Protection of Fingerprint Data with the Glass Maze Algorithm

Markus Springer

Department of Computer Science
Hochschule Darmstadt
Haardtring 100
64295 Darmstadt
markus.springer@stud.h-da.de

Abstract: This work proposes an implementation of the glass maze algorithm proposed by Carlo A. Trugenberger and presents a short summary of its theoretical basis. The implementation is roughly tested with synthetic fingerprints and the experimental results and findings during the testing phase regarding security issues and performance are discussed. It could be seen, that even though from a theoretical standpoint the system seems to be stable against brute force attacks, under certain circumstances the system may still be vulnerable to brute force attacks.

1 Introduction

With the help of cryptography it is possible to provide confidentiality, non-repudiation, integrity and authenticity. One of the big problems of traditional cryptography is the size of its keys. The size of the keys limits the possible applications of cryptographic functions. While on the one hand, longer keys often guarantee for a higher level of security and a wider search space for brute force attacks, they are also in most cases not usable for memorization by a human. This problem gets intensified by the fact that most service providers use different cryptographic techniques and most of the time issue their own keys which makes it even harder for the user to remember the keys. One way around this problem is the idea of biometric cryptosystems. Biometric cryptosystems combine biometric and cryptographic systems and methods in order to make them more human usable. Biometric features, depending on the type that is used, can under normal circumstances neither be forgotten nor lost. The biometric trait is used as the key of the system. These so called templates of the user need to be saved in order to use them. This fact leads to a new problem: How can be made sure, that the biometric template data can not be stolen or abused in any way? The protection of these templates is not only a privacy issue but also an issue of security. In order to solve this problem the fuzzy fingerprint vault was developed [CKL03]. The fuzzy fingerprint vault is used to protect the minutiae data of a fingerprint. The security of the fuzzy vault is based on the difficulty of polynomial reconstruction [Tru11]. While the fuzzy fingerprint vault helps in decreasing the risk of stolen biometric template data and reduces privacy issues, it was shown to be vulnerable to brute force [Mih07] as well as cross-matching attacks [SB07]. Due to these problems another scheme is needed in order to replace the fingerprint vault. Because of this, Carlo A. Trugenberger proposed the idea of the glass maze algorithm as a new way for template protection.

2 The Hopfield Model

The Hopfield model is a neural network which is based on the Spin Glass Model. The Hopfield model was designed to model the associative memory of the human brain. It is able to recognize patterns which it previously learned [AGS85]. A mechanic that can be used for key retrieval as proposed by Trugenberger in [Tru11]. The Hopfield model consists of N binary neurons which may take one of two different states: firing (1) and resting (-1). The states of all neurons combined is considered the current state of the system. The neurons are connected by symmetric synapses with coupling strength $w_{ij} = w_{ij}$ and $w_{ii} = 0$ which makes them interact with each other. Depending on the sign of the coupling strength it can either be interpreted as inhibitory (< 0) or exhibitory (> 0). The process of recognizing a pattern is done via evolution of the network state. The dynamic evolution of the current state by random sequential updating of neurons is defined as follows:

$$s_i(t+1) = sign[h_i(t)] \tag{1}$$

$$h_i(t) = \sum_{i \neq j} w_{ij} s_j(t) + \theta_i \tag{2}$$

h_i is called the local magnetization of neuron s_i. The value θ_i is a threshold value that can be used to control when a neuron should change it's state. Trugenberger proposes this factor should be set to 0 [Tru11]. The synaptic coupling strength w_{ij} is chosen according to the Hebbian learning rule [Mov90] which is described by the following formula:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1...p} \sigma_i^\mu \sigma_j^\mu \tag{3}$$

The σ_i^μ, $\mu = 1...p$ are binary patterns that should be memorized by the neural network. The associative memory is defined as a dynamical memory that, upon preparing the network in an initial state s_i^0, retrieves the stored pattern σ_i^λ that most closely resembles the presented pattern s_i^0. Resemblance is defined by minimizing the Hamming distance between both of these configurations. The described mechanic encodes all the information that is stored in the neural network into the synaptic coupling strengths [Tru11]. As stated before, the neural network will try to retrieve patterns by dynamically updating the network state with the previously defined functions. With these neuron updates the neural network uses a hill climbing dynamic to retrieve patterns. The patterns which were stored in the neural network correspond to local minima of the system. This means that the stored patterns are attractors for the dynamic network updates which in turn means that the system will evolve till it overlaps with the closest stored pattern. At that point the state will not change anymore. The behavior of the Hopfield model depends on the so called loading factor $a = p/N$ which is the ratio between the stored patterns and the number of neurons of the system. By analyzing this factor in the thermodynamic limit $p \to \infty, N \to \infty$ it is possible to find three distinct sectors which are further described in [Tru11]:

$a < 0.051$: In this sector the system is in a ferromagnetic phase. This means that there are global minima that correspond to all stored pattern. This means that an exhaustive search for all stored patterns may be organized.

