UNIVERSITY OF SOUTHAMPTON

Keystroke Dynamics as a Biometric

by

John-David Marsters

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

June 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by John-David Marsters

Modern computer systems rely heavily on methods of authentication and identity verification to protect sensitive data. One of the most robust protective techniques involves adding a layer of biometric analysis to other security mechanisms, as a means of establishing the identity of an individual beyond reasonable doubt. In the search for a biometric technique which is both low-cost and transparent to the end user, researchers have considered analysing the typing patterns of keyboard users to determine their characteristic timing signatures.

Previous research into keystroke analysis has either required fixed performance of known keyboard input or relied on artificial tests involving the improvisation of a block of text for analysis. I is proposed that this is insufficient to determine the nature of unconstrained typing in a live computing environment. In an attempt to assess the utility of typing analysis for improving intrusion detection on computer systems, we present the notion of 'genuinely free text' (GFT). Through the course of this thesis, we discuss the nature of GFT and attempt to address whether it is feasible to produce a lightweight software platform for monitoring GFT keystroke biometrics, while protecting the privacy of users.

The thesis documents in depth the design, development and deployment of the multigraph-based BAKER software platform, a system for collecting statistical GFT data from live environments. This software platform has enabled the collection of an extensive set of keystroke biometric data for a group of participating computer users, the analysis of which we also present here. Several supervised learning techniques were used to demonstrate that the richness of keystroke information gathered from BAKER is indeed sufficient to recommend multigraph keystroke analysis, as a means of augmenting computer security. In addition, we present a discussion of the feasibility of applying data obtained from GFT profiles in circumventing traditional static and free text analysis biometrics.

Contents

Chapter 1

Introduction to Keystroke Biometrics

1.1 Information security and biometrics

Almost every aspect of our daily lives involves some level of reliance on computer systems; many of these protect highly sensitive personal, commercial and financial data. Numerous examples exist of sensitive data protected by information systems, where unauthorised access could result in the loss of money or the unwanted disclosure of confidential information. For such systems to be operated effectively, we must be prepared to place a high level of trust in the security measures protecting them. Ensuring the confidentiality and integrity of the information stored in such systems is a critical responsibility and various methods are commonly employed to achieve this (Siponen and Oinas-Kukkonen, 2007).

1.1.1 Security fundamentals

The most important consideration when securing any computer system must be the provision of physical security for the devices responsible for authentication. For any computer whose console may be accessed physically, and whose internal components have not been protected from tampering, it must be assumed that its integrity can and will be compromised. Aside from attempts to ensure the physical security of systems, however, much of the effort in protecting the information must be related to authentication and verification of the identity of users. Authentication mechanisms are typically divided into three distinct categories (Brainard *et al.*, 2006):

1. Those based on something you *know*—the most common means of authentication, this includes the use of passwords and personal identification numbers (PINs). The

difficulty with this approach is that knowledge is easily duplicated, perhaps without the consent of the original owner. Eavesdropping and password 'sniffing' may present a significant security risk, and this includes both the risk of passwords being observed physically and of them being stolen logically, e.g., data being observed as they traverse a computer network. Complex passwords may be forgotten; simple passwords may be easily guessed or cracked. The apparent ease with which unauthorised users have gained access to online systems previously considered to be secure highlights the need for improvements beyond simple password-based security technology (Denning, 1992).

2. Those based on something you *have*—including physical keys and authentication tokens. Generally considered to afford a superior means of authentication when used in conjunction with password or PIN-based security, this type of authentication still suffers from the risk of tokens being lost, borrowed or stolen.

3. Those based on something you *are*—or more generally, physiological and behavioural characteristics: the field of biometric security. While use of this third scheme of authenticating individuals is currently the least commonly deployed across computer systems, it may represent an effective means of associating computer access with individuals, since its data source is the individual in question. It may also offer improved immunity against certain 'social engineering' attacks on security: whereas excuses such as "I forgot to bring my ID card" may convince an administrator to relax token-based authentication requirements, it is unlikely that someone could claim to have left his or her body behind. Indeed, an additional bonus of biometric identification is that its data source is rarely lost (although bodily injury can present interesting challenges to a biometric system). It is on this stratum of authentication—biometric analysis—that this thesis will focus.

1.1.2 Biometric techniques

Within the field of biometric security, there are many different techniques currently undergoing active research. Various factors of both human physiology and behaviour have been under investigation over the last few decades with respect to their usefulness in ascertaining identity (Miller, 1994; Furnell and Clarke, 2005). Typically, commercial biometric security research has tended to concentrate on physiological measurement as a source of authentication data; systems which use fingerprint, iris or retina recognition are now widely available and are becoming increasingly popular. In contrast, behavioural biometrics have developed rather more slowly and so this continues to be an area whose full potential remains untapped. This is perhaps unsurprising; behavioural measurement tends to present a more poorly specified problem, and can require the analysis of data in ways significantly less intuitive than, for example, comparing the shapes and contours of body parts.

Unfortunately, many of the techniques—particularly the physiological metrics—suffer from practical attacks which may affect their deployment (Uludag and Jain, 2004). Fingerprint recognition, for example, has been shown to require further consideration of liveness testing: successful attacks on first-generation systems have demonstrated them to be insufficiently secure for high-security applications (Matsumoto *et al.*, 2002; Tan and Schuckers, 2006). Many imaging-based physiological biometrics also suffer from the difficulty of liveness testing: since there is no requirement of performance on the part of the user, only presence, this must raise the question of whether the analysis is being performed on a live subject, or merely a photograph, video clip or model of their corresponding physiology (Schuckers, 2002).

Another disadvantage of deploying commercial biometric security measures is the associated cost. Password security on its own typically comes (almost) for free (Morris and Thompson, 1979), in contrast to the considerable outlay required to deploy fingerprint scanners or iris recognition systems at every terminal within a large organisation, for example. In business environments where this cost may be critical, the appeal of relying solely on passwords has resulted in slow uptake of biometric security systems.

In addition, a significant entry barrier for biometric systems has been that of user acceptance (Moody, 2004). Encouraging users to submit their personal biometric data—while still allowing them to feel assured that their privacy is not being put at risk—continues to pose a considerable challenge (Woodward, 1997; Furnell *et al.*, 2000). Furthermore, users are likely to be rather more wary of interfacing their body parts (such as eyes) with unfamiliar equipment compared to simply typing their password at a standard keyboard (Jones *et al.*, 2007).

These factors all add to the attractiveness of a biometric assessment that could be performed transparently by a computer system without the need for any additional equipment. If sufficient characteristic information could be extracted from the way in which a user operates the system, this might provide a particularly useful biometric in itself.

As early as the 19th century, telegraph operators demonstrated the ability to recognise other remote operators purely by the dynamics of their keying. The characteristic timing patterns, known colloquially as the ‘hand’ or ‘fist’ of an operator, could be used to determine their identity (Guven *et al.*, 2007). This would suggest that a significant amount of biometric information is present even in the timings of the repeated operation of a single key. Although over recent decades the use of keyboard and keypad devices has risen dramatically, the idea of exploiting typing technique as a biometric was originally dismissed by some as impractical (Walker, 1977).

Fortunately, more recent work by a variety of authors has demonstrated that neurophysiological factors, similar to those which make the handwritten signature of an individual distinctive, do indeed manifest themselves in the typing patterns of computer keyboard users (Joyce and Gupta, 1990). Sporadic research carried out since the beginning of the 1980s has demonstrated that—if appropriately analysed—keyboard timing data may indeed be a useful source of biometric information.

1.2 Keystroke analysis

In this chapter, we present a more detailed discussion of the technology behind keystroke analysis. We also examine both the benefits and the drawbacks associated with its use in practical security systems.

1.2.1 Sources of data

If we consider the use of keyed devices as a whole, we may note a variety of physical characteristics of their use. We may choose to examine, for instance, differences in the speed at which keys are depressed (i.e., their rate of motion during key travel), the impact on the key bed (how hard the keys are pressed), the durations for which the keys are held down, the intervals between successive keystrokes and so forth.

To maximise the effectiveness of metrics based on data gathered from very small typed phrases or codes, it is especially important to maximise the number of useful dimensions of information within the feature vector. It is in the area of keypad biometrics, applied to keypads such as those used for ATMs and also in biometric smartcard development, where keystroke metrics incorporating pressure data find their most suitable and effective applications (Henderson *et al.*, 2001). A limited amount of interest in the use of pressure metrics with computer keyboards has also been demonstrated in patent applications (Garcia, 1986).

In contrast, the area of keystroke metrics covered by this research involves avoiding the use of specialised equipment, and so the use of pressure sensors is not applicable. As a result, the only information available to the biometric system is that of keystroke timing. Hereafter in this thesis, references to keystroke dynamics as a biometric will concentrate exclusively on timing-based metrics.

1.2.2 Drawbacks of keystroke analysis

Since it is a behavioural biometric, keystroke timing analysis presents an inherently under-specified problem. Physiological characteristics of the human body typically only vary gradually with time (excepting cases of trauma), meaning that the main problem to be solved by the biometric researcher is in finding the most effective and compact feature vectors to represent the physiology of the body parts involved. In contrast, behavioural biometrics involve local temporality and deal with data generated in real time by an unreliable and typically extremely noisy source—the human animal. They are also particularly subject to the problem of failed registrations: one cannot measure the gait cadence of someone who cannot walk; similarly it is very difficult to measure the typing patterns of someone who is unable to type.

Aside from the problem of registration, and from the purely technical constraints of a low-dimension feature set containing only timing data, the next difficulty in exploiting keystroke dynamics as a biometric must surely be in the search for an algorithmic approach which is invariant under an extremely wide range of field conditions. While simple lab tests under consistent environmental conditions might generate well-correlated results for an individual user, there exists a considerable number of potential sources of noise that might shift the user's behaviour away from their normal typing profile. Weather conditions, for example: a cold day might mean that a typist's fingers move more slowly. Fatigue and stress could be influencing factors on typing behaviour. Injury could prove a particularly difficult problem to manage, potentially adding to it the insult of being locked out of a vital information system. Even the simplest of distractions, such as might regularly occur in an office environment, could add a significant amount of noise to otherwise consistent typing.

In addition, no formal study has yet been performed to determine variance in typing techniques between different keyboards and environments. Nevertheless, it seems reasonable to suggest that different characteristics might be observed when comparing the same user under different conditions: keyboard equipment is available in a variety of styles and with various operating characteristics, even when the analysis is confined to a single layout. Having said this, existing literature documents some success with web-based keystroke dynamics for static authentication, and while this has not always been made explicit in such studies, it seems reasonable to presume that such systems will be intended (and so perhaps tested) for portable usage (Magalhaes and Santos, 2005).

There may be further challenges associated with remote access, both due to timing inaccuracies from networking latencies and also the effect of delayed feedback on user interaction. It has been posited that the neurophysiology responsible for the characteristic nature of typing rhythms is linked to, or at least comparable with, that involved in handwriting (Joyce and Gupta, 1990). Furthermore, it has long been demonstrated that delays in the cognitive feedback loop employed for hand-eye co-ordination can seriously degrade the accuracy of performance (Held and Durlach, 1993). Perhaps, then, it is not unreasonable to expect that where typing is not performed 'blind', i.e., the user is attempting to view the results of their typing by watching the characters appear on-screen, any form of lag may degrade a user's performance and introduce additional noise. This would be true whether the delay is due to a lack of responsiveness locally or, critically for remote analysis, as a result of delays in communication over a network.

For successful long-term deployment of a system relying on keystroke biometrics, therefore, these are all issues which must be taken into account. The trade-off between false acceptance and false rejection warrants careful consideration dependent on the application, as do the sanctions imposed when the system decides an access attempt is invalid. Falsely accusing legitimate users of fraud is not good politics when attempting to gain user acceptance of a novel technology. Indeed, it is for this very reason that commercial interest in the timing and pressure biometric for ATM keypads has been so limited: for any one vendor to adopt the technology, they would have to accept the high likelihood of making their ATMs significantly more unfriendly—through the inevitable occurrences of false rejection—than those of their competitors.

1.2.3 Benefits and applications of keystroke analysis

Despite these drawbacks, however, the use of keystroke analysis—provided that it proves a sufficiently robust metric—may be effective in a variety of situations alongside more traditional methods of authentication. As is typical with behavioural biometrics it is unlikely that the technique would scale well enough to be deployed across a large user base with no additional means of verification. As a result the technique is most likely to find its niche as a means of strengthening existing techniques, or to provide a non-critical estimate of the likelihood of a security breach.

It is important to note that the biometric may be employed in two different ways: either as a means to reject a user where a measure of the likelihood of authenticity is below a given threshold—which we may call *active* authentication, or alternatively as a means by which we can augment log files or real-time monitoring without interrupting a user's session—which we may call *passive* authentication.

Fortunately, since keystroke analysis requires no additional hardware, its use as a means of hardening existing security measures may be as simple as installing a software upgrade. This fact, along with its near-transparency to the end user, may indeed be its greatest strength, as the barriers to including it as part of a holistic security system should be minimal.

Perhaps the most obvious example of augmentation is the strengthening of fixed password authentication with static keystroke analysis techniques (see Section 1.3.1). Here, the addition of the biometric to the identification process may be performed almost transparently (Cho *et al.*, 2000), although if the biometric is to be used *actively*, i.e., to reject the user from the system, it would be important to tune the FRR (False Rejection Rate) for the application and select carefully the behaviour on authentication failure. Telling a user that their access attempt has been rejected when they have entered the correct password would tend to create confusion and cause considerable ill feeling toward the software. This is particularly important in systems where the frequency of access attempts is high. Where a system is only to be accessed infrequently by a limited number of users for security-critical tasks, some false rejections may be acceptable. In contrast, if a user must access their account as often as daily, for example, and they are rejected more than a few times in a month, the administrators of the system may find themselves dealing with escalating numbers of complaints about its behaviour. Nevertheless, even in a frequent access situation where the FRR has been tuned to ensure *no* false rejections are permissible, it may still be useful to employ keystroke analysis *passively*: rather than rejecting access attempts, a measure of the system's confidence in their validity may be kept and used as part of a wider Intrusion Detection System.

Similarly, when used passively, it may be possible to use the dynamic analysis of free text (see Section 1.3.1) as a means of providing a real-time confidence of the authenticity of all currently active users in a multi-user environment. Across a large network, for example, it may be of benefit to administrators to be provided with a real-time "top ten" of the sessions least likely to be genuine—where these are security-critical accounts (such as those with elevated privileges on the domain) this may perhaps be used to trigger a silent alarm or recommend the use of remote monitoring to ensure that a session is valid and not a security breach.

1.2.4 Robustness of the metric

If we are to consider whether an authentication metric is in some sense useful in augmenting security, it is also important for us to define what we mean by its utility. Here we propose the introduction of two terms to aid this discussion: *specific utility* and *generalised utility*. In the former case, we must consider the abilities of our biometric in preventing a specific, motivated attack on a system: this is perhaps the stronger notion of utility. If we take this to its logical limit, we might criticise the *specific utility* of password protection with an objection such as "Surely one could point a gun at a legitimate user, demand their password and then use it at one's leisure?" Under these circumstances we consider how easy one might find it to subvert the security of the system if one *really wanted to*. Other extreme measures such as using severed body parts to defeat physiological biometrics also fall into the category of attacks on the specific utility.

We propose that this should be considered distinct from the notion of generalised utility, where one asks: "does deployment of this metric effectively allow us to raise our level of confidence in the security of our system?" In a practical environment, this distinction may often be paralleled by whether we consider an attack as being perpetrated by a determined attacker or merely an opportunist. This is not always the case, however, as a determined attacker may also seek to reduce the *generalised* utility as a way of adversely affecting confidence in the metric. Deliberately introducing measures to increase the probability of false rejections for legitimate users could be considered an attack on the generalised utility. Under these circumstances a biometric might be removed due to user rejection, allowing the attacker to avoid the need for attacking the specific utility directly.

In assessing both the specific and general utility of a biometric, we must consider how easily it may be compromised if the normal conditions of operation are violated. In the case of behavioural biometrics, the problem of liveness testing is less critical than with physical metrics, as we always require an act of performance. As a result, we may discount those attacks which are equivalent to—for example—holding a printed image in front of a camera to circumvent face recognition. Nevertheless, if we are to consider attacks on a particular biometric modality (as distinct from, say, attempts to subvert protocols used in the implementation) then we must assess our level of confidence that the data being analysed accurately reflect the performance of the subject we are attempting to authenticate. In addition, we must assure ourselves that these data are reaching the system unaltered, or assess the risk of tampering.

It would seem that in the case of a metric such as keystroke analysis—designed to be used in a relatively unconstrained working environment, and with little or no specialist hardware present—we must naturally assume a low level of confidence in the authenticity of the presented data. To some extent, this will depend on the environment in which the biometric is deployed. However, if we are to use a publicly available protocol for the analysis (that is, one whose statistical operations are not cloaked with obscurity) we must accept that the simplicity in specifying the metric will be reflected in the simplicity of a theoretical, methodological attack on our biometric. At the most basic level, if we have both the ability to control the information reaching our analysis process and also knowledge of what our biometric system is "looking for", then we can mount an effective attack on its specific utility. It is important to consider these two elements of an attack separately if we are to mitigate against them.

