Dynamic Keystroke Analysis via Genetic Algorithms

Ron Luman II

Hewlett-Packard Company
Cupertino, California 95014
ron.luman@hp.com

Stanford University
Stanford, California 94305
luman@cs.stanford.edu

Abstract

Previous work has established that it is possible to distinguish between multiple users based solely on keystroke timing data. This can be trivially extended to authenticating a single user based on how well a particular set of keystroke timing data matches a previously collected set. The challenge lies in deciding exactly which parts of the timing graph are most important in identifying a user. This is in contrast to the portion of the data which varies from session to session or the data which is roughly equivalent among a set of users. In this paper, we apply the techniques established in the Genetic Algorithms field to dynamically select, on a per-user basis, the weights applied to each piece of the key-hold and inter-keystroke times to obtain a close-to-optimal template which can then be used to make an authentication decision. We then show that the distinguishing capabilities of these dynamically generated templates exceed the abilities of previously established static methods.

keywords: keystroke analysis, genetic algorithms, authentication, identification, machine learning, pattern recognition, classification, interkey times, key hold times, multi-factor.

1 Introduction and Overview

1.1 Authentication Mechanisms

Verifying a person's identity, also known as authentication, is a necessity for many environments. In the financial world, authentication has traditionally taken the form of a handwritten signature. Through a combination of learned behaviors and physiological development, it is extremely difficult to forge a signature without artificial assistance (in the form of a copier, for example). In recent times, there has been an increased emphasis on the use of passwords for identity verification. The problem of assuming an identity has moved from performing some physical act to learning some bit of knowledge. More generally, there are four major categories of authentication mechanisms [JG90]:

tokens or objects: Any identification scheme which requires the possession of a physical object falls into this category. Keys, smart-cards, and ATM cards are all examples of tokens that have been used for identification.

knowledge: This encompasses authentication protocols which rely on a whether or not the target user knows some piece of information. Commonly, this information takes the form of a password, but it might also be the combination to a padlock, the pin to access an account, etc.

actions: Some mechanisms require the user to perform an action and possibly examine some particular behavior or facet of that action. As indicated above, signatures fall within this category.

physiology: A whole field of identification solutions fall within the realm of verifying some physiological aspect. Also known as biometrics, mechanisms such as fingerprint scanning, retinal scanning, face recognition, voice recognition and DNA testing all fall within this category.

A given mechanism might fall into any subset of the above categories, and in practice, many identification schemes do require *multi-factor* authentication. A common example is withdrawing money from an Automated Teller Machine (ATM): one is required to provide a physical object in the form of the ATM card as well demonstrating a knowledge of the appropriate PIN.

One of the disadvantages of multi-factor authentication as it applies to computer systems is that it typically requires additional, possibly specialized, hardware, along with the associated costs that extra hardware brings. In comparison, the use of a password for authentication is relatively cheap as most systems already have a method for data input. The downside of using passwords alone is that users tend to choose passwords which are easily guessed. In a study of passwords chosen by users where there were no constraints put on the choice of password, Robert Morris and Ken Thompson found that of the 3289 passwords gathered, 86% could either be found in a dictionary or were sufficiently short such that they could be discovered in a minimal amount of time [MT79].

1.2 Keystroke Analysis

A possible mechanism for increasing the security of passwords while maintaining the advantage in cost afforded by the lack of need for additional hardware was proposed by Gaines et al. in 1980. They found that the timing patterns which occur in the normal course of typing can be used for identification [GLPS80]. By recording the precise time between each consecutive keystroke (*digraph* latency times), and comparing against previously collected data for the target user, they were able to determine, with a high degree of confidence, whether the typist was actually the target user. It was further determined that certain digraph times were more reliable than other and more efficient in determining identity. Gaines et al. labeled these *core* digraph times.

The challenge faced by Gaines and later researchers in determining identity based on keystroke dynamics lies in reducing the amount of typing that the user must do while at the same time maintaining the "certainty" of the authentication. There are multiple approaches to accomplishing this. The mechanism proposed in this paper is to dynamically create, when a user is first added to the system, a keystroke template for the user. This differs from previous approaches in that the weighting of each digraph is customized on a per-user basis rather than on a system-wide basis.