$a > 0.138$: The system is in a state of chaos where all retrieval capabilities are lost and no information can be retrieved successfully.

$0.051 < a < 0.138$: In this sector the system is in a mixed spin and ferromagnetic phase. This is the ideal sector for hiding and retrieving the key because the number of minima is increased exponentially and network states that are close to a stored pattern will converge to the closest (in Hamming distance) stored pattern. If the state is sufficiently different from the stored patterns it will converge to a minimum that was not stored.

3 The Glass Maze Algorithm

The enrollment of a fingerprint consists of a number of non-trivial steps. The first of these steps is the so called quantization. In this step the fingerprint is provided in the form of M pixel coordinates (x_i, y_i), where M corresponds to the amount of minutiae of the fingerprint. These coordinates are now used to generate a system configuration for the Hopfield model to remember. This is done by dividing the image of the fingerprint into N squares of almost equal size. Every one of these N squares corresponds to a neuron. A neuron can take two states: -1 if there is no minutiae in the corresponding square and 1 if at least one minutiae is in the square. The configuration that is generated out of this is called σ^{fp}. The next step is to generate the key configuration σ^{key}. The key is created by randomly flipping k bits of σ^{fp}. This means that σ^{key} and σ^{fp} have a Hamming distance of k. This way the fingerprint of the user is never directly stored in the system. After generating the key the next step is to generate $p-1$ random bit patterns. These patterns are then used to bind the key. The key binding mechanism is described by the following function:

$$w_{ij} = \frac{1}{N} \sum_{\mu=1...p} \sigma_i^{\mu} \sigma_j^{\mu} \tag{4}$$

$$\sigma_i^1 = \sigma_i^{key} \tag{5}$$

$$\sigma_i^{\mu} \epsilon \{-1, 1\}^N, \mu = 2...p \tag{6}$$

The number of patterns must be chosen according to loading factor $0.051 < a < 0.138$ as it was described in section 2. Trugenberger proposes $a = 0.1$ as a good choice. In the last step the it must be assured that the fingerprint can actually retrieve the generated key. This is done by simply trying to retrieve it via key retrieval. If key retrieval fails, the key must be generated by lowering the value k and trying it again. The key retrieval corresponds to the dynamic network updates of the Hopfield model. The key retrieval is achieved by first quantizing the fingerprint and the resulting configuration σ^{fp} is then chosen as the systems initial state s_i^0. The state is then dynamically evolved as described in 2 until the system reaches a minimum. If the provided fingerprint was part of the keys basin of attraction, the key will be retrieved successfully [Tru11].

4 Implementation

The implementation of the glass maze algorithm was mostly done the way it was proposed by Trugenberger in [Tru11]. Some parts were altered and will be discussed in further detail. The enrollment process was split up into five different non-trivial steps:

1. Quantization of the fingerprint
2. Key generation
3. Random pattern generation
4. Coefficient matrix calculation
5. Enrollment verification

The quantization (1) of a fingerprint is the process of creating σ^{fp} from raw fingerprint data. The fingerprint data is provided in the form of a list of minutiae coordinate pairs M. Coordinate pairs (x, y) represent minutiae. Other necessary parameters are the dimension D of the fingerprint scanner image, as well as the number of neurons N. During quantization the minutiae are mapped directly to the neurons which are represented by a vector S of size N. The overall number of pixels on one image of the scanner can be calculated with the given dimension parameter which are interpreted as $width$ and $height$. These two are used to calculate the number of pixels the image consists of. With the help of the given number of neurons N we can compute the amount of pixels that represent the same neuron field. All pixels together are then interpreted as a long string of points. The minutiae are placed on this string and afterwards are mapped onto their corresponding field in the neuron vector via the following formulas:

$$p_i = y_i \cdot width + x_i, \quad i = 1...M \tag{7}$$

$$S_i = \lceil (\frac{N \cdot (p_i + 1)}{width \cdot height}) \rceil, \quad i = 1...M \tag{8}$$

The described method is not the best way to quantize a fingerprint, but it is one of the easier and faster ways to implement it. This part of the implementation is probably the part that can be improved the most. The big problem with this method of quantization is that it does not make use of the fingerprints properties. There are positions that are a lot more likely to contain minutiae as others. The edges of an image for example will be a lot less likely to contain minutiae than the rest of the image. This makes it easier for an attacker since he may exclude those areas in brute force attacks which decreases the overall search space. The main decision for this approach was the ease of implementation and flexibility for different image sizes.

The key generation (2) depends on the quantized fingerprint σ^{fp} and the number of flipped bits k. The flipping of bits is done in rounds. In each round a value between 1 and N is chosen at random and used as the index of the neuron that will be flipped. The algorithm repeats the rounds until the Hamming distance between the currently derived key σ^{key} and σ^{fp} is k.