Firstly, let us consider the ease with which we might subvert the input to a keystroke analysis system. In the most simple case, our keystroke analyser might consist of software deployed on a local workstation, with no additional hardware or software protection. Here it is likely that we can attack the system with either software (provided that we are able to execute the software at the appropriate privilege level) or hardware. If we take the route of a software attack then it is possible that simply injecting an additional hook into the operating system's hooking chain will be sufficient to allow us to manipulate the timing of keystroke events, prior to their reaching the analyser. In the same way that the biometric analysis software may employ such a hook to *read* timing data, a rogue software agent might use the technique to alter the timing of keystroke events 'upstream' before they are received by the analyser.

Interestingly, the development of such a software agent by a determined attacker might allow either the specific OR the generalised utility of the system to be compromised. If a specific attack on an individual account proved too difficult for an attacker they might be able to use rogue software to disrupt the use of the metric by introducing timing "noise" into the system. Were this to occur, this would significantly decrease the generalised utility of the biometric system, raising the false positives to a point where user dissatisfaction resulted in the removal of the biometric protection. Fortunately, on a system with an appropriately applied security policy, the probability of this type of disruptive attack should remain low since it would require the attacker to install software which affected the machine's operation either at logon or during the sessions of other users.

Nevertheless, in the case of dynamic keystroke analysis, an attack on a specific account might still be possible with software, given that the code necessary for manipulating perceived keystroke timings need only execute in (unprivileged) user space. Preventing this type of attack would probably require particularly tight restrictions on software execution.

Perhaps more difficult to guard against than software attacks would be the deployment of hardware-based apparatus, designed either to disrupt the generalised utility of the system or to perpetrate a specific attack. Once again it is important that maintaining a high level of security in a critical system must include measures to ensure the integrity of the hardware. It is feasible that an in-line device could be constructed which sits between the keyboard socket of the computer and the keyboard itself. A keyboard whose internal processor has been modified might also be used in a hardware-based attack. In either case, an attacker would have the opportunity to use a suitably programmed microcontroller to store and manipulate the timing of keystroke signals upstream of the analysis software.

Of course, using these methods to disrupt the keystroke timings randomly before they reach our analyser only allows for an attack on the general utility. The second element we require for a *specific* attack is knowledge of the trends for which our analyser will be searching. In the simplest terms, if we are to attack Fred's account, we must know what information the analyser will use to determine our identity as Fred.

For the purposes of this discussion, let us consider our role to be that of an attacker. We can tackle this problem of impersonation in two ways: ideally, we would like to have knowledge of the algorithms used by the analyser and a copy of the exact statistical profile registered for the user in question. Given this information, the task of altering our stream of keystrokes to simulate the typing patterns of our target becomes significantly easier. On the other hand, if we do not have this information, we must consider whether we are able to use other algorithms and statistics to produce an effective simulation of our target's typing style, so that we maximise our chances of verification.

Let us consider the former case as an example. We can see clearly that if we are able to produce a series of keystrokes where the timing intervals are adjusted to mimic with those in a legitimate user's statistical profile, there will be little to distinguish us from that user. If we are able to obtain our target's profile directly from the configuration data of the analyser then our task becomes almost trivial. If not, we must find a way to construct an equivalent profile ourselves.

Once again, we are able to tackle such a problem using either software or hardware. With hardware, the device required to collect a keystroke profile would be essentially the same as that involved in deploying the biometric to enhance security. A standard hardware keystroke logger modified to capture timings as well as keystroke events would be trivial to manufacture. Deploying this for long enough to mimic the registration process would provide us with all of the data that we require. The greatest hurdle to the hardware-based approach is, once again, that we must deploy a physical device, which depending on the security of the target system may increase the chances of detection.

If we choose a software-based approach we may be afforded the luxury of making our attack almost invisible, using rootkit-like stealth techniques in our software installation, and this could potentially allow us a significant amount of time to gather typing statistics from our target. Nevertheless, such an approach would only be feasible if the local security on the system were sufficiently poor to allow us to install our software and for the software to operate within a privileged enough context to capture keystroke timings from our target user. One might assume that this is unlikely, given a computer system whose administrator has deployed biometric security analysis. However, it is important to remember that the target computer may not be the only machine used by our chosen user, nor is it essential for us to deploy the software agent covertly.

One of the most common ways that information and identity theft is perpetrated against individuals is through the use of the so-called "trojan horse" approach (Schneier, 1999). Rather than attempting to execute our profile-gathering code covertly we might choose to operate overtly but under the guise of some other function. This might offer us an alternative means to gather statistical information about our target's typing profile without his or her knowledge. Perhaps we might offer them a helpful utility for their word processor, or an amusing diversion for their lunch break. In any case, if we are able to convince our target to elect to execute some code of our choice, we should be able to begin collecting statistics relating to the target's subsequent use of the keyboard.

Additionally, it is interesting to note that this approach may not have to apply only to the target machine. If we are able to gather information about the target's use of his home computer, it is likely that the statistics will be useful in our attempts to compromise the security on his office computer, for example. While the portability of typing metrics has yet to be studied extensively, it is sensible for us to assume that there will be at least a reasonable amount of correlation between an individual's typing style on different keyboards. In the worst case, this means that we have a metric which (as with all biometrics) suffers from similar risks to the re-use of passwords across multiple systems.

Let us now assume that, by some means or other, we have accrued sufficient data to begin an attack on our target account. The focus of this research has largely been on dynamic analysis—that is, continuous monitoring applied after the initial authentication process. It is interesting to consider how well data gathered by this mechanism might apply to the problem of static analysis where the keystroke timings of an initial password (fixed text) are collected and verified. However, before we discuss this, let us concentrate on the more direct issue: once we have a profile constructed to attack a dynamic analysis system, how do we use this to aid our compromise of the account?

Our purpose in attacking a dynamic analysis system must be to avoid intrusion detection, as this is the area in which dynamic analysis finds its most sensible application. This, we presume, must either be because the target account affords us the opportunity to elevate our privileges within the system (it affords a means of achieving some other security compromise) or because we wish to commit identity fraud specifically pertaining to our target individual (in which case it represents an end in itself). In either case, we find that we need to enter information into the computer system and make it appear as though our target user was responsible for its entry. This means that there must be some kind of conversion process taking place, where we provide the typed information to our hardware or software intermediary and it re-aligns the timing of our keystrokes to match those of the impersonated user. We must therefore consider how such an intermediary might be implemented.

Relative to the desired statistical profile, it is clear that some of our input keystroke events will be too early and some too late. However, it is also evident that our intermediate device cannot perform alignment by moving events *earlier* in time: it is obviously not possible to generate output events before they arrive at the input. The only way for the device to re-align the timings is to move events *later* by varying amounts. Therefore we must introduce a delay in our system, a buffering period which we manipulate on an event-by-event basis, modifying the relative positions of the events in the output stream. As an example, let us imagine that our input includes two keystroke events, and their inter-key latency is 50 ms. If our target statistical profile suggested that these events should actually have a mean time difference of -20 ms, we might delay the first event by 2000 ms and the second by 1930 ms.

The problem with applying this approach naively (and the reason that the buffering period needs to be non-trivial in size) is that alterations in the buffering period must be cumulative. At a macroscopic level, we can see that if our typing speed is slower than that of the desired target, we will eventually experience a buffer under-run. Considering the example above, let us imagine that we have a continuous stream of input events with latencies of 50 ms, all of which need to be re-aligned to −20 ms. Our buffering period will reduce as: 2000, 1930, 1860, 1790, ... etc., and eventually this figure will reach zero and our re-alignment algorithm will be unable to introduce the appropriate delays.

How, then, are we able to use such a method to fool a timing analyser? The trick is simple. Any practical timing analysis biometric must allow for "pauses for thought"—rejecting latencies higher than some threshold as not being pertinent to analysis of the user's normal typing behaviour. Statistically it makes little sense for the analyser to do anything other than to place a ceiling on the measured latencies. This means that for glitches in the input stream of *below* the ceiling latency we are likely to be penalised for deviating from the expected statistical profile, but for pauses which last *longer* than the ceiling latency the analyser will simply ignore the discrepancy as human unreliability. Therefore if we ensure that our buffering period is above that ceiling, we have a way to break the stream of events without penalty. Each time the buffer period drops near zero, we simply stall for the length of the analyser's latency ceiling and re-accumulate that amount of time in the buffer. To clarify: if the events in our example continue to require a reducing buffer period, we keep going until the buffer nears zero. Then, if our analyser has a ceiling of 2000 ms (or less) we simply stall our output for that length of time and accumulate the 2000 ms in our buffer length. This will result in a statistically correct output stream for some 28 events, followed by a pause for a couple of seconds (large enough that the analyser will ignore it), followed by a continued statistically correct output stream, and so on.

Clearly the major downside to a device designed in this way is that the response from the system will not be real time. The users' experience will be significantly degraded, as keypresses will not reach the underlying operating system for a considerable period of time. However, we suggest that an attacker attempting to subvert a dynamic analysis system in this manner would be afforded little sympathy!

1.3 Keystroke analysis in published literature

Although discussions had previously taken place regarding the plausibility of recognising individuality in typing rhythms, the first researchers to publish a formal study using computer equipment were Gaines et al. in 1980. Even though their work was on a very small scale, using a total of only six professional touch-typing secretaries, its contribution to the field was significant in that it catalysed further studies over subsequent years. Not only did their attempts to perform *manual* identification of timing data sets (i.e., having a human operator compare tables of figures) provide encouragement with their 100% recognition rate, but also their *automated* processing, highly statistical in nature, showed that the technique was worthy of further investigation.

Although the recognition methods developed to effect authentication through keystroke analysis could be applied to circumstances of both verification ('is this person who they claim to be?') and identification ('who is this person?'), it has been suggested that the former applies more directly to keystroke biometrics (Ilonen, 2003). This is perhaps typical for many biometrics, since it is an easier problem to solve, especially when the nature of the information is underspecified.

Nevertheless, the concepts are discussed interchangeably in this thesis; measurement of the effectiveness of approaches is typically considered with respect to accuracy of identification within a gallery of possible users. This allows us to focus on the level of useful information in the data, rather than on the applicability of any one metric or its accept-reject threshold. If we develop a robust metric that can identify users and separate their data successfully, the verification problem is also necessarily solved: we simply use the raw confidences calculated by our classifier to determine how well our sample matches with the profile of a specific user in the gallery.

1.3.1 Static versus dynamic authentication

There are two distinct security methodologies to which keystroke dynamics can be applied. More commonly, previous research has concentrated on the ability to use keystroke analysis on a known phrase at the point of login, known as *static* authentication. The timing patterns of a user's ID or password, or the two in combination, are considered in comparison to a reference profile with which the system has been explicitly trained (Monrose *et al.*, 1999, 2002). The degree of similarity between the trial and the expected typing profile determines acceptance or rejection of the login attempt. The popularity in the literature of this application of keystroke biometrics is perhaps unsurprising: it is a very simple way of applying keystroke analysis to aid system security, since the nature of the sample data is essentially invariant (Monrose and Rubin, 1997). The same set of keystrokes should be delivered every time a login attempt is valid, and so the profile can be built rigidly around the timing of the training attempts. The metric used and the selectiveness of the algorithm can be tuned to balance the false acceptance rate (FAR) and the false rejection rate (FRR) appropriately for the application. In addition, it may be possible to incorporate live re-training on patterns considered to be generated by impostors, so as to reinforce the strength of the classification model (Lee and Cho, 2007).

Furthermore, applying keystroke analysis in this static context allows for practice and therefore increasing consistency through learning. Typing one's password in a particular fashion over many login attempts will tend to be habit-forming, meaning that the process of matching becomes even more effective. Even inexperienced 'hunt-and-peck' typists will tend to develop a repeatable rhythm to their password once they become accustomed to where its constituent keys are on the keyboard (Obaidat and Sadoun, 1999).

Unfortunately, this also has its disadvantages. Firstly, there is a requirement for the system either to have a reasonably generous tolerance with reference to the initial training, or otherwise to have some sort of temporal adaptability (Umphress and Williams, 1985). As a user becomes increasingly familiar with their password, their rhythm is likely to change somewhat, and the total speed of entry will increase. Patterns in the timing may gradually emerge which differ from the rhythm of the initial training, and it would be undesirable for increased familiarity to lead to an increased FRR.

Furthermore, at the point where a user then changes their password, the biometric system must re-learn the user's typing style in the new context from scratch, since the previous profile will almost certainly fail to generalise over the new data stream (Yu and Cho, 2004). It should be a concern, therefore, that the inconvenience associated with re-registration may lead to avoidance and therefore password stagnation. The fixed-text nature of the registered input stream may be good for keeping the algorithms simple, and for demonstrating good theoretical results in those studies which do not account for the training of the user as well as the system. Unfortunately this may disguise the resulting inconvenience which may manifest itself only in the deployment of practical systems designed for long term use.

In contrast, the more generalised keystroke security paradigm of *dynamic* involves attempting to match profiles of typing styles to an *unknown* and highly varying stream of input data (Leggett *et al.*, 1991). Continuous monitoring of keyboard use in this way could potentially allow a system to generate real-time or near-real-time estimates of the confidence in its users' authenticity. Similarly, applied offline it could be used (in combination with appropriate cryptographic techniques) in the context of secure document processing, providing an means of establishing a level of confidence in the identity of a document's author.

The technique of dynamic authentication also addresses some of the difficulties associated with securing computer terminals. Simply establishing an appropriate level of confidence at the point of login on a system provides no guarantee that the person who logged in *then* is the person still operating the terminal *now*. Systems logged in correctly may still be subverted if left unattended; indeed, if we consider the extreme of login-only security measures, we have nothing to protect against the possibility of an authorised user being made to gain access under duress and then to relinquish control of the terminal.

The wide ranging possibilities for a transparent biometric able to address these issues make dynamic authentication an attractive proposition (Brown and Rogers, 1993). However, it is not without its drawbacks. Not only does a dynamic monitoring system have to cope with the highly varying character stream at its input, and seek profiling methods which generalise as much as possible over this stream, it is also far more likely to be subject to a high level of noise. While a short phrase such as a well-practised password may be entered with a low likelihood of interruption, the breaks in typing in a dynamic situation—such as those caused by distraction and interruption, pauses for task cognition and so on—would form such a significant part of the latencies in the data stream that an algorithmic method would have to be very carefully designed to take this high unreliability into account.

Furnell *et al.* (1996) also proposed the possibility of using static authentication in a continual context: using static techniques to protect the use of certain console commands, which they term 'volatile' (high risk). However, their work suggests that the risk of repeated false rejections in a session is too great and, therefore, they do not believe the technique to be useful for deployment unless it can be strengthened in some way. Furthermore, this technique is perhaps less useful today than it might have been at the time when their report was written, because of the gradual decline of console use in favour of Graphical User Interface (GUI)-based administration: their report, for example, talks about the volatility of various MS-DOS commands which are rarely used today even under its successor, the Windows command interpreter. Nevertheless, the idea of using keystroke analysis in an application-specific context is still given consideration by some authors (Dowland *et al.*, 2002).

1.3.2 Inter-key versus key hold timing

In searching for a feature vector which incorporates as much of the useful timing information as possible and yet generalises well, studies have investigated measures both of the inter-key time and the hold time. Inter-key time represents the time between consecutive keystrokes in a phrase, in contrast to hold time which represents the duration for which each consecutive key is held down. The original work of Gaines et al. was based entirely on the idea of inter-key digraph latencies; other work, however has given a greater deal of significance to the hold times of individual keys, and it has been proposed that using hold time alone may afford a more effective metric than using inter-key time alone (Obaidat and Sadoun, 1997a,b). Naturally, however, fusing the two metrics affords a richer feature vector, and the increased dimensionality may improve the reliability of the authentication technique.

Within the field of keystroke research, there also appears to be some variation in the nomenclature. In some cases inter-key time is considered to be the time between two key-down events. In other cases, perhaps less commonly, it is considered to be the period between a key-up and a consecutive key-down event. In this thesis, we use the latter definition. This is deliberate: by using this definition we are able to identify the overlapping of keypresses with ease, through inspection. Where a user has a tendency to overlap keypresses, their average inter-key latencies will be *negative*. Results gathered as part of this research demonstrated very clear variation between users with respect to whether they overlap keystrokes or not—some of the core contributors to the study were easily separable purely by this simple measure.

Typically, the research into the relative effectiveness of hold and inter-key timing as metrics has been performed in the context of static verification (Bleha *et al.*, 1992). This simplifies the matching problem considerably since the context of each letter within the whole phrase is known and is effectively invariant during training and testing. In extending the systems to allow continuous monitoring, research has typically tended to depend on the notion of building profiles of digraphs.

More recently, however, work carried out by Bergadano *et al.* (2002) has relied on the use of a measure which is neither hold time nor inter-key time—rather, they use a metric which they refer to as the *duration* of a trigraph. This measure can be thought of as a kind of 'compound' inter-key latency, since the period of measurement is between the key-down event for the first element of the trigraph and the key-down event for the last element. Similarly research published by Dowland and Furnell during the lifetime of this study (and perhaps the closest in research focus to this thesis) has used a notion of 'overall' duration for trigraphs and pre-selected keywords (Downland and Furnell, 2004).