1.3 Genetic Algorithms

The approach we propose for generating this template is based on a technique first established by John Holland [Hol75]. Genetic algorithms are a form of machine learning and adaptation based on the ideas of evolution in nature. The basic idea goes as follows: The solution to a given problem can generally be encoded in some string representation. Certain of these strings are better than others in the context of the problem. If we start out with a population of random strings and we apply certain genetic operations such as reproduction, crossover, and mutation according to some *fitness* criteria, the hope is that the strings will improve over time.

The fitness function is of critical importance to the success of the genetic algorithm. Since the algorithm itself has no knowledge of the actual meaning of the strings that it is operating on, [1] the fitness function generally dictates which strings are selected for reproduction and therefore will continue on to a later generation. Conversely, the fitness function also dictates which strings are somehow bad and will die off.

The actual mathematics behind the success of genetic algorithms is beyond the scope of this paper, but intuitively, they take advantage of all of the information present in a population of strings (or *genes*), to implicitly create a set of weights. These weights are not explicitly used in any of the operations, but they determine which building blocks (*schemata*) are most likely to be present in later generations [Gol89]. It is this building block approach which allows genetic algorithms to emulate human thought and synthesis.

[1] There are occasionally exceptions to this for situations where it is necessary or desirable to have problem-specific crossover and/or mutation operations. This is commonly the case when only a small subset of total possible strings are valid.

Despite the large domain over which genetic algorithms may be applied, there are certain types of problems where they are especially successful when compared to other methods. In particular, evolutionary algorithms are best suited to problems where the solution is heavily interdependent in a non-linear way. An example provided by Koza is the design of a giraffe [Koz92]. If one were to optimize the features individually, then the giraffe would probably not have survived as it has. After all, the long neck is only optimal in the context of the herbivorous nature of the giraffe, the leafy food environment with little competition, and the long agile tongue. If any of these other features is missing, then the long neck becomes a detriment in that it is more fragile than a short neck. If we were to try to optimize these features individually, we would end up with a less fit individual.

In the rest of this paper, we will further explore this last idea as it applies to the keystroke latency pattern recognition problem. Section 2 details the exact form of the problem that we are analyzing along with the methodology used in collecting the keystroke timing information. Section 3 provides the *tableau* for the genetic algorithm and discusses the fitness function along the operational parameters used in our solution. In section 4 we give the results of our research along with a comparison to statically generated results. We provide a more detailed analysis of the results in section 5. Finally, in section 6, we close and discuss possible areas of future research.

2 Data Collection

2.1 General Strategy

The general keystroke input characteristics, as well as the program that we instrumented to gather the user data is based on a particular use model for authentication. While the suitability of genetic algorithms for analysis of keystroke timing data should not be limited to this particular scenario, [2] we will use it as a starting point for the rest of our analysis. We assume the following:

- There exists some application or operating system with a requirement of user identification.
- This application either resides locally on the client machine or it has direct access to the client terminal/keyboard.
- There exists some setup phase for a particular user where baseline data can be gathered. During this time, the user is prompted to type some amount of text. This phase is synonymous with choosing an initial password on a timeshare system.
- The data gathered in the setup phase is processed, producing an *identification template* which explicitly defines the authentication criteria. This processing need not occur in real-time, and would likely take place at some point after the setup phase, when the system is idle.
- At any point after the processing phase, the user identifies himself/herself by typing some text in response to a prompt. This template established in the previous step is applied to this response, and the authentication is either accepted or rejected.
- Optionally, the incremental user authentication data can be used to further refine the identification template.

2.2 Input Characteristics

There are a number of possible scenarios under which keystroke latency data may be collected and analyzed. For the purposes of this project, we chose to obtain multiple samples of a short, fixed string from each test subject. While shorter strings provide less data and are therefore more difficult to reliably use for authentication, they are also much more practical, in the sense that most users are unwilling to type paragraphs of text just to log in to their application.

[2] In fact, genetic algorithms are even better suited to the analysis of free-form text, due to the inherent difficulty of applying a rigorous mathematical or statistical model to a larger body of text.