The random pattern generation (3) takes the number of neurons N and the loading factor a. It uses a to calculate the number of random patterns $p - 1$ and generates a $N \times (p - 1)$ matrix R of random patterns. Each field of these patterns represents a neuron as it was described in section 2. A random pattern is generated by creating a vector of N random values between 0 and 1. The values in this vector are then rounded and mapped in a way that values ≥ 0.5 are mapped to 1 and values < 0.5 are mapped to -1. Each of these vectors represents one column of the matrix.

The coefficient matrix calculation (4) was not directly mentioned in [Tru11] but in [Tru12]. This step is done in order to hide the stored patterns. This is achieved by applying the Hebbian learning rule, which was mentioned in section 2, to the previously generated random patterns and the key. The function takes the parameters N, σ^{key} as well as the previously created matrix R and outputs a $N \times p$ matrix W containing the coupling strengths.

In the last step the enrollment verification (5) is done to ensure that the provided fingerprint is able to retrieve the previously generated key. This is done by calling the key retrieval mechanism with the fingerprint M and the matrix W and check if it returns the key σ^{key}. If it works, the enrollment is finished. If it did not work, k is decreased by one and from step (2) onwards all steps are repeated. The overall output of the enrollment is the matrix W which contains the key as well as the random patterns.

The key retrieval mechanism was mostly implemented the way it was proposed by Trugenberger. The key retrieval takes the parameters W, the fingerprint coordinates M, the dimension D of the scanner image as well as the number of neurons N. It starts by quantizing the given fingerprint coordinates M. The quantized fingerprint σ^{fp} is then used as the initial state of the network. The neurons are then evolved as described in 3 until the Hamming distance between the old neuron state s_{i-1} equals the newly evolved neuron state s_i. An important thing to note is, that Trugenberger defines the updating of neurons as random and sequential. In order to ease the implementation, it was decided to update each neuron in strict rotation. Such an update may be considered as a round. After each round it is checked, whether the system has reached a minimum or if further updating is required. If updating was done randomly it would be harder or more error prone to deter-

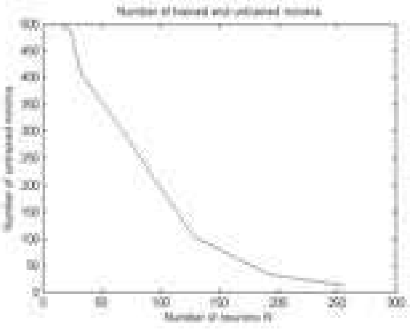


Figure 1: Number of untrained minima

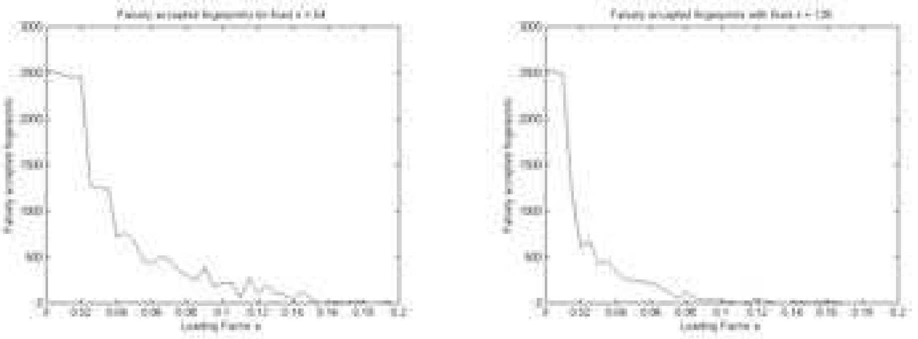
mine if the system has reached a minimum. During testing no significant differences could be found between different rotations, though this would need further testing.

5 Experimental Results

For all experiments described only synthetic fingerprints were used to avoid problems due to alignment and in order to test the algorithms on a more general level. During the implementation and testing phases a first glimpse on possible problems could be examined pretty early on. One problem was the resemblance between quantized fingerprint, key and randomly generated patterns. In relation to the number of neurons the fingerprints and the key resemble each other a lot more then they would with the randomly generated patterns. This would mean that the FAR can be quite high. This stems from the fact that the used random generator most of the time creates a rather uniform pattern while fingerprints often do not tend to have a uniform pattern. The associative memory will relate fingerprints to each other more often then it would with random patterns or any other minimum at all.