1.3.3 Statistical versus neural methods

There is one further loose distinction which we may draw when discussing approaches to interpreting keystroke timing data: we may separate methods which are statistical in nature, and those which employ techniques based on the neural network paradigm. The literature shows conflicting favour for each approach, with several authors keen to stress the overriding benefits of their chosen methodology. Neither is the sum total of published works clear on whether the two approaches apply with equal benefit to both static and dynamic authentication, or whether there is merit in considering the two problems separately. Given this lack of clarity, and considering that machine learning approaches based on techniques such as fuzzy logic tend to blur the distinction between paradigms (indeed, are not all classification methods ultimately statistical to some degree?), the merits of separating the two entirely are open to question. Nevertheless, it is a distinction already drawn by some authors (Haider *et al.*, 2000), and so we give this some brief consideration here.

Initially, the work undertaken by Gaines *et al.* was entirely and obviously statistical in its nature. Their work involved plotting histograms of the latencies of a limited number of digraphs for each user, and used a combination of statistical methods and pure inspection. Many other authors have followed a similar approach with greater degrees of sophistication, both when tackling static and also dynamic authentication. Robinson *et al.* used statistical classifiers in attempting to tackle the problem of purely static analysis. Bleha *et al.*, Mahar *et al.*, and Bergadano *et al.* also used approaches which, rather than using a neural (or in some sense self-organising, nebulous or biologically motivated) methodology, relied on transparent statistical calculations to produce their classification results. Most recently, the work of Downland and Furnell, which focused exclusively on dynamic analysis, took a statistical approach, examining overall latencies for multigraphs and keywords to determine deviation from the mean values of a user's profile (Downland and Furnell, 2004).

In contrast other literature, typically published since the mid-1990s, has included the application of artificial neural networks to the authentication problem, and the results have been equally encouraging (Brown and Rogers, 1993). The consolidated works of Obaidat go so far as to suggest that, at least for static authentication, the use of automated machine learning techniques can surpass that of the purely statistical methods (Obaidat and Macchiarolo, 1993, 1994; Obaidat and Sadoun, 1997b). Interestingly, Furnell *et al.* also used neural techniques for their static verifier, but opt for statistical techniques in developing a dynamic authentication mechanism (Furnell *et al.*, 1996). Whether the the apparent superiority of the neural techniques holds in general is not clear, but it is evident from this body of work that the use of automated learning, as distinct from crafting formulae, has a clear place in tackling the ultimately awkward and underspecified problem of keystroke analysis.

1.3.4 Digraphs and contextual information

Profiling users' typing patterns in a manner which lends itself to continuous authentication poses an information challenge. Typically, work on dynamic authentication has been based on the notion of digraph latencies originally introduced for static techniques by Gaines et al. The authors also developed the notion of 'core digraphs', that is, the idea of building a small set of characteristic digraphs which themselves provide sufficient means by which to authenticate a user (<in>, <io>, <no>, <on> and <ul> are suggested for right-handed typists). Later work, however, proposes that this idea is flawed and that the practice lends unwanted bias, overfitting to the particular sample set in the study rather than generalising well. To the modern observer, this should come as no surprise, and it is only fair to note that the Gaines study was only intended as a preliminary report.

In their work extending keystroke analysis to dynamic authentication, Leggett and Williams undertake an examination of the effectiveness of the use of digraphs in building user profiles (Leggett and Williams, 1988). Of the approaches attempted, the digraph-based metric certainly seems to give the most promising results. Examining the timing patterns of keystroke pairs makes a good degree of sense in trading off the difficulty of handling large and complex data sets versus having sets too small and homogeneous to be useful. The importance of providing sufficient contextual information to afford greater data richness is discussed in more depth in Section 2.4.

1.3.5 Commercial implementations

Although keystroke analysis has been a recognised area of interest within biometric research for some time, it is suggested that there is still considerable room for development and improvement on current techniques. Static authentication is by no means a mature technology, dynamic monitoring even less so. A variety of questions remain unanswered, not least the issue of how to deal with typing errors and how to incorporate tolerance for genuine mistakes in typing patterns. Thus far, no study has investigated how to deal with such errors most effectively (Obaidat and Sadoun, 1999).

At the time of writing, it seems that no commercial systems have been produced which exploit the dynamic monitoring paradigm. There has been some limited commercial interest in using static analysis as a means of hardening, for example, the login process on Windows machines. Although there has not been the opportunity to examine such systems formally, as the full software requires significant financial outlay, informal trials were attempted using a downloadable technology demonstration from one such company, Net Nanny Software Inc. (from `http://www.biopassword.com/`).

Asking colleagues to train the system and then attempting to mimic their typing style gave the impression that the software was limited in its reliability. This applied to both false acceptance of mimicking and also false rejection, where legitimate users were unable to reproduce their own behaviour. It was also noted that a remarkably effective way to defeat the system for a known password was simply to listen to the rhythmic typing pattern of the legitimate user and to mimic this when attempting to defeat the system. It is suggested that this may apply more generally to any static authentication system, particularly those which rely heavily on inter-key analysis. Augmenting 'shoulder surfing' of passwords with covert audio recording might enable an attacker to practise the right keystroke pattern to defeat the static analysis. Reported accuracies for neural techniques applied to small scale *static* authentication under laboratory conditions are typically as high as 97.8% (e.g., Obaidat and Macchiarolo, 1993). How well such systems scale to cope with very large user bases is less well published.

1.4 Research hypothesis

Research in the field of dynamic analysis has thus far concentrated on data collection under controlled conditions, requiring specific performance from users who have been encouraged to 'type some things now and fill this text box' (Gunetti and Picardi, 2005). Our hypothesis, however, is that we can extend the use of timing analysis to cover the *genuine live computing environment* where there is no requirement for specific performance. This being the case, we would be able to demonstrate the utility of keystroke timing analysis in monitoring security in a practical setting, where the analyser could be used to provide a measure of identity likelihood, even where the activities of users are unknown. In particular, this would lend itself to the paradigm of intrusion detection, where we might augment existing methods of identifying unauthorised computer use.

For genuinely free text, the only effective way to gather the necessary statistics to investigate this hypothesis is to capture timing information during normal computer operation. Having determined the lack of existing research into the examination of genuinely free text, it was decided that this should be the primary focus of this research project. The question we set out to answer is this:

Is it feasible to develop a software platform which will enable collection and analysis of biometric data based on genuinely free text, and will this provide us with sufficient richness of information for the metric to be useful?

This is the idea which motivated the development of the BAKER (Biometric Analysis of Keystroke Entry Rhythms) software.

The BAKER software (discussed in Chapter 3 in great detail) was developed to collect keystroke timing data in a way that would be transparent to the end users—in this case our test volunteers—while safeguarding the users' privacy. At the same time, it had to produce a sufficiently broad set of biometric data that we would be able to perform offline analysis, to determine statistical significance with respect to identity. It follows that the criteria for success in this project are:

- totally transparent collection of keystroke data from real users, without compromising privacy and security;
- demonstration of the ability to separate users according to their keystroke data;
- selection of a classifier to demonstrate the practicality of using the biometric live and in real time, in a genuine computing environment.

In short, we set out to determine if and how typing behaviour of individuals can form an effective component of intrusion detection in computer security.

1.5 Structure of the thesis

Given the research hypothesis outlined above, and the consequential criteria for success that we aim to address via the BAKER system, the remainder of this thesis is structured as follows.

To facilitate the development of such a system, a series of preliminary experiments was carried out. We present the results of these experiments and an analysis of their implications in the next chapter.

The subsequent chapter sets out the development process for the BAKER platform and discusses its resulting design in detail. The structure of each of the software components is presented along with explanations of the motives for each design decision.

Having explained the technical structure of the software platform, we continue in Chapter 4 with a discussion of the deployment of BAKER into the live environment. Our approach to the data collection process is presented, along with details of the analytical tools used to examine the incoming information.

Our experimental results are then set out in Chapter 5. We also attempt to fulfil our third success criterion, by selecting a classifier which separates users by typing pattern in a manner which is both accurate and also fast enough for a live computing environment.

Finally, we close with a critical assessment of our experimental results.

Chapter 2

Fixed-text Experiments

To provide practical data which would guide the development of GFT-analysis software, a set of preliminary experiments was carried out. These experiments were based on fixed text rather than free text, and they were intended to collect raw keystroke timing data from a variety of users with different levels of typing experience. To avoid unnecessary environmental bias, no constraints were placed on the users regarding the total time of execution, nor the circumstances under which the tests should take place.

The software for the experiments was made freely downloadable from the world-wide web. Those invited to take part were primarily friends and colleagues of the researcher, and their friends and associates in turn. Of those who agreed to take part, a total of 14 submitted their results in a timely manner and agreed for their analysis to be included in this thesis.

No particular experimental rigour was used with respect to the mix of participants involved. Nevertheless, the range of typing expertise—according to the participants' self-assessment—covered a broad range, from hunt-and-peck keyboard users to experienced touch typists.

2.1 Software and experimental method

The software used for the experiments was written in the C language and compiled to run on any recent Microsoft Windows platform (or mature emulator). The decision to use the Windows platform was largely motivated by its wide user base. To maximise the number of potential participants both for this experiment and also for the more in-depth analysis later in the study, it seemed natural to build the software on the OS used by the largest proportion of the target group. In tackling a high-resolution timing problem such as this, it is important to be aware that timing mechanisms on computers tend to be very much operating system specific. As a result, and due to the overwhelming popularity of Windows versus other operating systems, it was decided that producing either cross-platform software or multiple versions for different systems would not be a worthwhile investment of time.

In an attempt to gather accurate timing statistics, the software used a method of the Windows API (Application Programming Interface) called `QueryPerformanceCounter`. Several of the timing mechanisms on the platform (typically those based on system clock interrupts) fail to give accurate results as their granularity is too coarse. In contrast, this very high resolution system timer (used in combination with its sister method QueryPerformanceFrequency) gave the most reliable, accurate and hardware-portable mechanism for recording timestamps of events. For every key-down and key-up event generated by the user typing into the text box in the test window, the software was able to store a record of the key, the event type (whether up or down) and the timestamp counter, accurate to 0.1 ms.

The program for collecting the experimental data was designed to be simple and self-contained. At the end of the experiments the software saved a log file into a user-selected directory, and the participant was then invited to e-mail this results file to the researcher at their convenience.

The data collected in these preliminary experiments were of a form analogous to that used for static authentication: participants were required to type fixed known phrases rather than random uncorrelated text. The simplicity of this initial approach meant that the data could be more easily analysed, comparing like with like. In addition, the constraints on the nature of each sample allowed the sizes of the data sets to be much smaller, which made for rather more willing participants: successful dynamic trials in previous literature have typically required much larger sample sets from each individual than for static analysis. Managing dynamic sets and attempting to ensure quality control on a small-scale experiment such as this one would have been prohibitively difficult.

At the start of each experimental session, the software presents the participant with the opportunity to undertake a 'dummy run' with a phrase not present in the main tests. This allows the participant to become familiar with the interface, decreasing the chances of cognitive and procedural difficulties which might lead to problems with the first formal test run. In this 'dummy' phase the software presents the short phrase '`AMBER LIGHTS`' and invites the user to type this a total of four times. Both the phrase and the number of repetitions were chosen arbitrarily, the only goal being to allow some user accommodation.

In this phase, as in all the other phases, the software only accepts correctly typed phrases, with no punctuation or use of alternative keys (such as using `Shift` for capital letters). All typing exercises and the results of all keypresses are given directly in capitals. In common with all other published studies thus far, the software makes no attempt to handle typing errors intelligently: if an attempt is made to correct an entry, the whole phrase is cleared and the user is invited to make a fresh attempt. It is acknowledged that for users making a disproportionate amount of mistakes this may provide them with a slightly increased chance of learning a pattern to the phrase, through carrying out a greater number of total attempts. After consideration, this effect was ignored, on the assumption that its impact should not be significant, and it should not affect the ability to distinguish between participants. Indeed, were there to be any measurable effect it should act to compact the field very slightly, bringing the 'worse' typists' data slightly closer to that recovered from the 'better' typists—the final analysis would then be slightly more cautious rather than slightly overstated.

After the dummy run, the main part of the trial begins, in three distinct sections. In each section, the user is invited to type a set phrase correctly a given number of times. The user is given the phrase before the test begins, and they are reminded that it is not a competition and that they should not try to rush and end up making mistakes. The user may view the test phrase for as long as they wish before beginning the typing trial. The phrase is displayed on the screen at all times during the trial, along with a counter showing the number of remaining times that they should enter it. Characters entered are displayed a text box as with any standard text entry interface. All attempts, both valid and abortive, are logged, but only valid attempts used in the analysis.

The three phrases selected for use in the experiments were as follows:

- `THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG`, collected 6 times
- `THERE ARE MANY DIFFERENT TYPING PATTERNS`, collected 6 times
- `THERENT TYPINY DIFFERNG PARERE MATTE ANS`, collected 10 times

The first of these is well-known stock text which covers all of the letters of the alphabet in a reasonably compact phrase. The second and third phrases form a pair chosen to investigate the effect of context and cognitive delays. Prior literature has suggested that digraphs alone provide sufficient information to profile users for dynamic monitoring. For this reason, phrase 3, clearly less natural in its composition, was deliberately crafted to include an identical set of digraphs to phrase 2 so that a direct comparison could be made between the two.

Once the data had been gathered and converted into a suitable format, they were then analysed using the MATLAB software toolkit.

2.2 Data analysis and discussion

To analyse the data gathered from each of the 14 participants, the data were first assembled into six distinct matrices per user, a pair for each of the three test phrases.

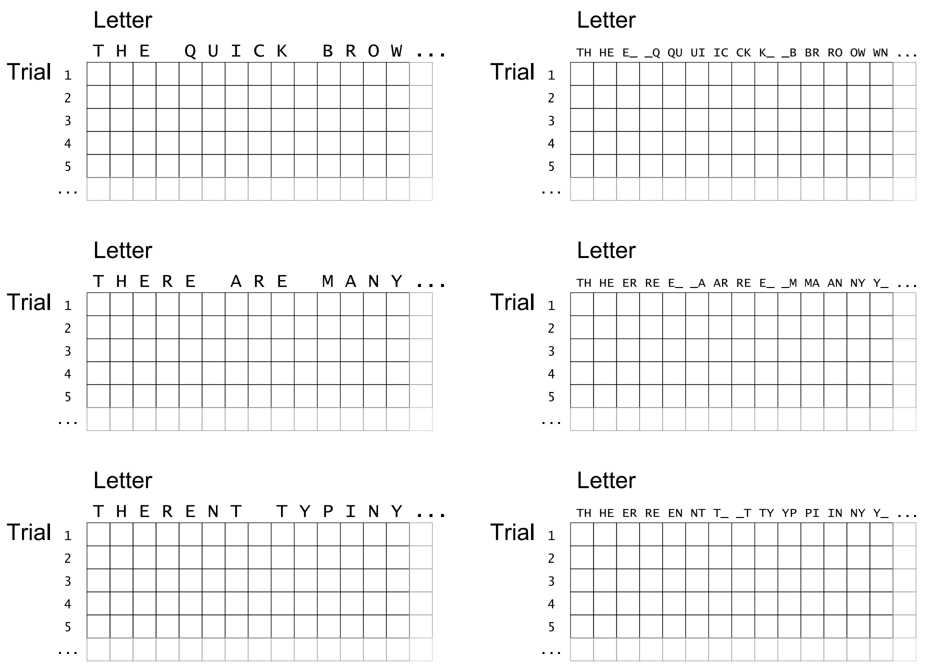

Letter
T H E Q U I C K B R O W ...
Trial
Letter
TH HE E_ _Q QU UI IC CK K_ _B BR RO OW WN ...
Trial
Letter
T H E R E A R E M A N Y ...
Trial
Letter
TH HE ER RE E_ _A AR RE E_ _M MA AN NY Y_ ...
Trial
Letter
T H E R E N T T Y P I N Y ...
Trial
Letter
TH HE ER RE EN NT T_ _T TY YP PI IN NY Y_ ...
Trial

FIGURE 2.1: The set of input data was assembled into six matrices for each user, two per phrase, representing hold times (left) and inter-key times (right)

Of the pair, one corresponded to the hold times for each key, the other to the inter-key delay between consecutive keystrokes (Figure 2.1). In this way similar tests could be performed using both hold time and inter-key metrics.

With each row in the matrix representing one trial by one individual, the columns corresponding to the position in the string, we derive feature vectors from each trial. For a test phrase of length N, let n represent the location of a letter (or letter pair) in the phrase, defined over the range $1..N$. Let the up-time and down-time vectors be:

$$\begin{aligned} \mathbf{U} &= [u_1, u_2, \ldots, u_N] \\ \mathbf{D} &= [d_1, d_2, \ldots, d_N] \end{aligned}$$

Then:

$$\begin{aligned} \phi_n &= u_n - d_n; \qquad \forall n = 1, 2, \ldots, N \\ \mathbf{H} &= [\phi_1, \phi_2, \ldots, \phi_N] \end{aligned}$$

for the hold time analysis, and correspondingly:

$$\begin{aligned}\psi_n &= d_{n+1} - u_n; \qquad \forall n = 1, 2, \ldots, N-1 \\ \mathbf{L} &= [\psi_1, \psi_2, \ldots, \psi_{N-1}]\end{aligned}$$

for the inter-key metrics. It may be noted that in the inter-key case, overlapping of typed characters can result in negative delays being recorded.

To perform some simple matching tests on the results obtained in this way, a simple Euclidean distance classifier was constructed to operate on arrays of these feature vectors. The same procedure was followed for both styles of metric; the notation in the discussion below refers to the hold time features, but the method applies equally to the inter-key tests.