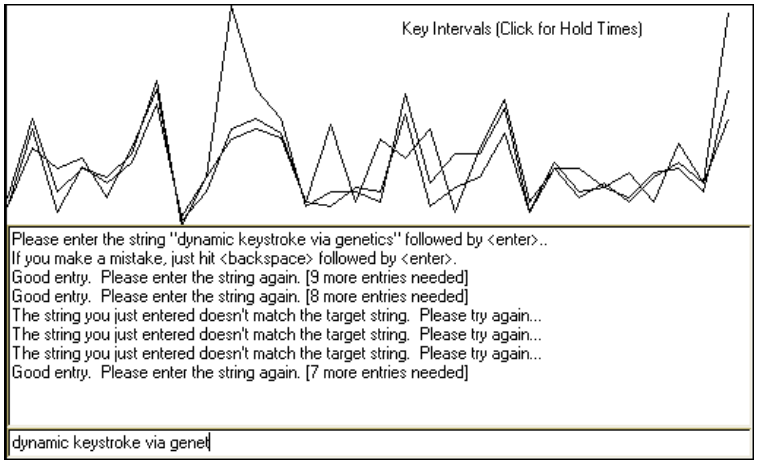

Figure 1: a sample input session

In addition to requiring relatively short authentication tokens, we also used the same four words for each user: *"dynamic keystroke via genetics"*. This differs from authentication schemes where the users type words familiar to them, in that the input timing characteristics are not quite as consistent. While our scheme presents additional difficulty in this respect, it also multiplies the amount of data that is available to be analyzed. Even though we had a limited number of test subjects available (14), we were able to gather sufficient data for analysis.

2.3 Baseline Data Collection

There is little keystroke timing data publicly available, so for the purposes of this experiment it was necessary to instrument a program which prompted the user to type the keywords previously specified and record the results. The program we created took the form of a java applet (figure 1) to allow for remote distribution of the test program while at the same time satisfying the requirement that the actual program is running locally on the target machine. This is necessary to ensure that the keystroke latencies can be recorded with sufficient precision. [3] The actual data collection itself took place on a variety of client machines and web browsers.

Each user was asked to enter the target string ten times. Any entries which did not match the target string or which contained extra keystrokes were discarded and requested again. For each input string, the duration for which each key was held (*key hold times*) and the time between consecutive key-presses (*inter-keystroke latencies*) was recorded and displayed back to the user in the form of a timing graph. Additionally, the data was transmitted back over the network to a central collection server. For analysis, every other input from a given user was considered to be the baseline data for that user. The other five entries were used to validate the resulting template.

2.4 Limitations

While we attempted to make the baseline data collection as unbiased as possible, there are still a few limitations. Most notably, since the data was all collected from a given user in a single session, there is no additional variance in the data due to the user's mental state (i.e. tired, excited, hurried, etc.,) nor do we capture the effects of slight variations due to different hardware (keyboard.) Finally, we applied no effort to ensure a sufficient diversity in test subject, so it is unclear what effect it would have on the results to apply the tests to a wider range of ages and/or ethnic groups.

[3] The Java timer APIs specify millisecond resolution. Looking at the actual data, it appears the most Java Virtual Machines provide timing data at a 10ms. resolution.

3 Genetic Analysis Methodology

For the actual genetic algorithm analysis of the keystroke data, we used a slightly modified version of the GENESIS 5.0 software [Gre90]. These modifications primarily took the form of additional hooks added to the driver code to allow the enhanced reporting necessary for the figures in this paper. The other modification was to use the "Experiments" parameter to control which target user we are generating the template data for. This differs slightly from the default behavior, which was to run multiple, independent experiments on the same data. Other than these modifications, the software was used in a normal fashion with the parameters and fitness function described below.

3.1 Tableau

Objective:	To find an optimal set of weights and intervals for a particular user's baseline key hold times and interkey latencies.
Representation Scheme:	**Structure:** Fixed length string **Alphabet Size:** $K = 2$ (binary) **String Length:** $L = 120$ **Mapping:** Each 10-bit group represents a floating point value between 0.0 and 5.0. Each group of four values represents the intervals and weights for a particular character's hold time and interkey time.
Fitness Cases:	Each set of template values is measured against the five baseline inputs for the actual user (gain points for accept, lose points for reject), as well as the baseline inputs for each of the other users (lose points for accept, gain points for reject).
Raw Fitness:	5 points for each correct match -5 points for each false negative 1 point for each correct rejection -1 point for each false positive
Parameters:	**Population Size:** $M = 50$ **Generations:** $G = 2000$
Termination Criteria:	The algorithm terminates after G generations.
Result Designation:	The individual with the greatest fitness value at the end of the run is designated the result template.

