. We refer to the case of the full execution of the loop body as a "full iteration", while we call the opposite case (loop left before creating a table entry "stopped iteration").

Listing 1: Excerpt of C code for the inter-fingerprint table creation, parts omitted for clarity.

```
1  // ...
2  for (k = 1; k < probe_ptrlist_len; k++)
3  {
4    // ...
5    for (j = st; j <= gallery_ptrlist_len; j++)
6    {
7      // ...
8      if (SQUARED(dz) > SQUARED(fi))
9      {
10         if ( dz < 0 ) {
11           st = j + 1;
12           continue;
13         } else
14           break;
15     }
16
17     for (i = 1; i < 3; i++) {
18       // ...
19       if (dz_squared > TXS &&
20           dz_squared < CTXS)
21         break;
22     }
23
24     if ( i < 3 )
25       continue;
26
27     // code for table entry follows
28   }
29 }
```

We can measure the time difference between two patterns and thus obtain the number of stopped iterations—in the present case, a stopped iteration had a length of approximately 2,100 sample points, so the distance between two patterns divided by this length gives the number of stopped iterations betweens two patterns. Hence, we at the very least can work in the approximate branches model (Section 10).

However, note that with this straightforward application of SPA, we do not reach the exact branches model. First, there are multiple ways for the inner loop to be aborted in the case of a stopped iteration, namely `continue` (line 12 and line 25) and `break` (line 14). Note that the `break` on line 21 triggers the second `continue`, but does not directly

leave the loop. Secondly, the start value for the inner loop changes on line 11, further complicating the situation.

In conclusion, the attack of Section 4 cannot be carried out using the information available from this straightforward application of SPA. However, it seems likely that templates can be built for the different possibilities of exiting the loop, for example using techniques as presented in [23, 12]. This would allow to work in the exact branches model and hence apply the attack of Section 4. We leave this aspect for future work.

6. CONCLUSION AND FUTURE WORK

In this paper, we present SPA-based side-channel attacks on two fingerprint matching algorithms, including Bozorth3. After establishing models for potentially relevant leakages, we develop novel attack procedures and show that the attacks can be applied in practice to a microcontroller running the NIST implementation of Bozorth3 and a text-book matching algorithm. Our results are a starting point for future work with respect to the side-channel security of fingerprint matching:

Further studying the side-channel leakage of the Bozorth3 implementation, e.g., using template attacks, is a necessary next step. After that, the presented methods could be applied to real fingerprint readers or match-on-card systems in a (semi-)blackbox scenario. Extending these results to commercial products is a very interesting task, but substantially complicated by the fact that source code and implementation details are unavailable. Furthermore, studying software side channels of PC libraries, for example of `libfprint`, is of interest. Specifically, cache-timing attacks seem to be an promising approach to obtain side-channel leakage in this context.

7. REFERENCES

[1] Fingerprint verification challenge 2004. http://bias.csr.unibo.it/fvc2004/. [Online; accessed 22-July-2016].

[2] O. Aciiçmez, Ç. K. Koç, and J.-P. Seifert. On the power of simple branch prediction analysis. In *2nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS '07, pages 312–320, New York, NY, USA, 2007. ACM.

[3] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In *Cryptographic Hardware and Embedded Systems – CHES'02*, pages 29–45. Springer, 2002.

[4] D. J. Bernstein. Cache-timing attacks on AES. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[5] M. Berthier, Y. Bocktaels, J. Bringer, H. Chabanne, T. Chouta, J.-L. Danger, M. Favre, and T. Graba. Studying leakages on an embedded biometric system using side channel analysis. In *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop – COSADE '14*, pages 281–298. Springer, 2014.

[6] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems – CHES'04*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.

[7] CCC. Chaos Computer Club breaks Apple TouchID. https://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid, Sept. 2013. [Online; accessed 27-July-2016].

[8] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *Cryptographic Hardware and Embedded Systems – CHES'02*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[9] T. Chouta, J. L. Danger, L. Sauvage, and T. Graba. A Small and High-Performance Coprocessor for Fingerprint Match-on-Card. In *Digital System Design – DSD'12*, pages 915–922, Sept. 2012.

[10] T. Chouta, T. Graba, J.-l. Danger, J. Bringer, M. Berthier, Y. Bocktaels, M. Favre, and H. Chabanne. Side Channel Analysis on an Embedded Hardware Fingerprint Biometric Comparator & Low Cost Countermeasures. In *HASP'14*, pages 6:1–6:6, New York, NY, USA, 2014. ACM.

[11] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In *Advances in Cryptology – CRYPTO'08*, volume 5157 of *LNCS*, pages 203–220. Springer, 2008.

[12] T. Eisenbarth, C. Paar, and B. Weghenkel. Building a Side Channel Based Disassembler. *Transactions on Computational Science X - Special Issue on Security in Computing, Part I*, 6340:78–99, 2010.

[13] A. K. Jain, A. A. Ross, and K. Nandakumar. *Introduction to Biometrics*. Springer, 2011.

[14] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology – CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[15] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology – CRYPTO'99*, LNCS, pages 388–397. Springer, 1999.

[16] libfprint project. Online at https://www.freedesktop.org/wiki/Software/fprint/libfprint/. [Online; accessed 28-July-2016].

[17] libfprint project. Security notes. https://www.freedesktop.org/wiki/Software/fprint/Security_notes/. [Online; accessed 28-July-2016].

[18] NIST. NIST Biometric Image Software. http://www.nist.gov/itl/iad/ig/nbis.cfm. [Online; accessed 22-July-2016].

[19] D. Oswald and C. Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In *Cryptographic Hardware and Embedded Systems – CHES'11*, volume 6917 of *LNCS*, pages 207–222. Springer, 2011.

[20] D. Oswald, B. Richter, and C. Paar. Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. In *16th International Symposium on Research in Attacks, Intrusions and Defenses – RAID'13*, volume 8145 of *LNCS*, pages 202–222. Springer, 2013.

[21] Starbug, R. Hänsch, and T. Fiebig. Ich sehe, also bin ich ... du. presentation at 31C3, https://events.ccc.de/congress/2014/Fahrplan/system/attachments/2573/original/congress2014.pdf, 2014.

[22] STMicroelectronics. STM32F407xx Data Sheet.

http://www.st.com/resource/en/datasheet/stm32f407vg.pdf. [Online; accessed 28-July-2016].

[23] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar. SCANDALee: A Side-ChANnel-based DisAssembLer using Local Electromagnetic Emanations. In *Design, Automation and Test in Europe – DATE'15*, 2015.

[24] U. Uludag and A. K. Jain. Attacks on Biometric Systems: A Case Study in Fingerprints. In *Security, Steganography and Watermarking of Multimedia Contents – SPIE-EI'04*, pages 622–633, 2004.

[25] C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. M. McCabe, S. Janet, and K. Ko. User's Guide to Export Controlled Distribution of NIST Biometric Image Software (NBIS-EC). Online at https://www.nist.gov/customcf/get_pdf.cfm?pub_id=51097, 2007.

[26] Y. W. Yun. An Introduction to Biometric Match-On-Card. https://www.e-xpertsolutions.com/images/pdf/moc/3_BiometricMOC.pdf, 2005. [Online; accessed 28-July-2016].

[27] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-VM side channels and their use to extract private keys. In *ACM conference on computer and communications security*, pages 305–316. ACM, 2012.