To perform the testing, we selected a representative subset of the data for training: in this case, all test runs except the last. The last test run from each participant was used exclusively for testing, with the assumption that the typing was likely to become more stable during the test rather than less.

For each phrase, we defined a *gallery* of vectors, $\mathbf{G}$, representing each performance of the phrase for each of the participants. The training cases were all included separately, rather than using means per-user, as this meant outlying cases (where, for example, a participant may have been interrupted during the test) did not skew the other results from that participant.

Then for each test case for matching, we compared it to the set of gallery vectors applying the Euclidean distance classifier as follows: we define the distance ε_k between the test vector $\mathbf{T}$ and each gallery member $\mathbf{G}_k$ as:

$$\varepsilon_k = \|\mathbf{T} - \mathbf{G}_k\|^2 \quad \forall k = 1, 2, ..., |\mathbf{G}|$$

given the definition of the Euclidean Distance:

$$\|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

We then match the test case to the best fit vector in the gallery $\mathbf{G}_j$, where:

$$\varepsilon_j = \min_k \varepsilon_k$$

and declare the classified user for the test case to be the same user responsible for the $\mathbf{G}_j$ training data.

	Hold times (%)	Inter-key times (%)	Combined metric (%)
Phrase 1	55.7	60.0	62.9
Phrase 2	31.43	32.9	32.9
Phrase 3	34.9	36.5	38.9

Table 2.1: Percentage correct classification of raw unfiltered data using the simple Euclidean distance measure. Although low-scoring, the results demonstrate that good separation of users can be achieved. With 14 participants, the *a priori* probability is 7.1%.

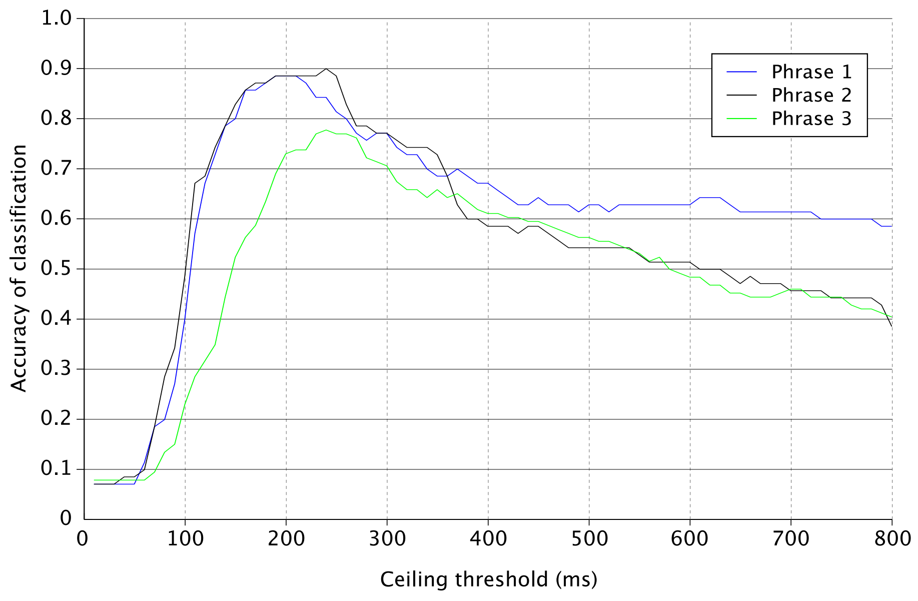
The proportions of correct classification using the two styles of metric (hold and inter-key) with this classifier, using the raw experimental data, are given in Table 2.1. In addition, the two feature vectors (hold and inter-key timings) were concatenated to provide a richer combined vector, and the effect of applying the classifier to this new vector was also calculated; the results of this are included in the table, labelled as "Combined metric". Initially, these results were considered disappointingly low, but it must be remembered that the chance rate is only 7%. From this perspective, the ability to select the correct typist from a set of 14 this proportion of the time is more impressive.

By inspection, however, it was noted that the matrices contained a minority of extremely high values corresponding to long pauses, probably caused by thinking time or distraction. In these cases, the magnitude of the peak in time delay was sufficient to prevent the distance-matching technique from selecting the data as a good fit, even if the rest of the feature vector was well matched to the test case.

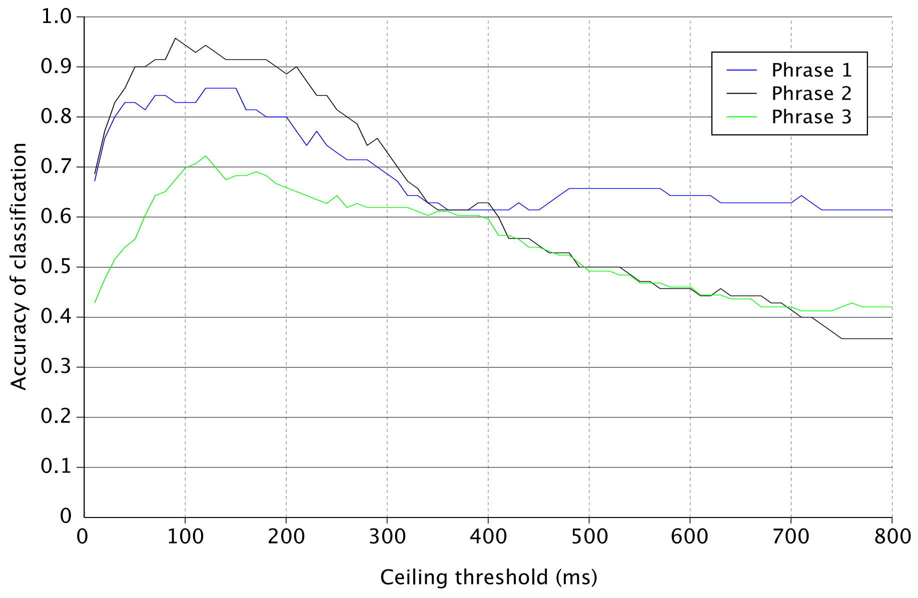
2.3 Filtering technique

In an attempt to reduce the swamping effect of long pauses during typing, an artificial ceiling was applied to the measurements in the gallery of test vectors, such that any value exceeding a given ceiling would be reduced to equality with that ceiling value. To assess the effectiveness of the ceiling technique the classification tests were performed repeatedly, varying the ceiling from 10 ms to 800 ms. The results are shown in Figures 2.2(a) and 2.2(b). It should be noted that although the results peak at very high recognition rates, it would not be practical to set the ceiling filter this low in the field, since this would most likely represent overfitting to the current data set. To maintain generality and retain as much useful data as possible while rejecting long pauses, if a threshold were used in a practical system it is suggested that it would need to be set higher than fits the 'best' results shown here.

We may note that in the case of the hold times metric, the effect of reducing the threshold near zero produces classification results corresponding to the level of pure chance, since all of the useful information has been removed from the data. In contrast, the effect on the inter-key metric is to reduce its effectiveness partially but not completely: this is



(a) Hold times



(b) Inter-key times

FIGURE 2.2: Ceiling filter effect on accuracy for (a) hold times and (b) inter-key times.

	Hold times (%)	Inter-key times (%)	Combined metric (%)
Phrase 1	81.4	72.9	85.7
Phrase 2	88.6	81.4	87.1
Phrase 3	71.4	70.6	75.4
Phrases 1 and 2			95.7

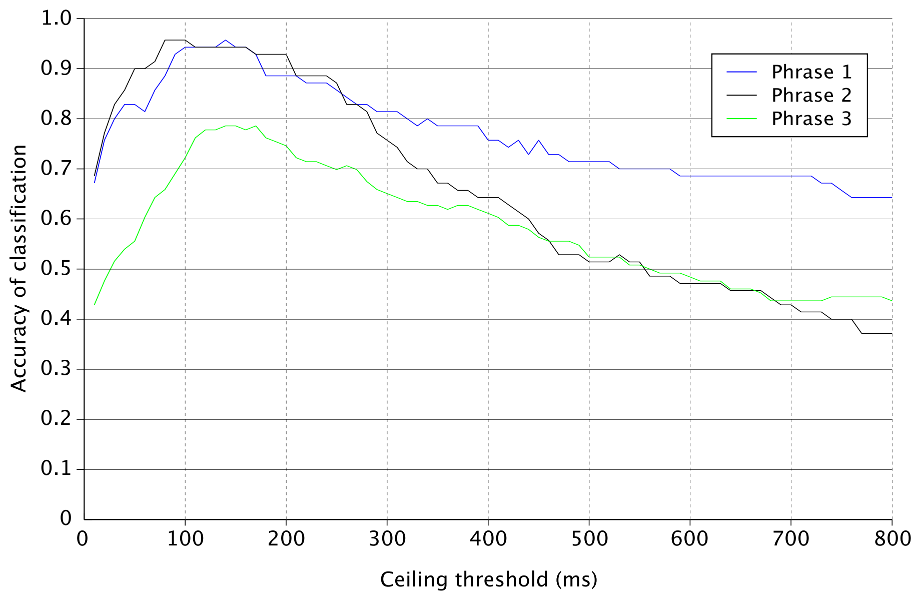
Table 2.2: Percentage correct classification using the simple Euclidean distance measure with data thresholded at 250 ms.

because inter-key times may be negative and so an upper limit of zero will not remove all of the useful classification information.

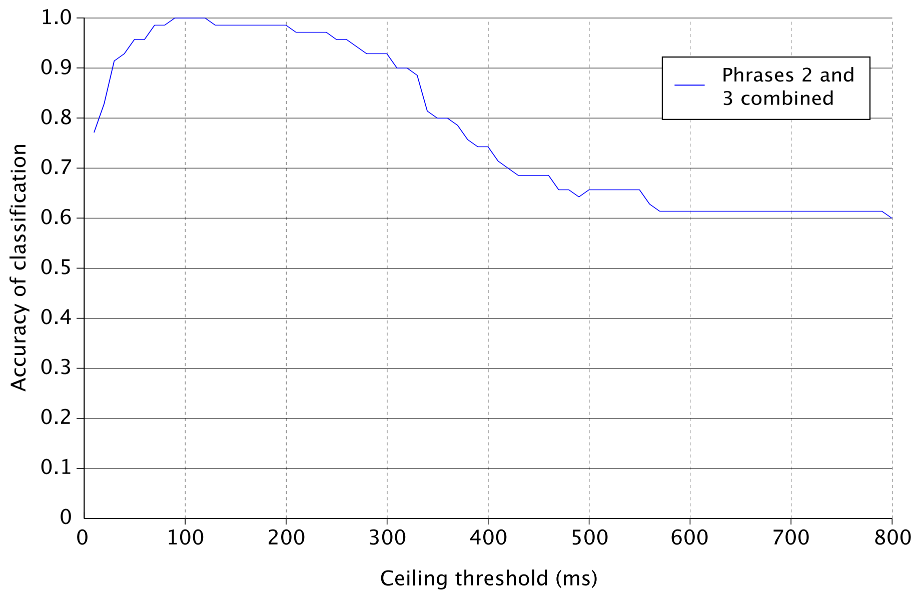
The ceiling test was also performed using the combined feature vectors of hold time and inter-key time, producing the classification accuracies shown in Figure 2.3(a). A further set of calculations were also performed to demonstrate the classification accuracy obtained by combining the data from both phrase 1 and phrase 2 (by vector concatenation). The results of this combination can be seen in Figure 2.3(b). Following inspection of these results, a fixed ceiling at 250 ms was used to obtain the figures shown in Table 2.2. It is important to note, however, that although these conservatively filtered results are still not as high as some given in other literature, they are not intended as a demonstration of the efficacy of this naive classifier. The purpose of these results is to confirm that there is statistically useful information in the data obtained from keystroke timing. Since using the combined metric on the concatenated vector allows us to achieve up to 100% classification, this is clearly the case.

That such a measure should improve the results is perhaps unsurprising. I is suggested that most of the longer delays are caused by the neurological rather than the physiological aspects of the latency. Card *et al.* (1980) propose that the time taken for the performance of human-computer interaction tasks is composed of two constituents, the time taken to think about the performance, and the time to carry it out. This effect is evident when performing the timing analysis of the keystroke data. For the purposes of this biometric the neurological aspect is essentially random noise; in order to extract the useful information representing the (largely invariant) physiology of the user, attempts to filter out the effect of the larger cognitive delays seem well motivated.

In addition, considering the physiological component of our data stream to be the useful part may give us further insight into static authentication. Repeated training on a fixed phrase tends to show an increasingly stable and self-correlating timing pattern as the typing becomes a matter of 'muscle memory'. It is suggested that as the user learns the phrase, the contribution of cognitive delays to the timing, caused by recalling the order of the keystrokes in the phrase and the preparation time in placing the fingers correctly, will decrease significantly. Consequently, this improves the stability of the metric.



(a) Combined metric



(b) Combined metric with phrases 1 and 2 concatenated

FIGURE 2.3: Ceiling filter effect on accuracy for (a) the combined metric and (b) combined metric with Phrase 1 and Phrase 2 concatenated.

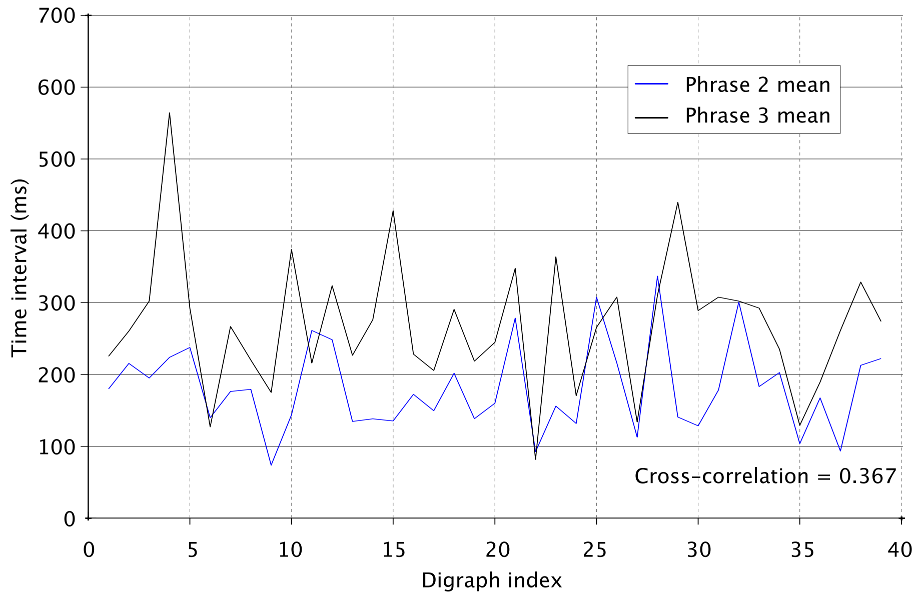
It is with this in mind that the question was also posed: do non-contextual digraph profiles really provide us with sufficient information for an optimal profiling process for dynamic authentication? Since dynamic authentication data are likely to be highly influenced by cognitive 'noise' (since there is no training process for the user), does storing only digraphs with no contextual information allow us to mitigate this sufficiently?

2.4 Digraphs and context

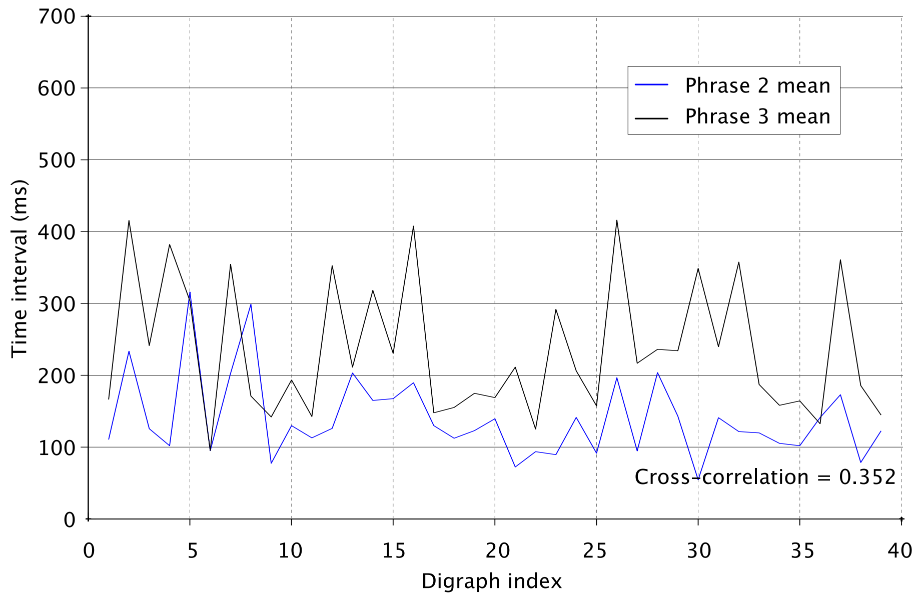
To provide some insight into the level of variance that might be expected between identical digraphs in significantly different contexts, a comparison by inspection was made of the two phrases in the study '`THERE ARE MANY DIFFERENT TYPING PATTERNS`' and '`THERENT TYPINY DIFFERNG PARERE MATTE ANS`'. The two phrases contain an identical set of digraphs, but in the latter case rather than providing a simple set of English words, the digraphs manifest themselves as apparent nonsense (though it may be noted that the last three words are an Italian/Latin word, an uncommon English word and a French word). The cognitive disorder for this phrase is rather higher, with the intention of simulating circumstances under which a typist might require an increased level of consideration about their typing. If the increased neurological influence on the typing were uniform across the phrase and if we could also assume that digraphs provide sufficient information for use to consider them out of context, we might expect the two sets of digraphs to show reasonable similarity in pattern. This should be sufficient for matching techniques to be applied with a reasonable level of success.