3.2 Genetic Parameters and Operations

The major parameters to the genetic algorithm used in generating the keystroke templates were determined primarily though trial and error. We began with the parameters first suggested by De Jong after a study of the performance of genetic algorithms in optimizing a representative set of functions [De 75]. In particular, his results indicated that a relatively low mutation rate (.001), a moderate crossover probability (.6), and population size of 50 were optimal in many cases. We started with these values and varied them slightly, plotting the results. Figure 2 shows a sample plot examining the effects of varying population size. It is interesting to note that while doubling the population size suggested by De Jong has little effect, if we double this again to 200, the performance drops off significantly. Other experiments validated De Jong's suggestions for crossover and mutation rates, so the final results were generated with the values indicated above.

Also discussed by De Jong is the generation gap parameter, which determines how many individuals are carried over from generation to generation without being subject to the genetic operations. The primary advantage in this strategy is that the best performing individuals are preserved as the run progresses. For situations where this *on-line* performance is important,[4] a value of $G < 1$ may offer some advantage. Since we are focusing on the case where our analysis runs off-line, we have set $G = 1$ for our experiments.

[4] the One-Armed Bandit problem being the classical example

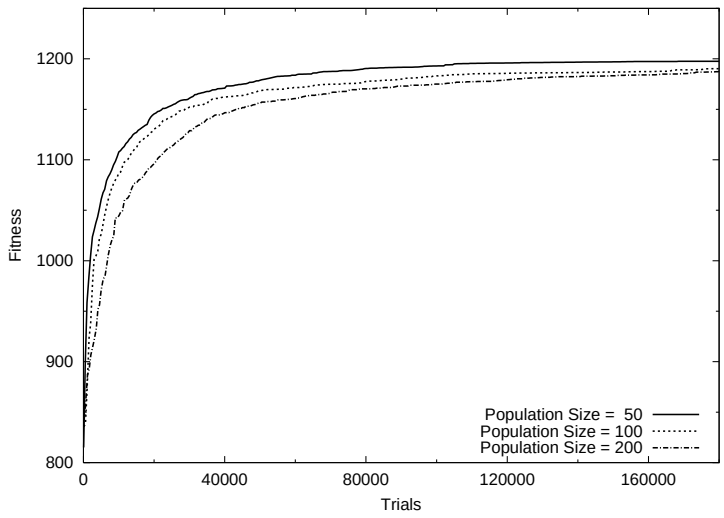

Figure 2: fitness evolution

The initial population was chosen at random and not primed in any fashion.

3.3 Representation

There are four sets of parameters that we chose to optimize in creating an authentication temple: hold-time interval, hold-time weight, interkey-time internal, and interkey-time weight. The weights allow a greater emphasis to be placed on those keystrokes and digraphs that more reliably identify a user. The greater variance in the weight values, the more useful they are. Since these weights have no intrinsic meaning, and only have an effect relative to the other weights, the actual interval of possible weights is unimportant. We arbitrarily choose the interval as 0.0 to 5.0, with 1024 possible values in this range. (A 10 bit representation per weight).

The interval values, on the other hand, do have intrinsic meaning. They represent a multiplier on the standard deviation for a given time. An interval value of .5 would indicate that a test value is valid if it falls within half a standard deviation (computed from the baseline data) of the baseline mean. Experiments with the test data showed that this value almost always falls below 5, so we chose our interval for this set as 0.0 to 5.0 as well, with 1024 possible values in between.

Each of the four values per character, then, is represented in ten bits. The string "dynamic keystroke via genetics" contains 30 characters for a total gene size of 1200 bits. It should be noted that the total search space for this problem is then 2^1200, which makes a blind random search quite unlikely to be effective.

These four values per character are arranged in the gene such that the related weights and intervals are mostly likely to survive the crossover operation.[5] For the given input string, the gene was arranged as follows (h=hold-time, i=interkey-time, d=deviation, w=weight):

$$\mathtt{d}_{hd}\mathtt{d}_{hw}\mathtt{dy}_{id}\mathtt{dy}_{iw}\mathtt{y}_{hd}\mathtt{y}_{hw}\mathtt{yn}_{id}\mathtt{yn}_{iw}\ldots\mathtt{s}_{hd}\mathtt{s}_{hw}\mathtt{s}\hookleftarrow_{id}\mathtt{s}\hookleftarrow_{iw}$$

3.4 Fitness Function

By far the most crucial part of the experiment is the determination of the genetic algorithm fitness function. As discussed in section 1, the fitness function determines the selections that the genetic algorithm makes and therefore directly controls the success or failure of the experiment.