Another factor that plays into this can be seen in figure 1. The greater we chose the factor N of neurons, the more likely it is that key retrieval will actually return a previously trained pattern rather then a random minimum. The test was done with 1000 synthetic fingerprints per value of N. The factor a was kept at 0.1 which was proposed by Trugenberger to be close to a good choice. This also makes the hiding of patterns within the coefficient matrix a rather useless trial, because an attacker may brute force and guess most of the saved patterns in minimal time given that N was chosen big enough. A way to circumvent this problem could be in choosing parameter k according to the randomly generated patterns to make the pattern more uniform or by improving the quantization step to generate a more uniform distribution which will make the quantized fingerprint more similar to the saved patterns.

Another experiment was done testing the false acceptance rate for fixed values of N. In these experiments the loading factor a was varied. For each chosen value of a 5000 key retrievals were executed. What can be seen is that for low values of a the system often



(a) N = 64 (b) N = 128

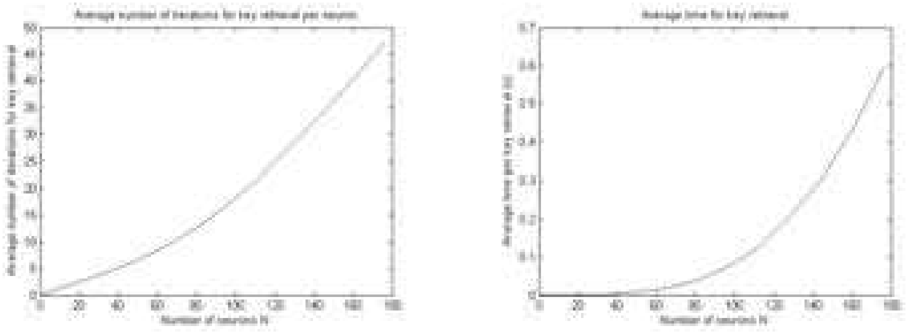
Figure 2: Falsely accepted fingerprints for fixed N and variable a

falsely accepts the presented fingerprint while for rather high values of a the system rarely falsely accepts a fingerprint. What is to note though is, that the higher the value N is chosen, the better the false acceptance rate gets for smaller values of a. Figure 2 displays this asymmetric observation. For $N = 128$ the curve is a lot steeper then for $N = 64$. As Trugenberger described in [Tru11], higher values of a set the system in a state of chaos so that saved patterns are not retrievable. Further testing this effect may be able to loosen the lower bound for higher values of N.

As for performance the system was tested according to the number of iterations the key retrieval mechanism needs to find a minimum, as well as the average execution time of finding a minimum. The results of the experiment can be seen in figure 3. For every chosen value of N 5000 key retrievals were executed. For the testing a simple Intel i5 processor with 2.26Ghz was used. Something to note is, that per 16 additional neurons the average number of iterations is increased by $38 - 40\%$. But while the system is exponential and not linear, it is still growing rather slow. The average execution time on the other hand is growing fast and is also exponential. This seems logical, as the number of computations for one iteration grows with the number of neurons, as well as the number of iterations.

6 Conclusion and Future Work

Overall the proposed idea of Trugenberger seems promising. In order to protect the templates the system is required to make it hard for the attacker to get the stored pattern. The privacy issue may be solved, but the security issues of the system can not be dismissed. A big problems seems to be that, while the number of minima increases with higher numbers of neurons, the actual search space seems to decrease because these minima are scarcer reached. This limits the system to smaller numbers of neurons, which in return reduces the time of its usefulness a lot. Further testing is needed to determine if this problem can be circumvented and check the influence of the loading factor, k and neurons more. Also a



(a) Iterations (b) Time

Figure 3: Performance measurements for time and iterations

better method for quantization should be implemented to increase the overall search space by distributing the minutiae more. Further tests have to be conducted with real minutiae data in order to check for FAR and FRR with real user data and check for the influence of alignment.

References

[AGS85] Daniel J. Amit, Hanoch Gutfreund, and H. Sompolinsky. Spin-glass models of neural networks. *Phys. Rev. A*, 32:1007–1018, Aug 1985.

[CKL03] T. C. Clancy, N Kiyavash, and D. J. Lin. Secure Smartcart-Based Fingerprint Authentication. In *ACM SIGMM Workshop on Biometric Methods and Applications*, pages 45–52, 2003.

[Mih07] P. Mihăilescu. The Fuzzy Vault for Fingerprints is Vulnerable to Brute Force Attack, 2007.

[Mov90] Javier R. Movellan. Contrastive Hebbian Learning in the Continuous Hopfield Model, 1990.

[SB07] W. J. Scheirer and T. F. Boult. Cracking the Fuzzy Vault and Biometric Encryption. In *Biometric Symposium*, pages 1–6, 2007.

[Tru11] Carlo A. Trugenberger. The Glass Maze: Hiding Keys in Spin Glasses. In *BIOSIG*, pages 89–102, 2011.

[Tru12] Carlo A. Trugenberger. Hiding Identities in Spin Glasses. In *Proceedings of the 2012 Eighth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pages 35–40. IEEE Computer Society, 2012.