To test for this, the inter-key data for the two phrases were sorted into alphabetical order by digraph, a ceiling filter was applied at 600 ms to reduce the noise and a side-by-side comparison of the two phrases was made for different users. Some trends appear to be shared between the sets, in proportion to a more cautious typing rate for the more difficult phrase. However, other characteristics of the typing patterns appear to be in opposition to each other. Examples of this for two typical cases are given in Figure 2.4. We can see from the graphs that the relationship between digraphs is not consistent across the two phrases, suggesting that the latencies associated with each digraph are dependent on context.

To illustrate this further, we used the vector cross-correlation capabilities of MATLAB to calculate the Pearson correlation coefficient (Rodgers and Nicewander, 1988), a measure of the relationship in variability of two data sets. The Pearson coefficient is a real number in the range $[-1.0, 1.0]$ which represents either a negative or positive correlation between the data. Strictly, the calculation only remains invariant (highly correlated) where a linear relationship exists between the sets, and we have not attempted to establish that typing latencies are subject to linear behaviour (indeed, this is improbable).



(a) Subject X



(b) Subject Y

FIGURE 2.4: Example of comparison between Phrase 2 and Phrase 3 digraphs for two typical participants. The correlation between the two phrases is indicated in each case.

Nevertheless, we can use the measure here to note that there is considerable difference between the data gathered from the two phrases. If digraph latencies were insensitive to context, we would expect a much higher coefficient than the 0.37 here, which shows only a limited positive relationship between the latencies of the two phrases. While only two example graphs are shown here, this poor correlation is consistent throughout the data sets.

It is proposed that there are two factors which may be causing this poor correlation. Firstly, if we accept that performance of a task may be separated into the time taken for its neurological component and that for its physiological affectation, we might expect the disordered phrase to be typed more slowly on average. The phrase is less easily remembered, since the patterns are less common, and therefore more neurological 'effort' is required to type it accurately. However, we would also expect that over the course of the repetitions required for ten successful entries, the random noise part of this component would be smoothed to some extent. Thus we might expect that the profile of the disordered phrase would tend towards that of the more ordered phrase. This seems not to have happened, however, which suggests that there may be some reason, physiological, neurological—or most likely both—that the phrase is being approached differently in its performance. It is perhaps not unreasonable, therefore, to posit that considering hold times for only single keys, or inter-key times for only digraphs, fails to incorporate sufficient notion of the 'context' in which unigraphs or digraphs are entered. As a result, the later BAKER work (see Chapter 3) extends its data capture to include additional contextual information by gathering trigraphs and quadgraphs.

Chapter 3

Development of BAKER

In this chapter, we examine the development process for the BAKER platform, both in terms of the technology and the ethics. We also discuss some of the technical challenges which arose during and subsequent to the initial deployment.

The BAKER system was developed during the course of this research as a means of collecting timing statistics from the normal operation of computers running the Microsoft Windows operating system. The software is loaded as soon as the user logs on to the computer and it operates as a background task. During the operation of the computer, BAKER collects both hold-time and inter-key time statistics for all multigraphs entered (with the exception of those entered into password boxes—see Sections 3.1 and 3.3.2).

3.1 Requirements for keystroke collection

Possibly the greatest challenge encountered while developing BAKER was apparent even before the first line of code was written. The problem was this: how is it possible to gather statistics from people's free text typing and still assure them of a suitable level of privacy?

While it might be ideal from a statistical point of view to collect a raw key log from each user complete with timestamps for all key-up and key-down events, few participants would agree to such a serious risk to their privacy and security! Even if we could guarantee the users that *we* would not read back the chronological log of their keystrokes, we felt that the very notion of handling files which could contain the users' passwords in raw form would make the project unworkable and unlikely to receive ethics approval. It became necessary, therefore, to find a method of data collection that would strike a balance between its statistical value and the level of trust required by the end users.

The first step to making the exercise practical was to dispense with the luxury of wider chronology, at least at the level of words and sentences. The idea of using trigraphs and quadgraphs was a natural extension of the digraph analyses presented in earlier literature and in the preliminary study.

As discussed in the fixed text analysis, there was a concern that the use of digraphs for determining inter-key intervals might suffer from a lack of contextual information. Similarly examining hold times only in terms of the keypress in question is likely to have a much wider variance than when the values are separated by the context in which they were generated. As a result, the decision was made to collect quadgraphs for inter-key times, providing an additional preceding character (related to the position the hands were in prior to the middle digraph) and also a succeeding character (related to the travel of the hands of an experienced typist preparing for the subsequent keystrokes). The same applies to the collection of trigraphs for hold times. By storing this information in the matrices rather than as an ordered log, we lose the ability to recover a chronological log of keystrokes, thereby improving the privacy of the data.

Of course, this necessarily still allows some information to be recovered from the files about what has (or indeed has not) been typed. If, for example, a matrix file contains no instances of the Y keystroke, we can assume that the user has not entered the URL `www.typinganalysis.net` during their session. Furthermore, a matrix with very few keystrokes in it may make attempts to reconstruct the generating sequence somewhat more feasible. Therefore, to provide additional protection to the most sensitive type of input, extra filtering code was added to the software. As discussed above, the set of keys monitored by the software is limited, and so strong passwords (containing non-alphabetic characters) are likely to be rejected as a result of their nature. To provide further reassurance, however, filtering was added to ensure that those keystrokes destined for password boxes (traditionally masked with asterisks or filled circles in the Windows environment, and identified by their window class) are explicitly excluded from the input to the analyser.

3.2 Resulting functionality and structure of BAKER

The timing statistics gathered by the software are stored as large multi-dimensional matrix representations of the possible key sequences entered at the keyboard. In the case of hold time analysis, the matrix used is 3-dimensional and collects trigraph data; Figure 3.1 shows a simplified schematic representation of this. The inter-key time analysis uses quadgraphs and so the matrix is 4-dimensional (and therefore not shown here). In the actual implementation, the data collector recognises and stores information from the alphabet keys, the comma, full stop and space keys, and the operation of `Shift` and `Backspace`. All other keystroke information is discarded. This helps to keep the size of the matrices down and also adds additional reassurance of privacy since good passwords tend to (or perhaps *should*) include non-alphabetic characters. It was felt that the

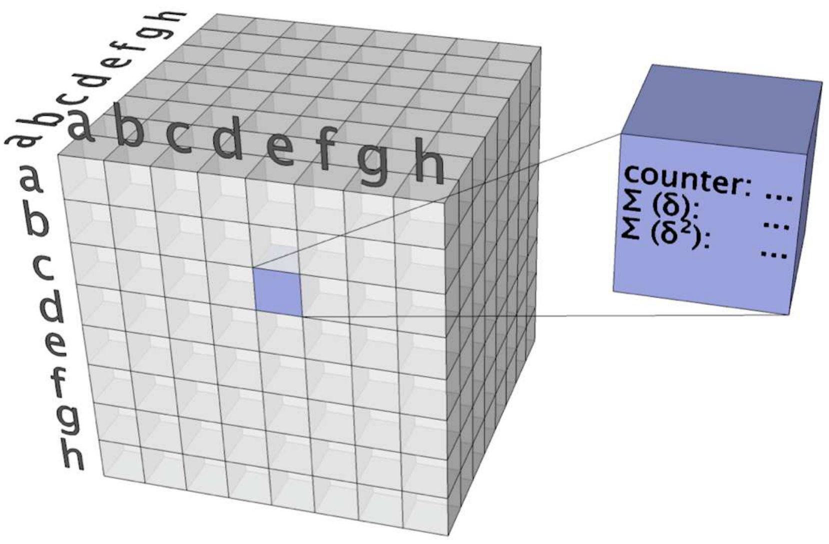


FIGURE 3.1: Simplified schematic representation of the matrix structure for hold times. We may imagine that the small cube indicated represents the storage space for the hold time of the *A* key (shown in the dimension extending away form the viewer), in the case that a *C* has been entered immediately beforehand (vertical) and an *E* is to be entered directly afterwards (horizontal). For clarity, we have represented the alphabet as if it only contains the first eight letters.

operation of the numeric keys, for example, was sufficiently infrequent—and possibly too atypical—to add significant value to the study.

The nature of the matrices is such that every cell contains a data structure with sufficient information to record a running count, mean and variance for its corresponding multigraph (see section 3.3.3 for a more detailed description of the storage system). As well as storing this information in an unfiltered form, each matrix has three other 'sibling' matrices. These siblings are generated from the same source data, but have had filtering applied, either to discard multigraphs where the `Shift` key has been held down, or where the keystrokes have not been destined for a known word processing application, or with both of these filters applied.

In addition to the timing statistics, several other types of data are also gathered during the study. Some of the information is purely objective, for example participants are requested to provide information on the layout of the keyboard used on a target system and, where possible, the FCC ID of the keyboard model. Other information requested is subjective but tangible or verifiable, such as the nature of the environment in which the system is operated (such as an office or bedroom) and the tactile quality of the keyboard. In an extended study it may be that trends can be identified based on this information—for example whether or not working environments with high typical levels of interruption provide more erratic typing results.

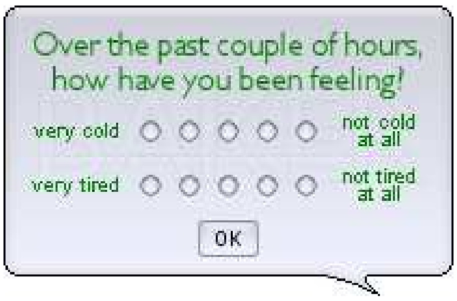


FIGURE 3.2: Example of the Feelings Query pop-up window.

The other type of data gathered is entirely subjective, and it may or may not prove useful. Periodically, the software displays a pop-up window in the lower right corner of the screen. This window invites the user to enter a rating of how fatigued they have been feeling over the last few hours, and similarly how cold they have been feeling, on an arbitrary 5-point scale ranging from 'very' to 'not at all'. As an additional point of interest, the intention was to allow comparison between the typing statistics and these 'Feelings Queries' reports, to see if there is any noticeable degradation of typing performance under conditions of reported psychological or environmental discomfort.

All data gathered by the system are time-sliced in three hour periods, allowing for analysis of temporal variations. Further discussion of the time-slicing and data management systems is given in section 3.3.

In addition to its data gathering capabilities and the ability to automate uploads of the gathered data to a central server, the software has also been designed with an architecture which should allow for automatic updates. One of the supported jobs handled by the task system (see section 3.3.7) is that of downloading new modules for future revisions of the software. It is hoped that this will provide flexibility as the study develops.

3.3 Detailed system design

Many of the challenges faced during the design of the software were technical. In particular, differences between versions of Windows (typically anomalous behaviours in Windows 98) and subtle inaccuracies and ambiguities in the platform's API documentation have made the development process additionally difficult. Since the software is designed to hook the operating system's behaviour at a relatively low level, errors in its core behaviour have the potential to render the system unusable. This has resulted in the need for very careful design, particularly at the hook level (see Section 3.3.1).

This section documents the design of the BAKER system and its component modules. A copy of the resulting code is included in Appendix D (bound separately).

3.3.1 Keyboard hooking library

At the heart of the BAKER software is the hooking module designed to monitor keystroke messages at the operating system level. For keystrokes to be intercepted, a custom function must be added in to the Windows 'hook chain' for the relevant I/O event types. It is a constraint of the design of Windows that if such a hook is to be global, i.e., if it must be able to monitor keystrokes outside the context of the host application, then it must be contained in an external library (a Dynamic Link Library, or DLL).

The requirement of externalising the hook function complicates the design of the software in several ways. Firstly, since the DLL is a separate file to the main executable, it can make it more difficult to guarantee the integrity of the installed software as the number of logical components must increase. As regards file handling, this difficulty is overcome in the BAKER system by embedding the DLL as an internal resource inside the executable file, and then enabling the core executable to extract this resource to a temporary file ready for use at runtime. Achieving this requires some minor technical adaptations to the way in which the DLL functions are called by the software. 'Hard-wiring' the function calls in the usual manner would cause the loading of the software to fail: the operating system would determine the necessary external DLL file to be 'missing' at the start of the execution and terminate the process. With the appropriate late linking, however, this method successfully overcomes the need for delivering separate files to the user.

Nevertheless, the difficulties associated with designing an effective, transparent hooking procedure are far from insignificant. Most importantly, the code inside the core hook function must be not only robust, but also speedy, as it is executed twice with every new keystroke (once on the key down event, and once on the key up event). This means that handling the processing and storage of keystrokes inside the hook function would be impractical. Not only would it be unwise to place I/O calls (such as hard drive access) inside the hook chain, but also if any future revisions of the software were to require increased processing, this would risk affecting the responsiveness of the system.

It is important to note that while the hook function is executing, the effect of the key press (or release) on the rest of the system is necessarily delayed. As a result, the sole purpose of the hook function is to gather the data on the nature of the key event, insert the information into a suitable data structure, and place that data unit into a transition buffer ready for the main executable to handle as appropriate. By performing the more extensive data processing in the main executable, not in the hook function, any processing delays affect only the BAKER software and not the entire system. The only exception to this performance isolation occurs in the unlikely event that the transition buffer becomes full. Under these circumstances, the software is designed so that after a brief delay, a timeout occurs, the data unit is discarded by the software, and the operating system is allowed to continue processing the key event as normal. By using a large enough buffer and ensuring that the receiving software is fast enough, this situation can be avoided.

The case of automatic key repeat is also something which must be considered carefully. In general, the circumstances under which key repeat occurs are such that it would be unhelpful to include the events in the typing analysis study. Nevertheless, the repeated events are delivered to the hook function by the operating system, and so it is necessary to filter them out at this stage.

The other significant burden on the design of the hook function is that the events may occur in the context of any thread of any process in the multitasking environment. This means that the software design must be thread-safe as it will almost certainly be called from a wide variety of threads across the system. Consequently, the buffer into which it places its data units must also be constructed in a thread-safe manner, as must the software on the receiving end. This presents us with a classic producer-consumer problem, solved through the use of a ring buffer protected by operating system mutexes (Andrews and Schneider, 1983). Without discussing the science behind thread-safe buffer design here, it is sufficient to say that great care must be taken in design to prevent deadlock, where the software as a whole—and possibly the entire keyboard subsystem—could stop responding. Appropriately implemented, however, such protected code can prevent inconsistencies, and guard against lost events where two threads attempt to manipulate the buffer concurrently. These hurdles were overcome very effectively in the BAKER design.

Each data unit inserted into the buffer contains several pieces of information gathered at the time of the keystroke. Firstly, the 'key code' is stored, which identifies which key on the keyboard initiated the event. Added to this code is an identifier which denotes whether or not any modifier keys were down at the time of the event (i.e., the `Shift`, `Ctrl` or `Alt` keys). Secondly, the operating system's high performance frequency counter is queried (and subsequently reduced in resolution so that its accuracy is more appropriate for the study) to provide an accurate relative local-system timestamp with a resolution of 0.1 ms. This is also placed into the data structure. Thirdly, a value is set in the structure recording the handle of the foreground window at the time of the event (enabling subsequent filtering according to the foreground process). Finally, a flag is set in the structure to denote whether this was an `Up` event (key release) rather than a `Down` event (key press).

In addition to the hook function itself, the DLL also contains code for various housekeeping procedures, including those necessary for installing and uninstalling the hook on demand.

3.3.2 Synchronous event handler

The processing of events gathered by the hooking DLL begins in the synchronous event handler, part of the main executable. When the software is started, it is the responsibility of this event handling subsystem to allocate shared memory for the thread-safe event buffer (hereafter the 'Event buffer'), to prepare the semaphores and mutexes which protect this buffer, to launch the hooking DLL and to initiate the installation of the system-wide hooks.

Once the hooks have been installed successfully, the Event buffer will start receiving the data from any keystrokes which occur. As soon as the hooking completes, the event handler spawns a new thread—the consumer thread in our producer-consumer problem. This thread is at the core of the Synchronous event subsystem. Its purpose is to collect events from the Event buffer (which have been passed in by the hooking DLL), and to perform the process referred to as Grouping.

Prior to Grouping, the format of the data gathered by the hooking subsystem is simply a stream of Up and Down events with no superstructure to correlate them. It is the job of the Synchronous event handler subsystem to sort these events so that latencies for individual keys, and the context in which the keystrokes occurred, may be derived.

This is where the process of building sets of trigraphs (for hold time analysis) and quadgraphs (for inter-key time analysis) takes place. The information which reaches the subsequent storage subsystem must be in the form: 'Key Y was held for n milliseconds when preceded by key X and succeeded by key Z' for hold time analysis, and in equivalent form for inter-key analysis. This means that the processing of individual keystrokes must be delayed until the successive element of their context is determined. To achieve this delaying and sorting, a secondary buffer known as the Grouping buffer is used. The details of the Grouping process are given in Appendix A.