For this problem, we evolved the fitness function over a number of iterations. In all cases, we first analyzed the half of the test data that we are using for analysis to determine the mean and standard deviation over the five inputs for every user. Since this is calculated for every character in the target string, this yields four

[5] i.e., the characteristic length is shortest for the weights and intervals of two consecutive characters

arrays of times for each user. This data is then used to evaluate the success of each proposed template as follows:

1. We test a particular instance of timing data against the template by iterating over each character. For each hold time, we find the distance from the mean hold time of the target user. If this distance is less than the deviation allowed by the template (gene), then we consider the character a match. We apply a similar test for interkey times.

2. If the entry that we are looking at actually belongs to the target user, then we would expect the entry to "match" and therefore we increase the score for every character that matches. The amount that we increase the score is determined by the weight specified for the particular character in the gene.

3. If the entry that we are looking at belongs to another user, then we increase the score for each character of the entry that *does not* match.

4. We iterate through all the entries of the target user in this fashions, as well as all the entries of all of the other users.

5. The final score is returned as the fitness value.[6]

While the above strategy had limited success, we made some modifications in successive runs to address issues that arose. As stated, the fitness function has a very conservative affinity. Since we have 14 impostors but only a single legitimate user, the impostor increases tended to overwhelm the results, leading us to add a fixed weight to the legitimate user increases to even out the results. There was also an issue with the weights specified by the genes tending toward the maximum value, since there was no disincentive to this behavior. We first remedied this by dividing the resulting score by the sum of the weights, hoping to gain a better distribution of weights. This, unfortunately, led to the opposite effect, where all but one weight was zero, with the remaining weight at .1, giving inflated results. The final modification, then was to divide by $sum_{weights} + C$, where C was experimentally determined.

This particular application of genetic algorithms is slightly unusual in that the fitness function evaluated against every individual differs from the function used in the final evaluation of the success of the experiment. Part of the motivation for this was the additional knowledge at fitness evaluation of whether the entry we were applying the template to was legitimate. In addition, as we discuss further in the next section, the actual results are best represented as a continuum of error probabilities.

4 Results

The effectiveness of an authentication solution can generally be summed up in two values: *false acceptance rate* and *false rejection rate*. The false acceptance rate (false positive) indicates the percentage of impostors that were incorrectly identified as legitimate. Conversely, the false rejection rate (false negative) identifies the percentage of legitimate identification attempts that are rejected. Since these parameters tend to be inversely proportional, an ideal keystroke timing authentication solution should provide a parameter which controls the relative emphasis on these two values. Figure 3 shows a sample plot of these two values as they vary between an emphasis on decreasing false negatives and an emphasis on decreasing false positives. This particular figure shows a static analysis as suggested by Umphress and Williams of the keystroke data collected for this experiment[UW85]. The x-axis indicates varying values of a deviation interval applied to all users. The entry was accepted if 60% of the values fell within the interval. Notice the steep rise of false negatives as the false acceptance rate approaches 0.

Figure 4 shows the error rates obtained by a randomly generated template at generation 0. The relative symmetry in the error rates is caused by our choice of weight applied to legitimate entries as discussed in section 3.4 above. The parameter that is varied across the x-axis is the threshold score that an entry must achieve to be accepted. The minimum and maximum values of this threshold are directly determined by the weights in the generated template.