Due to the nature of the incoming event stream, it is theoretically possible (although rare) for the Grouping buffer to become full. We can imagine, for instance, a situation in which a key is held down while a very large number of others are pressed and released. Clearly, to obtain the hold time, the processing of the `Down` event from this held key must be delayed until its corresponding `Up` event occurs on key release. The `Down` event will therefore not be released from the buffer, and nor will those keys which depend on its identifier for their context. Because the Grouping algorithm clears from the front of the buffer, this can delay the emptying of the buffer and cause the events to back up. To prevent this becoming a problem, a large Grouping buffer is used, and if the software detects a situation where the buffer is almost full, it will elect to discard any 'sticky' events from the Grouping buffer so that processing may be allowed to continue.

In addition, the software keeps track of the peak level of the Grouping buffer, and the number of events dropped, and includes this information in its status reports to the main server. A last resort mechanism is also included which is able to detect any unexpected ‘runaway’ conditions: if the Grouping buffer elects to purge itself more than 5 times in 25 events, the software will assume that the monitoring subsystem has entered an indeterminate state due to a fault, and it will exit and reload BAKER. Originally included as a debugging tool during development, this feature has been left in as a safety measure for the released copies of the software.

The other very important task of the Synchronous event handler is to perform event filtering, removing data relating to keypresses which must never reach the Grouping buffer—discarding that information which the processing part of the software must not be allowed to ‘see’. Specifically, it is the event handler which determines the nature of the windowing component responsible for the event, and tests to see if it is a password entry field. If it is, the keystroke is instantly discarded (for reasons of privacy) and the event is not used in Grouping. This presents one small additional challenge: if any keys are held down while the OS switches focus to a password field, this would result in a `Down` event reaching the Grouping buffer (since it occurred inside a non-protected field) and the `Up` event being rejected (since it occurred inside a password box). This would lead to the buffer filling up as described above, as the `Down` event would block on an `Up` that it would never receive. To protect against this, additional code was added to allow `Up` events to propagate to the buffer handler if (and only if) they have matching `Down` events waiting for them in the buffer. This poses no additional security risk. Once the events have been entered into the Grouping buffer, the event subsystem must then scan that buffer to detect any completed events which have fulfilled their context requirements. At this stage, they are packaged into a unit containing the keystroke (or keystrokes for inter-key pairs) to which the event refers and also the ‘contextual’ information, the latency, and the originating window handle. The events which are no longer needed as context for subsequent keystrokes are then marked for deletion, and the processed unit is forwarded to the storage subsystem.

3.3.3 Storage core

The storage subsystem (known internally as the ‘datacore’ module) is the central point responsible for handling the majority of the data structure formats used throughout the BAKER software. Most importantly, it contains the processing code designed to receive sets of trigraph and quadgraph data, to filter and categorise them as appropriate, and to enter them into the ‘matrix maps’ which are the end product of the software.

In the early versions of BAKER, the ultimate storage units were the large, sparse 3- and 4-dimensional matrices representing all possible trigraph/quadgraph combinations (for hold time and inter-key time respectively). Conceptually, the storage is still equivalent to this, and each matrix cell still maintains the original format of an incremental counter, a cumulative latency value used to calculate a running mean, and a cumulative ‘sum of squares' value used in combination with the mean to provide a running variance measure. However, improvements to the efficiency of the storage code have been made possible, partly due to the modular nature of the software design. The resulting files are now considerably smaller, since rather than containing the fully populated matrix, they contain a condensed 'matrix map' representing only those cells in the matrix that are non-zero. When the files are ready for processing, they may be expanded to full matrix form.

The first thing that the storage system does upon receiving a data unit is to check that the keys in the set match with those in the table of allowed values. Numeric and extended and punctuation keys, for example, are rejected at this stage. Once this is done, the latency is compared against the preset threshold level. Early versions of BAKER contained only one threshold, set conservatively high based on the results of the preliminary experiments. Adopting a single threshold was a necessary consequence of the early, bulky matrix files. Since the improvements in space efficiency, however, a later revision of BAKER also included the facility for setting up to ten threshold levels; were this to be used in a wide study, the earlier work on thresholding static analysis could be extended to determine optimal rejection levels for dynamic processing (and perhaps allow investigation of the feasibility of setting per-user thresholds). In the preliminary static analysis experiments, it was necessary to apply a ceiling to the data to simulate thresholding, since it is not possible to 'remove' individual elements from the fixed data stream. In contrast, the dynamic system allows those latencies affected by neurological delays and 'pauses for thought' to be rejected altogether.

A second element of the processing in the storage core is controlled by the key modifier flags passed through from the hooking subsystem. Two separate sets of matrix maps are kept: one set allows shifted (capitalised) letters to be included in the matrix totals, one set rejects all shifted keystrokes. Due to the linear nature of the matrix storage, it would also be possible to subtract the latter from the former to obtain data related only to shifted keys.

A further element of processing undertaken in the storage core concerns the nature of the foreground window responsible for the event. By enumerating the various windows in the system and determining their owning processes, it is possible to obtain the name of the executable responsible for their creation. This enumeration is a fairly laborious process, but the foreground window tends to change relatively infrequently and so a system of result caching was implemented to avoid unnecessary repetition of this function. Once the name of the process is known, it is compared against a list of known word processors and text editors (such as Notepad, Microsoft Word and so on), and again the storage files are split into two sets—those containing keystrokes irrespective of their window, and those containing only keystrokes who events were destined for word processors. And again, the linear nature of the matrices means that the results for keystrokes occurring outside a word processor environment may be obtained through simple matrix subtraction. Once the data are thresholded and filtered to determine which of the various matrix files are applicable (based on shifting, originating process and whether the data unit represents an inter-key quadgraph or a hold time trigraph) the information is passed to the relevant file handler functions. These determine the appropriate location in the corresponding map file, perform some trivial encryption (essentially obfuscation) and add the information to the map.

The space-saving 'mapping' process for representing the matrices in condensed form is controlled by a set of hash tables corresponding to the contents of the currently active map files. When a trigraph or quadgraph (a multigraph) is marked for storage in a file, the handler attempts to look up the key combination in the hash table. If it is present, the data stored in the hash table will represent the position in the file used for storing that multigraph, and so the data can be entered directly. If it is absent from the table, this indicates that it is a previously unseen multigraph and so it must be added to the map file in a new location, and the matrix file must be updated. It is important that these hash tables are populated correctly on initialisation if files built previously are to be accessed.

The storage core also has a variety of other responsibilities. Partly a hang-over from the original large matrix file design, the software still employs the technique of converting all data files to compressed (gzip) archives prior to uploading: this increases the efficiency of the transfers and reduces the storage requirements on the remote server. These compression routines are managed from within the datacore module. Additionally, it is the storage core that is responsible for collecting the various information which lives in the file header, including the current user of the system. Furthermore, since the study is time-sliced in sections of three hours, the task of deciding which file set should be currently in use is handled by this module. When a time-slice expires, the storage core communicates with the task handler subsystem (detailed in section 3.3.7) to co-ordinate moving the data files into the 'pending' folder ready for upload to the server, triggering their compression, and scheduling the 'Query Feelings' mechanism (see section 3.2). The datacore module also contains code to open and close the various files on startup, on shutdown and on time-slice switch, as well as to manage the various file handles.

3.3.4 Web link module

The 'weblink' module is responsible for handling all communications with the remote BAKER server. It was decided that all data transferred between the client software and the server would be via HTTP transactions on TCP port 80 (hence the 'web' element of the module name). This provides two important benefits. Firstly, it minimises potential problems with client firewalls, since almost all consumer firewall installations tend to treat port 80 traffic as benign. Secondly, and perhaps even more importantly, this allows the use of server software with a known heritage and wide user base. Rather than having to develop a suitable network-enabled software service for the server platform, which is frequently a cause of errors and security problems in the absence of a lengthy testing period, it is simpler, faster and safer to use known web server software in which there is already reasonable confidence. As a result, the weblink module contains various functions responsible for assembling and performing different HTTP GET and POST transactions and for processing the results, as well as for lower level networking functions such as DNS lookup.

When the module is initialised, it spawns the main weblink thread and sets it to a priority class below normal so that it may operate in the background with minimal impact on the use of the system. The outer loop of this thread typically cycles every 50 seconds, although this delay may be remotely reconfigured. Each cycle is broken down into several segments (a fixed value of four segments per cycle in the current version of BAKER). During each segment, the software performs a series of tests to determine whether the local system has an active connection to the internet and whether the remote server can be reached. This is used to control the online/offline status of the software so that no polling of the remote server is attempted if no internet connection is found. Once this is determined, the weblink thread enters a sleep state, yielding its processor time. On the final segment of each cycle, the weblink thread polls the remote server.

The polling process incorporates four key stages:

1. The session handler establishes the validity of the local user's security credentials and either rejects the access attempt (in which case the client software will display a password dialog to the user and suspend weblink activities until valid credentials are entered) or accepts the authentication and marks the session identifier (established at the start of the exchange) as valid for that user.

2. The client software uploads a status report containing information about the state of the local BAKER system (total number of keystrokes processed, number of keystrokes ignored, last peak size of the Grouping buffer, and various configuration data).

3. The remote system responds with a list of any tasks to be added to the local task handler (see 3.3.7).

4. The client sends a command to terminate the authenticated session and discards its session identifier.

The relationship between the contents of the status report and the task list response is critical. The nature of the task handler is discussed in more detail in section 3.3.7, but one of the tasks that the local system can be requested to perform is the uploading of any pending matrix map files. When the weblink polls the server, it reports the total number of map files ready (and compressed) for upload. At this point, the server is able to decide if it is ready for the files to be sent. Currently, the algorithm used by the server to decide whether or not to begin the transfer is based on a simple threshold, but it would be possible to incorporate some measure of the current server load to afford more sophisticated load balancing capabilities. When the upload task is initiated, the task handler uses further functions inside the weblink module to transfer the files—this is performed asynchronously with respect to the main weblink thread (since it is executed within the separate context of the task handler). Only when each file has been successfully uploaded is it removed from the 'pending' directory on the local machine. This approach therefore affords some measure of fault tolerance, since the client will never upload data until it is asked, and any failure in the transfer will result in the local copy remaining cached until the next upload request.

In addition to the ability to perform asynchronous uploads of data files, the weblink module also manages part of the notifications system. When a local task completes its execution, the task handler is designed to notify the originator of the task (where appropriate). If the originator was the remote server, it is the responsibility of the weblink module to connect to the server and send the details of the notification: the ID of the task, whether its completion code indicates success or failure, and any additional information. This same notification system is used in later versions of BAKER to handle other trivial data transfers (such as the results of feelings queries—see Section 3.2).

3.3.5 BAKER server

One of the most important parts of the BAKER system with respect to data collection is the server-side software. The client program is designed to communicate over the internet with a central server, via HTTP transactions on TCP port 80, the standard web services port. The reasons for using a web-based platform are outlined in the weblink description (see section 3.3.4); essentially this provides greater compatibility with target systems and avoids the need for developing custom service software.

Two web servers are used as part of the project. One handles the non-technical requirements and is responsible for serving the public face of the BAKER project, its site at `http://www.typinganalysis.net`. The other is the data server used to communicate with the remote client software, and this responsibility is undertaken by the main server for user web pages at the School of Electronics and Computer Science, University of Southampton. The reason for passing all experimental data via this second server is that the main database is homed on a third host (the development server) which is inside the ECS local network. For security reasons, only those machines already inside the network are permitted to communicate directly with the database server. Indeed, the chosen local configuration means that only the ECS web server itself is permitted to make inbound connections to the database. As a result, any information which must be presented by the main `typinganalysis` site but which is held by the database server must have the requests proxied to the ECS system (for an example, see the discussion of the user control panel in section 3.3.9).

The server-side software is written in PHP, a robust and mature scripting language designed for web development. The ease with which database-driven web applications can be built in PHP, along with its built-in support for session management, means that it has been possible to handle not only all data uploads but also all user and account control from within the script files. Through careful development and deployment alongside the client software, it has been possible to create scripting modules which undertake user authentication, the management of uploaded data files, control of the remote 'synchro' modules (section 3.3.6) and version and task management. The scripts also collect all status reports and notifications from the clients and with appropriate data processing allow messages to be sent via e-mail to the developer, indicating the status of the system and warning of any anomalous client behaviour.

The database design for the back-end MySQL server consists of nine tables responsible for different aspects of the system's operation. The details of these tables and their relationships is given in Appendix B. In the early releases of the software, uploaded matrix map files were not stored in the database table prepared for the purpose, but saved as files within allocated space in the school file storage system, again protected inside the ECS network. Later revisions of the software were updated for use with database-backed storage.

All client-server authentication procedures are completed using MD5 hashing and challenge-response mechanisms to avoid the transmission of plain-text passwords over the internet. The implementation of a public-key architecture may have been possible, but for the levels of risk involved in the project and the amount of effort required to accomplish this, it was judged to be unnecessary.

3.3.6 Synchroniser

To ensure that the timestamps on the various uploaded data sets are consistent, and to prevent clock drift or manual alteration from skewing or mixing the order of the time-sliced files, BAKER includes a dedicated module for clock handling: the 'synchro' module. The micro-timing inside the hooking DLL uses an accurate time measure to determine relative latency and this cannot be affected by the end user. In contrast, the system clock which governs the data handling systems (the 'time of day' clock) gives an absolute measure which can be altered by the user and therefore cannot be trusted.

To overcome this difficulty, all calls to obtain the current time of day are passed through the synchro module. On initialisation of the software, the communication between the weblink module and the remote server is used to calibrate the starting clock offset used in the synchroniser: the difference between the clock on the BAKER server (which is used as a reference) and that of the local system is stored and used in subsequent calculations. Each successive weblink communication is also timestamped, and in this way (subject to the effect of network latencies) the software can keep track of the approximate time on the server's clock, and use this to determine the appropriate labelling and time-slicing of the matrix map files.

It is important to note, however, that this periodic (typically 50 s) polling of the server is insufficient to account for manual local clock changes. We can imagine a situation, for example, where a local user erroneously adjusts their system time forward by 12 hours. Without additional protective measures, this would result in the reported synchro time moving forward 12 hours until corrected by the next web poll. If a keystroke event occurred before correction, the datacore would check the synchro time, determine that the current three-hour time slice had ended, and produce a new empty set of matrix map files with a time-code some 12 hours in the future. On the subsequent web poll the synchro time would move back to the correct value, but until the software was restarted, the system would continue to use the 'future' matrix maps, since it is designed to ignore backward changes in time (and so protect against fluctuation at three-hour boundary points). Furthermore, a software restart during the intervening period could result in the production of data sets in the wrong order, mislabelled, and possibly even sets with duplicated time-codes. Additional protection is required, therefore, to guard against the effects of manual clock changes.

To accomplish this clock protection, guard code is included in the synchro module which executes every time a timestamp request is made. As well as tracking the local system time and the time of the remote server, the module also uses a second local clock—the operating system's 'tick count' (a low resolution, stable counter which cannot be manually altered but which only provides a relative time measure)—as a clock 'follower'. Each time the timestamp request function is called, the guard code compares the current system time with the expected system time based on the previous time and how much the tick count has increased. If there is a discrepancy between these two local clocks of greater than three seconds, it assumes there must have been a manual change to the system time, and so forces a re-sync with the remote server before continuing.

3.3.7 Task handler

Vital to the co-ordination of BAKER's various components is the task handler module. Its purpose is to manage the execution of a variety of tasks according to their priority and timing schedules. The list of all pending tasks is kept in a data file stored on the local hard drive. By keeping the information in a file rather than in memory, and only removing entries from the list once they are completed, this makes the system more robust. If execution is terminated for any reason while a task is incomplete, it will be restarted as soon as the software is reloaded.

On initialisation, the main task handling thread is spawned, and its first responsibility is to prepare the critical section markers which protect the task file. A critical section is a designated segment of code which may only be entered by one thread at a time. By marking several segments of code with the same critical section marker, it is possible to ensure that only one of those segments may be under execution at any one instance. Appropriately applied, this ensures that even with multithreaded execution, there is no way that the task file can be undergoing concurrent reads and writes.

This protection is particularly important as tasks can be added from a number of different modules in the BAKER software, and the information must be read back by the task handler's own main thread. This adds an additional constraint, however—that the functions which access the file must not manipulate additional copies of the task table in memory outside the critical sections, otherwise this may lead to race conditions from the use of stale data. Only inside the critical sections can we be certain that the contents of the file have not changed. Therefore, for destructive manipulation of the task table, the entire table must be read into memory inside a critical section, altered, the file must be cleared, and the data must be rewritten to the file before the critical section is exited. This has implications for performance, since threads wishing to enter one of the critical sections must block, and wait until no other thread is in any of the protected segments. As a result it is important that the code inside the segments is as efficient as possible—and this recommends the use of a simple file format to keep the processing time down.

The module provides a single function for adding new tasks into the task file. This function is called from several places, including the datacore module, where it is used to schedule the compression and movement of completed matrix map files, and most importantly the weblink, from which reconfiguration commands may be transferred, as well as the command to begin uploading all pending data files. The function is also called internally by the task handler's housekeeping function: periodically this module will schedule the compression of any left-over data files which may have been left behind (depending on the timing of upload commands and software restarts), and make them ready for transmission.

Every task in the list has a start time and an expiry time, as well as a local unique ID generated by the 'add task' function. It also contains a designated originator, the ID given to it *by* its originator, and its command code along with up to three parameters. Each time the inner loop of the task handler's main thread cycles, the handler enters a critical section. Next, it loads the contents of the table into memory and deletes the task whose local ID matches that of the task most recently performed. When this is done, it removes any tasks past their expiry time, dumps the data back out, and leaves the critical section.

Once outside the critical section, the list is sorted into priority order to determine which task will be performed next. Tasks are sorted by their start time, although those originating on the local system (such as instructions for moving local files around) are allowed to request a start time of 'immediately', which is of an even higher priority than normal tasks that are ready to execute. In this way, any housekeeping jobs can be performed before remote tasks are allowed to interfere with them. The highest priority task is then executed.

All tasks are scheduled serially, that is, no two are executed at the same time. This is a useful approach since the completion of a task is often required for subsequent commands to execute successfully. In addition, serial execution greatly simplifies the handling system as no additional infrastructure is required to manage problems with concurrency.

Every time a task either expires or completes (whether successfully or unsuccessfully), the handler reports the result code back to the originator of the task, along with the originator's designated ID for the request. The result code may be notification of expiry, it may be a generic success or failure code, or it may be specific to the individual command. The task handler does not re-schedule tasks which expire or fail to complete correctly: this is considered to be the responsibility of the originator, which may elect to re-schedule or discard the task based on the result.

There is a variety of supported commands currently built in to the task handler module. When the module is initialised, it builds a table of all supported command codes, their descriptions, and their corresponding function addresses in the executable, allowing the handler to match code names directly to execution addresses. The commands currently supported include those for moving files (with built-in path protection for added security, so that the function only operates inside the current user's own data files area), calling the datacore's compression routines to pack files, displaying messages to the user, triggering the 'Query Feelings' code (see section 3.2), altering values in the configuration tables (section 3.3.8), exiting and restarting the software and downloading updates.

The module is also responsible for preparing the data used in the task table shown in the status screen (3.3.9) and communicating it to the user interface code. While non-critical, this still requires the use of appropriate thread-safe design, particularly to prevent the software falling over when it is requested to terminate: if either the task handler or the user interface threads perform an ungraceful exit, the other may find itself attempting to pass data to or from a buffer that no longer exists!

3.3.8 Configuration management

The configuration manager module handles all of the configurable parameters controlling the behaviour of the BAKER software. To maximise both the flexibility and speed of the configuration system, the module uses two hash tables—one optimised for numeric parameters and one optimised for string parameters—which are populated from the system registry when the software initialises. The release versions of BAKER also contain a set of defaults that are written to the registry at start-up if the information is missing from the local system. The module also deals with reading those parameters which are set in the registry at a system-wide level (under the `HKEY_LOCAL_MACHINE` root in Windows) rather than just those specific to the current user (under `HKEY_CURRENT_USER`).

The module provides several helper functions for other modules that are useful for high speed configuration reads, as well as enabling two types of configuration writes: the temporary and the permanent. In the case where a parameter is to be altered for the short term (surviving only the current execution of the software), its values are entered or altered only in the hash tables. For permanent changes to the software's execution, the facility is provided for writing these changes through to the system registry.

The performance of the module is important, since the configuration read functions are called numerous times in all but the most speed-critical parts of the software. It is for this reason that the hash table parameter cache was designed, to avoid repeated manipulation of system registry keys. The module also provides access to several 'pseudo-parameters' via the same interface which are non-configurable and not stored in the registry.

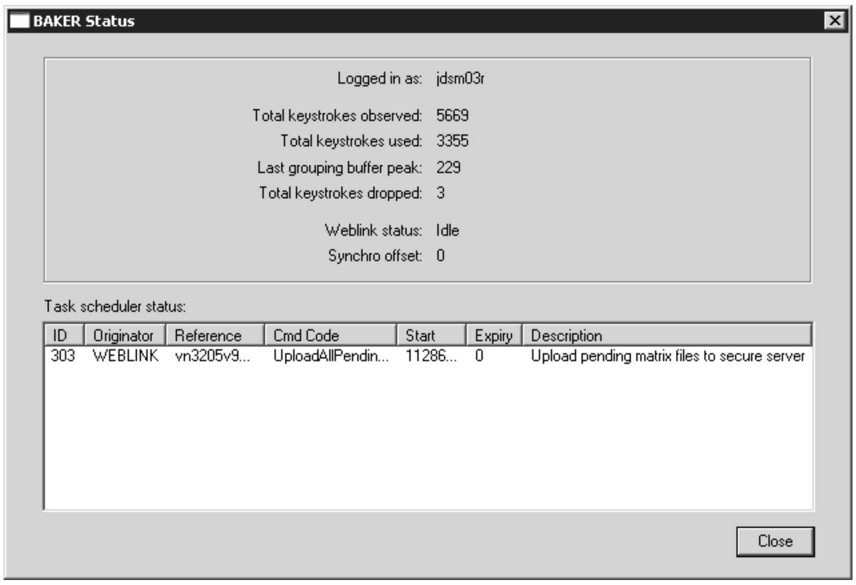
In deployed copies of the BAKER software it is likely that most configuration changes will be initiated by the remote BAKER server, as few of the parameters are designed to be manipulated by the end user. The facility for remote reconfiguration is provided by the combination of the weblink and task handler modules, since one of the tasks recognised by the handler is an instruction to set (either temporarily or permanently) the value of a numeric or string parameter in the configuration tables (see Section 3.3.7).

3.3.9 User interface

The native user interface for the BAKER client is intentionally minimal. The software is designed to be as unintrusive as possible, with only a few controls built into the software. When the program first loads, a small 'splash screen' is displayed for around a second and a half, as a courtesy reminder that keystrokes are being analysed.

Once the software has completed its initialisation, a 16×16 icon of a red capital letter B is added to the Windows system tray. During normal operation, this icon is the only graphical presence of the software on the system. To access further information about

FIGURE 3.3: BAKER loading splash.



Logged in as: jdsm03r

Total keystrokes observed: 5669
Total keystrokes used: 3355
Last grouping buffer peak: 229
Total keystrokes dropped: 3

Weblink status: Idle
Synchro offset: 0

Task scheduler status:

ID	Originator	Reference	Cmd Code	Start	Expiry	Description
303	WEBLINK	vn3205v9...	UploadAllPendin...	11286...	0	Upload pending matrix files to secure server

FIGURE 3.4: Example of the BAKER client status screen.

the status of the software, the user may right-click on the icon—in accordance with the standard interface model of the system tray area—to display a popup menu. From here, the user may elect to exit the software (where this facility has been enabled in the configuration tables), disable the software temporarily without unloading it, show the summary status screen, or launch the default browser and access a web page containing information about the status of the user's account on the BAKER system.

The status screen (Figure 3.4) contains a summary of the performance of the software and the amount of data gathered, as well as a list of the tasks currently pending execution by the task handler module. This information is primarily made available as a means of demystifying the software's behaviours for the end user, and to show a few basic statistics that the user may find interesting.

The web page accessed via the system's default browser also shows statistics relating to the individual account. With the released version of the software, however, this information is somewhat basic. The URI presented to the user is `http://www.typinganalysis.net/controlpanel/` which is ostensibly part of the main `typinganalysis.net` website. However, this page is actually served by one of the ECS school web servers, since it must access the database host whose connectivity is restricted to those machines already inside the ECS network. This is achieved by instructing the web server for `typinganalysis.net` to proxy the requests, but it presents an additional challenge in handling the session and user management.

To achieve automated login for the relevant account on the BAKER system, there must be a session established on the ECS web server where the user has been authenticated by the client software. However, as part of the standard session security model, web browsers will only transmit session tokens (session 'cookies') to the domain from which the tokens originated—and therefore tokens valid for the ECS system will not be valid on the `typinganalysis` domain. Therefore the identifiers used to establish authenticity on the ECS system must be passed via the aliased `typinganalysis` address, and then reflected back to the browser from that address so that it will permit the authentication token to be sent to the new domain. This is achieved with a rapid series of HTTP GET requests and server-side redirections. Fortunately, this slightly convoluted exchange is transparent to the end user, and the result is a browser window which appears to open directly to the above URI with an authenticated session already established. This managed model also prevents bookmarking of session-specific URIs, since the final destination of the browser is the unified address given above.

Chapter 4

Data Collection and Analysis

Once the BAKER platform was successfully deployed for testing, we were able to begin analysis of the incoming data. This chapter discusses characteristics of the data collected and technical challenges faced during experimentation. We also examine the techniques employed in analysing the data.

4.1 Participation and collected data

During the initial beta testing phase, the software had been trialled with three primary participants, all close associates of the investigator, who agreed to allow the software to execute in the background during normal operation of their computers. Later in the study, once the software platform had proven to be stable, a further seven participants were added to broaden the scope of the collected data. These participants were also all acquaintances of the investigator, but with a wide range of ages (teenage to post-retirement) and with various levels of typing experience, from traditionally trained touch-typists to casual hunt-and-peck keyboard users.

The test data from the early beta tests were nevertheless retained; in this section we outline a change to the use of the early data, and also discuss several logistical problems encountered during the deployment of the BAKER client in a genuinely free text environment.

4.1.1 External interest and research focus

The initial period of testing the software platform yielded preliminary results which were very promising. Shortly after obtaining these preliminary results, we were very encouraged by the suggestion that a large external organisation was interested in becoming involved in the deployment of the BAKER software. Initially, this interest was expressed at a biometrics conference, where a representative of the company suggested that it might be possible to collaborate in using the software with some several hundred users. As a result, the planned focus of the research shifted towards producing a short but very broad investigation into the typing habits of a very large user base.

Unfortunately, some weeks and months passed after this initial interest, and despite repeated attempts to follow this up, the trail seemed to have gone cold. It transpires that in the interim period, our contact at the company had left, meaning that a collaborative project was no longer possible. In the absence of such an opportunity, it was decided that the data collected during the beta testing phase (originally not explicitly earmarked for inclusion in the final study) would in fact become part of the research focus, as this meant that we would be able to investigate the stability of the typing metric and how well the results performed over an extended period of some 18 months.

Consequently, the research was extended so that we might also ask: does typing behaviour exhibit sufficient stability over an extended time period that we are able to validate users from typing profiles gathered months or years apart?

4.1.2 Data tainting from shared use

One of the stipulations for participation in the BAKER study was that the software should only be installed on systems where a single user uses a single account on that machine. While the client is active, if another individual were to use the keyboard without switching to a separate user context, this would mean that the data for the contemporaneous time-slice would be tainted with bogus timing information. An option for temporarily disabling the keystroke analysis was provided as part of the client design, but due to the inherently transparent and unobtrusive nature of the software it is likely that the need to disable analysis would be easily overlooked.

It is therefore possible, even likely, that this will have occurred at several times throughout the study, and therefore there are probably several of the BAKER data files which are contaminated with incorrect data. These data may have come from strangers, or even be cross-contamination from other participants in the study, as in several cases there are users who live or work in the same environment as each other. The extent to which this has occurred is unknown, although the positive results in classification from supervised learning (see 5.3) suggest that this cannot have been a *frequent* occurrence.

There is the possibility that the use of unsupervised learning techniques, which would attempt to separate the samples in the matrix files without taking into account to the purported user, might offer some indications of unexpected input (note also the outliers in the self-organising map representation, see section 5.3.1). However, where contamination of a matrix file has been partial within a time-slice, the information from the valid user and the impostor would not be separable, and so using such a technique to explain away anomalous input may be questionable or even dangerous. How do we determine the difference between natural variance and a slightly contaminated time-slice? At any stage where we assume the latter, we risk artificially overstating the effectiveness of the metric. As a result, this technique has not been applied as part of this study.

4.2 Technical challenges during data collection

The BAKER software was initially tested by a group of four beta testers running the software transparently throughout the development process. All of the data collected during the development and testing phase were kept, meaning that in addition to later experiments with a broader user base, it was possible to calculate statistics relating to the stability of the biometric over an extended time period (see section 5.3). During this extended beta-testing period, a variety of additional challenges were faced which affected the progress of the project. These challenges are outlined in the following chapter.

4.2.1 Network outage and software stability

During the extended beta testing phase, the data collection process was unexpectedly halted, due to a major fire in the School of Electronics and Computer Science at Southampton, where the servers were housed. The extensive damage caused by the fire destroyed much of the networking equipment upon which the client-server link was reliant. This meant that the local area network had to be hastily restructured by systems staff, and subsequently it was necessary to reconfigure the flow of the BAKER traffic once connectivity was restored.

Fortunately, the design of the system was sufficiently flexible that it was possible to perform this configuration relatively easily. Since the initial point of contact for the session management and security of the BAKER client weblink was the main school website—restored to functionality as a matter of priority—this meant that any configuration changes could be propagated via this server with relative ease. As a result of the fail-safe design of the client, the outage caused no loss of data, nor did the data collection cease during the downtime. When the network failure occurred, code in the client weblink module successfully detected that the server was either absent or incorrectly configured. Under these conditions, the client was designed to perform in an offline mode, where the keyboard hooking and data collection would continue as normal, and matrix files would be queued on the local machine. In addition, this offline mode invoked the assumption that the local system time would be sufficiently accurate for time-slicing without the need for synchronisation. The protective code designed to detect local clock changes would, under these circumstances, ensure that any apparent time reversal could not result in multiple files with the same time-slice number.

Since the system for uploading files and deleting them from the local machine was designed to be initiated by commands from the server, this meant that no data files were lost in the intervening period. Rather than attempting to push matrices to the server, the client software was designed to wait to be instructed to upload files, and only on receipt of a notification of successful upload would any local files be removed. As a result, when full server connectivity was eventually restored, the server-side code to send upload requests was re-enabled, and all of the data files generated during the outage were successfully transferred to the server. That this worked as intended was a cause of some considerable relief.

4.2.2 Server hard drive failure

Several months after the fire, an additional service outage for BAKER occurred, resulting in a longer period of downtime. The hard drive in the database server (which was also the desktop machine for the investigator) suffered sudden and catastrophic hard drive failure. Initially, this was considered to be a problem with limited impact, as recent data files were kept on the central school filestore, and older files would have been backed up automatically across the network.

Unfortunately, after the (initially) temporary changes to the local area network following the fire, it transpired that the DNS settings had never been updated for the new topology. Since the machine was being backed up by name, this effectively meant that the automated backups were not occurring at all. Copies of older matrix files had been made on removable media, but it turned out that there was a period between the last of these and the first of the files on the central filestore where the only copy of the data was on the now defunct hard drive.

Fortunately, the school generously organised for the hard drive to be sent for specialist recovery. However, a serious of unfortunate administrative delays left the investigator with no desktop machine for a period totalling several months. Furthermore, the hard drive was apparently mis-filed after recovery, which meant that the expected recovery time of two weeks rose to a number of months.

Nevertheless, at the end of this period, the same robustness of the software design that had been so useful during the earlier network outage once again showed its value. Several months' worth of data were successfully collected from the remote clients of the beta testers, and this information eventually joined that from the completed hard drive recovery.

4.2.3 Unexpected web server reconfiguration

More recently, during the main period of data collection, the project was also affected by an unexpected change to the configuration of the school web servers. Originally, the provision of such services had been assumed to be something that would remain invariant throughout the duration of the project. However, an internal memo regarding forthcoming changes to the way in which users' pages would be addressed was received only retrospectively.

Up this point, the BAKER clients had been configured (via a registry setting stored on the local machine) to poll for configuration updates and exchange status information over HTTP to the web site at `http://www.ecs.soton.ac.uk/~jdsm03r/baker/`. The reconfiguration of the services within the school meant that the pages of this site had been moved to `http://users.ecs.soton.ac.uk/~jdsm03r/baker/` instead. With advance warning of this change, it would have been a trivial matter to propagate a configuration change to the clients and set them to poll the new site. Unfortunately, no memo was disseminated until the day of the change-over, and this meant that the BAKER clients were all instantly orphaned, unable to pick up reconfiguration data as their expected site was no longer present.

Again, the software behaved robustly insofar as the problem caused no loss of data. However, to solve the problem it was necessary to deploy a 'hotfix' (in this case, a small utility which would alter the relevant registry settings to contain the new URL and hostname). At this stage, the decision to retain contact e-mail addresses for each of the participants was vindicated, as it was possible to send them each an e-mail requesting that they download and apply the hotfix.

4.3 Analysis methods and metrics

During the development of the BAKER platform, we decided that it would be preferable to collect *too much* information and exclude some from analysis, rather than to reach the end of the testing period and discover that *too little* had been gathered for an adequate demonstration of the utility of keystroke analysis. In practice, however, the size of the set of data assembled was extremely large and therefore processing became somewhat challenging, and certainly time consuming.

To make the analytical process simpler and more reliable, no attempt was made to code anything but the most simple of the testing and learning algorithms. Accurately producing a robust classifier is a non-trivial task, requiring expertise in an area which is not a specialism of this study's author. Furthermore, spending a considerable amount of time on developing an analysis system from scratch carries with it a significant element of risk. It is not inconceivable that a small algorithmic error could render the entire set of results unreliable, and in a way that might be difficult to detect. In contrast, the use of open source statistical libraries proved to be a route which avoided this difficulty efficiently. Not only are such libraries typically subject to a reassuring level of field testing and peer review (and therefore much less likely to contain calculation inaccuracies) but their use reduced the time spent on what might reasonably have been regarded as “re-inventing the wheel”.

As a result of this decision, the development associated with the analytical process was in assembling a tool set which would extract and convert the incoming data into a form which could be readily analysed by off-the-shelf software. This section discusses the existence of these tools and then examines the machine learning toolkit used to perform the analysis of the BAKER data.