[6] Note that GENESIS was configured to maximize the fitness function, which is not the default

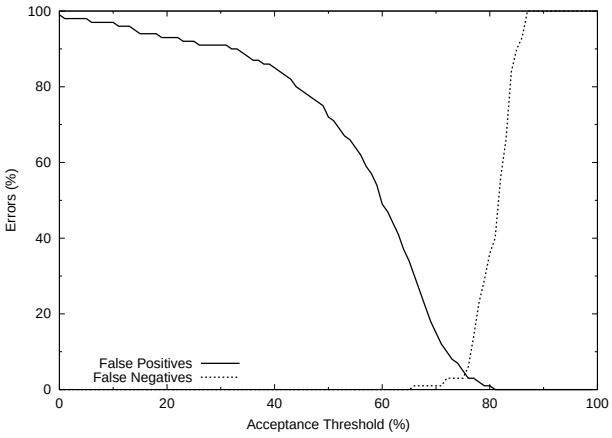

Figure 3: static analysis identification errors

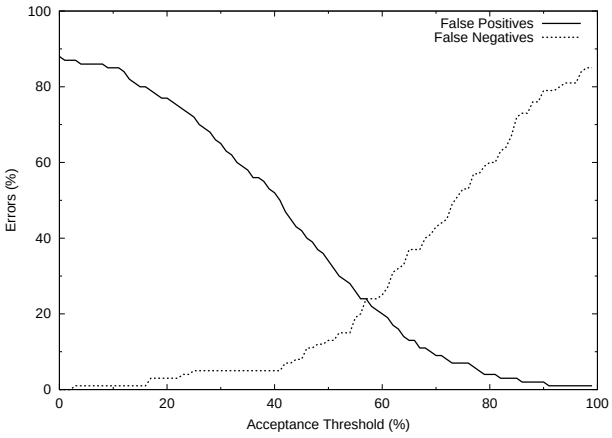

Figure 4: **generation 0** identification errors

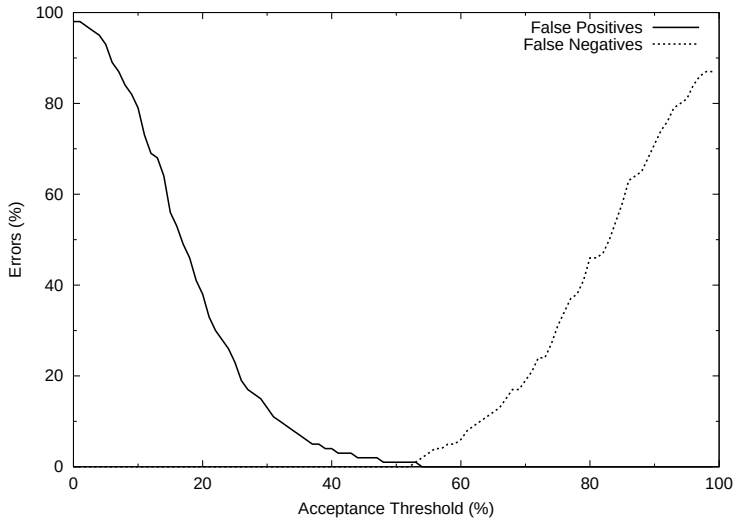

Figure 5: final generation identification errors

Figure 5 shows the aggregate error rate achieved at the finish of the genetic algorithm analysis. A significant result that is not apparent in this figure is that each individual had multiple values of the *threshold* parameter for which both the false acceptance rate as well as the false rejection rate was zero. The actual threshold at which this occurred varied slightly depending on the template, so the aggregate plot does not show this.

The average template deviation for all interkey-times, all users was .62, whereas the average hold-time deviation was .76, indicating a greater consistency in digraph times. Despite this, the average weight for digraph and hold times were roughly the same at 4.68 and 4.66, respectively.

The actual genetic algorithm analysis and template generation took just over 2 minutes per user on an Intel Pentium II 400MHz system. Consequently, all 14 identification templates were generated in under half an hour. Since all user information is analyzed in the generation of each template, it is expected that this analysis time would scale in a quadratic fashion.

5 Discussion

The results of the analysis were positive in multiple respects. The flexibility that the "score" threshold allows is significant in terms of the security options that it offers. For most situations, it is expected that

this threshold would be placed above the point where the false acceptance rate goes to 0, since forcing a user to retry authentication is merely an inconvenience, whereas allowing an impostor to login is unacceptable. Consequently, the initial slow increase of the false rejection rate is especially desirable in that it allows the threshold to be set at a point where the probability of a false acceptance is extremely low. Conversely, for those situations where the keystroke dynamics are used in addition to other methods to provide extra security, the threshold can be gracefully decreased to minimize the invasiveness of the keystroke timing. In total, the inverted bell curve shape is desirable for providing nice configurability characteristics.

There were a number of interesting trends in the individual templates generated that were hidden when looking at the data as a whole. A key result of this is that if we were to come up with a set of static criteria to apply to all authentication attempts, we would miss a number of potentially useful features. This validates the observations in [OS97], where neural nets were used to obtain a similar effect.

One such feature that is apparent in approximately half the user templates is the inconsistency of the inter-keystroke times before and after the space that separates words. This was manifested in either extremely low weights (approaching 0) for these values or extremely low deviations. In both cases, this leads to little differentiation between invalid attempts. The net result is that the derived template makes use of the "space-bar" transitions only for those users where it provides differentiation. Previous work has disagreed on the usefulness of these transitions, probably for the very reason that certain users are quite consistent in this respect, while others deviate greatly between words.

Another interesting result of the analysis was that the weights on the hold times and the digraph times were roughly the same. Our expectation, viewing the timing graphs displayed while the entries were being typed, was that the interkey times would be more distinctive and therefore more heavily weighted. In reality, the templates indicated that both sets of times provide roughly equal value.

In terms of the genetic algorithm analysis itself, the rapidity with which the algorithm converged on a reasonable template was slightly surprising. Despite the rather large representation, the algorithm was able to arrive at a "good" template in less than 20,000 trials. An unfortunate consequence of this rapid convergence is that the diversity in the population also quickly died away, with a typical population bias reaching 90% by 12,000 generations. In spite of this, the algorithm made progress up through approximately 120,000 trials before leveling off. (Figure 2)

6 Conclusion and Future Directions

We have demonstrated the use of genetic algorithms in creating templates useful for user identification and authentication. The false acceptance rate as well as the false rejection rate for each user template was 0% for some range of threshold values when applied to the set of 14 sample users. In addition, we have shown that the resulting error curves demonstrate that the identification templates are conducive to use in differing security evironments. This compares favorably to static analysis, which generally does not produce as favorable results with respect to configurability. Finally, we have shown that the analysis via genetic algorithms is practical for most timeshare environments, with each template generated in just over two minutes on a low end Intel Pentium II system.

One area of future work which requires a much larger sample set lies in the analysis of free-form sample text. Unlike the analysis of a fixed string, where the same characters appear in the same order, free-form text presents a number interesting problems in determining what set of characters are significant for identification purposes. Previous work has focused solely on character pairs, but larger chunks likely provide additional data. As discussed earlier in the paper, this type of problem is a perfect match for analysis via genetic algorithms in that the most significant chunks are identified and promoted via the inherent schema.

References

[De 75] Kenneth A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, 1975. Dissertation Abstracts International 36(10), 5140B; UMI 76-9381.

[GLPS80] R. Gaines, W. Lisowski, S. Press, and N. Shapiro. Authentication by keystroke timing: some preliminary results. *Rand Report R-256-NSF*, 1980.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learing.* Addison Weslay, 1989.

[Gre90] John Grefenstette. A User's Guide to GENESIS 5.0. Technical report, Naval Research Laboratory, Washington DC, 1990.

[Hol75] John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[JG90] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the Association for Computing Machinery*, 33(2):168–176, February 1990.

[Koz92] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

[MR97] Fabian Monrose and Aviel D. Rubin. Authentication via keystroke dynamics. In *ACM Conference on Computer and Communications Security*, pages 48–56, 1997.

[MRW99] Fabian Monrose, Michael K. Reiter, and Susanne Wetzel. Password hardening based on keystroke dynamics. In Gene Tsudik, editor, *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 73–82, Singapore, November 1999. ACM Press.

[MT79] Robert Morris and Ken Thompson. Password security: A case history. *Communications of the Association for Computing Machinery*, 22(11):594–597, November 1979.

[OS97] M. S. Obaidat and Balqies Sadoun. Verification of computer users using keystroke dynamics. *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 27(2):261–269, 1997.

[SVP97] Dawn Song, Peter Venable, and Adrian Perrig. User recognition by keystroke latency pattern analysis. Personal communications, April 1997. Class project.

[SWT01] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th USENIX Security Symposium*, Washington, D.C., August 2001. USENIX.

[UW85] David Umphress and Glen Williams. Identity verification through keyboard characteristics. *International Journal of Man-Machine Studies*, 23(3):263–273, 1985.