4.3.1 Extraction tools

As discussed, rather than begin data analysis by attempting to implement machine learning algorithms from scratch, the decision was taken to apply processing with a variety of existing algorithms from a suitable software toolkit. However, to facilitate using the collected data with generalised machine learning software, it was necessary to develop a set of utilities that would convert from BAKER’s own internal storage formats into plain text representations that would be more portable.

To this end, tools were built for trawling through the thousands of collected files on the database server and selecting them based on filter criteria (such as which member of which matrix family—hold or inter-key—the files contained). This required the use of decompression libraries, as the matrix files are stored in compressed format due to the high number of them gathered; the processing also needed code for reducing the dimensionality of matrices, so that for example 4-dimensional matrices could be processed with algorithms otherwise unable to handle the high memory requirements. The various files could then be converted in this way to work with algorithms expecting plain text (comma separated variable) input.

Initially, the plan was to perform analysis directly on the trigraph and quadgraph data, but the memory requirements for handling this information became prohibitive on the analysis platform. As a result, we proceeded by analysing unigraph and digraph data initially, with the intention of developing methods to increase the dimensionality if the simpler results were unsatisfactory at this level.

4.3.2 The RapidMiner Toolkit

To facilitate processing of the gathered data, the RapidMiner toolkit—previously known as YALE (Yet Another Learning Environment) (Mierswa *et al.*, 2006)—was used. This is a cross-platform environment written in Java for machine learning and data mining experiments, and it allowed testing to be performed on the extracted BAKER data with relative ease.

As discussed, the approach taken with the BAKER project was to carry out supervised learning experiments for *identification* within the sample set initially, as a means to determine the separability of the users' data. This involved the selection of several classifiers which were able to perform supervised learning with a multi-class (nominal) sample set. The algorithms chosen are listed in the next chapter. The use of the RapidMiner toolkit allowed these algorithms to be applied simply and effectively: its ability to manipulate a chain of operating elements (such as file readers, cross-validators and graph plotters) through a graphical interface reduced the time needed to build and test experiments.

Chapter 5

Experimental Results

Once the BAKER software platform had been developed and deployed, a large volume of data began accumulating on the database server and it was possible to start the analysis. This chapter documents the analytical approach and the results gathered from the data.

5.1 Initial analytical approach

Shortly after releasing the beta version of the BAKER software to a limited number of users, some initial testing was carried out to ensure that the system was operating correctly. These incoming data were plotted with respect to the mean hold and inter-key times. The purpose was to indicate whether it would be worth continuing to collect both hold and inter-key data, and whether the two were likely to provide some mutually exclusive information. The results of this are documented in Section 5.2.

Subsequently, once data were being collected from the larger user base after deployment of the release version of BAKER, we were able to begin more formal analysis using the RapidMiner toolkit. At this stage it became apparent that the exceptionally large volume of data was going to become a significant issue for our ability to carry out processing. Attempting to load the exceptionally large matrices into the automated testing software was found to be either slow or entirely unfeasible (causing out-of-memory errors on the testing platform) given the thousands of samples.

To begin testing, therefore, we reduced the dimensionality of the matrices and produced a simple single-dimension mean-values vector for each sample. In the case of hold time analysis, this was simply a vector of mean hold times for each registered key; in the case of inter-key analysis the vector contained mean values for each of the digraph combinations. Furthermore, because the hold time vectors were significantly smaller than the inter-key vectors, and therefore processing tended to be possible within minutes rather than hours, we chose to perform all of our testing in two ways: firstly, during the experimentation, using only the hold time vectors to test analytical techniques. Then secondly we applied the more successful algorithmic approaches to a combined vector, generated by concatenating the hold and inter-key information. For this reason, the majority of the results gathered appear below in two forms: analysis of the hold time data and analysis of the combined metric.

Since it transpired that none of the learning algorithms from the off-the-shelf analysis toolkit would allow us to provide detailed per-attribute-per-sample weights, we did not use the variance data collected by the BAKER software in our analysis. Instead, we simply rejected all attributes (i.e., latencies) of a sample (i.e., time-slice) where there had been fewer than three occurrences of that unigraph or digraph: this was performed by our custom extraction and conversion tools prior to using RapidMiner. Then, to take account of the relative weight of each sample, we marked each vector with the total number of keystrokes used to produce it, and used RapidMiner to perform thresholding on that number to filter our sample set. To put this another way, we actively rejected those time-slices which had fewer than a chosen number of keystrokes.

To determine an appropriate level for this threshold, we ran multiple analyses on our hold time vector with rejection of samples at different minimum-keystroke thresholds and with two different learning algorithms. The results of this are given below. From this, we generated our definition of an *experiment* within the context of BAKER: given that the results of hold-time analysis tend to show an acceptable level of identification after only 300 keystrokes, we refer to an *experiment* as a vector representing a single time-slice generated by a single user where there are at least 300 keystrokes represented in that vector.

5.2 Results from platform beta testing

During beta-testing of the BAKER software, some limited data were gathered to ensure that both the client software and the server-side collection and storage systems were operating correctly. Simple testing was then performed upon these data to demonstrate the nature of the incoming hold and inter-key information. It was possible by examining both the hold and inter-key files (using simple inspection and graph plotting) to determine that the two parts of the overall metric were providing information that was both separable by user using even the most naive approach, and that the hold and inter-key data did not correlate with each other.

Since the matrices generated by the BAKER software can be combined linearly, it was possible to produce one large matrix per user and plot histogram-like graphs representing the (estimated) probability of obtaining multigraphs of a given latency from that user. Since the amount of data for the different users varied so greatly, however, the graphs given here have been normalised so that the results from all three may be shown on the same graph. Although the results are shown as line-graphs for ease of inspection, they have been generated by calculating frequency histograms with a bar width of 5 ms for the hold-time graph (Figure 5.1(a)) and 10 ms for the wider inter-key time graph (Figure 5.1(b)). It is important to note that these results are peak-normalised rather than area-normalised—i.e., the results have been scaled linearly so that the peak value in each case is unity. It is also important to note that the results from User 2 have the lowest resolution (the greatest amount of observed noise) since they have been generated from relatively few multigraphs, and that User 1 shows the smoothest traces due to the large amount of data provided.

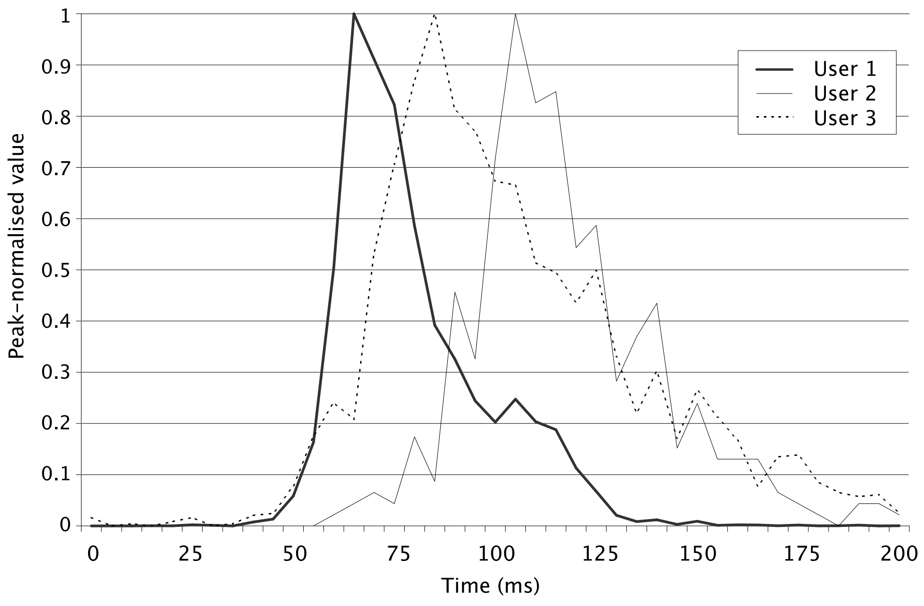
We can see from these simple graphs that even without pattern recognition processing the users are reasonably well separated in their typing styles. It is particularly interesting to note that if we consider the information from the two graphs together, the separation task becomes even easier. While User 2 has the highest hold time latencies, we observe the opposite trend in the inter-key results with the same user's latencies typically below zero, indicating a tendency to overlap keypresses. Additionally, users 1 and 3 retain the same ordering on the graphs.

Since the hold and inter-key times show neither a positive nor a negative relationship, this confirms that the hold and inter-key matrices must contain at least some information which is mutually exclusive. This suggests, perhaps unsurprisingly, that a classifier using both metrics in combination should yield results significantly more effective than one using either metric in isolation.

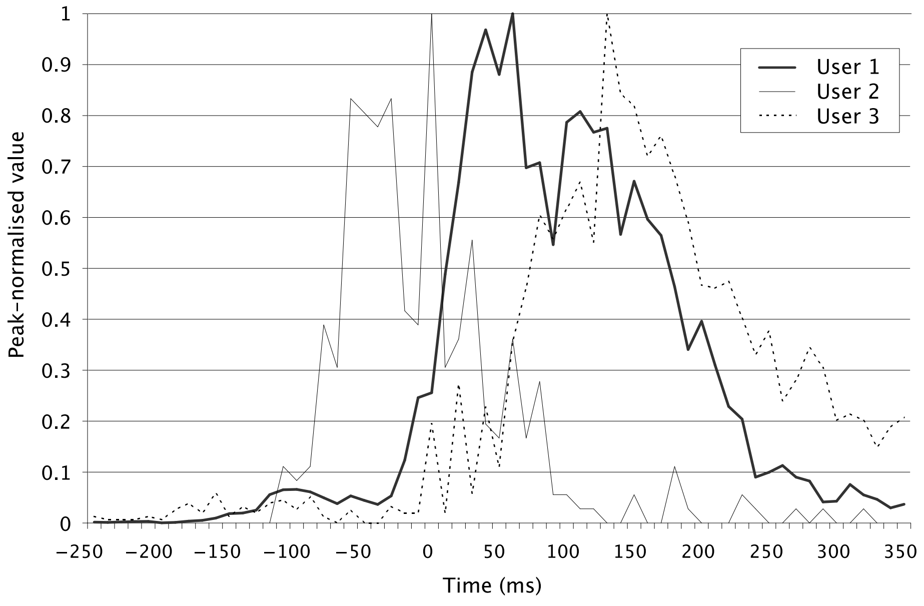
5.3 Extended BAKER results: classification

The main focus of this research has been to investigate the feasibility of long-term GFT analysis. We are interested in determining whether sufficient information (as distinct from random noise) can be found through GFT profiling, such that we may use the technique to augment intrusion detection in information systems. It is also important that we determine the level of usefulness in GFT profiles where the number of keystrokes in a test sample is low (so that future research may pursue practical implementations suitable for wider deployment).

Because of this focus, we are interested in techniques which use provide an indication of the degree of usefulness (i.e., the level of useful information present) in GFT analysis, rather than concentrating on the development of a specific metric for identification or verification in a live system. To this end, we evaluated several different classifiers used for automated supervised learning. These allow us good insight into the feasibility of the GFT analysis paradigm.



(a) Hold times



(b) Inter-key Times

Figure 5.1: Peak-normalised graph showing (a) hold time and (b) inter-key time latencies for three participants

Once the initial beta testing of the platform was complete, copies of the software were rolled out to a total of ten registered users, producing the extended result set. As expected, the system continued to operate effectively. We were able to monitor the flow of incoming data to the server, and this increased in the proportion expected. Monitoring the user experience by contacting the participants showed no difficulties with the user experience, and other than one isolated instance of the software terminating abnormally (which we could not subsequently reproduce, and which may have been unrelated to the BAKER platform itself) we were able to confirm that collection was proceeding as normal.

We also performed some preliminary dimensionality reduction testing prior to starting the more in-depth learning analysis. A sample two-dimensional plot is presented (Figure 5.2) showing that even with only the simplest hold-time information we are able to show reasonable separability of our users. This was encouraging, as it suggested that there was a good level of information present.

In this section we will set out the learning algorithms used for our more comprehensive GFT analysis, with consideration for the effect of missing value replacement. We also evaluate the learning algorithms as they are applied to simple hold time analysis, showing how we selected the most appropriate method for use with the combined metric (hold and inter-key times in combination). Using these same techniques we are then able to show the results of minimum-sample-size thresholding (which leads us to our definition of a valid *experiment* as having a minimum number of keystrokes performed).

These results allow us to select the algorithmic approach best suited to GFT analysis, at least in information-theoretic terms, and we then apply the chosen learning algorithm to our combined metric data, and examine its performance with respect to identification within our sample group. The results are obtained using n-fold cross-validation techniques, allowing us to produce a good estimate of classification performance. When training classifiers, there is always the risk of producing a configuration which fits the training data set but which does not generalise well to other examples. Using n-fold cross-validation involves holding back a small amount of data, which is kept exclusively for testing, training on the remainder, and then examining performance with the test case. By varying the test segment across the entire data set, we can test on all possible examples while never allowing the classifier to see the 'answer' for any test data during training. This is very effective in avoiding the problem of over-fitting, as the end result is based on the ability of the classifier to answer the general case. The results gathered from these tests are tabulated in sections 5.3.5 and 5.3.6, showing the confusion matrices (i.e., how often the classifier was correct) generated by our selected learning algorithm.

Having achieved good results from *identification*, we were then able to extract class-confidence data directly from our chosen classifier's model evaluator. By recording the confidences generated at each iteration of the cross-validation process, we examined each experiment and noted how closely the model matched it to the target GFT profile in the gallery. By setting a threshold for the confidence score associated with every experiment's true class (i.e., the user whose identity we imagine we are verifying), we were able to work back to a set of figures for False Positive and False Negative results within the context of *verification.*

It was then trivial to calculate the FAR (the False Acceptance Rate, where impostors are incorrectly permitted access) and the FRR (the False Rejection Rate, where access is incorrectly denied to legitimate users). By performing this calculation at a variety of confidence thresholds, we are able to plot a graph which is a form of ROC (Receiver Operating Characteristic) curve for the system.

Traditionally, metrics in keystroke research have been presented in terms of FAR vs. FRR. Interestingly, it is possible with a system such as a keystroke analyser to "fix" either the FAR or FRR at zero, by careful choice of threshold. The non-fixed percentage is then stated to give a guide to performance. We might specify a system, for example, where it is important to avoid false rejections, so we relax our notion of acceptance until we are confident that no legitimate user will be denied access. At this point we estimate the probability of failing to detect an impostor and consider whether this is sufficiently low to be useful in augmenting security. Comparably, for a critical system, we might specify that our classifier has to generate an absolute matching confidence (a value of 1.00) to avoid an alarm being raised—here we must evaluate how likely it is that a legitimate user will cause a false alarm. The evaluation of these probabilities is presented at the end of this section of experimental results.

5.3.1 Self-organising map representation

During the exploratory phase of evaluating the incoming data from the BAKER system, the information was processed by the RapidMiner toolkit to provide some preliminary visualisations of the data. The intention was to gain some understanding of whether the incoming samples of keystroke data would be separable by user, and whether useful information could be separated from the inevitable noise. To this end, the toolkit was used to produce a self-organising map (SOM) (Kohonen, 1982; Vesanto and Alhoniemi, 2000) representation of the data.

The self-organising map is a neural-network based processing technique which reduces a data set into a bounded low-dimensional (e.g., 2D) space. This helps to make complex multi-dimensional data more suitable for visualisation. By reducing the data set to two dimensions and plotting it on a 2D grid, it becomes possible to gain an appreciation of how effectively the input data can be separated and how feasible it might be to perform classification on the incoming data.

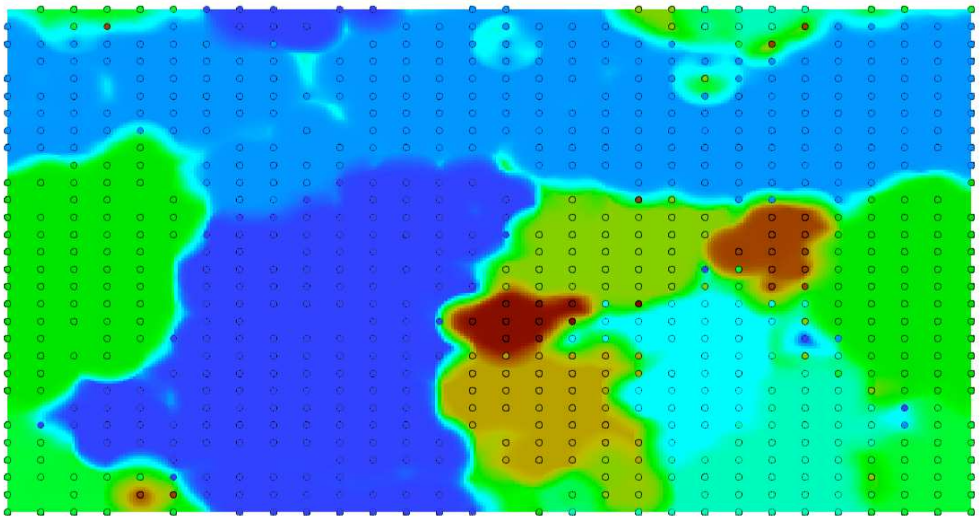


FIGURE 5.2: Two-dimensional topologically ordered self-organising of hold data with threshold set to 400 keystrokes. In the map, each colour represents one of 10 users. Noting that the graph 'wraps around' at the edges, the separation into reasonably large distinct 'islands' of each colour indicates a good level of characteristic information in the input data.

In using this technique, we assign a different arbitrary colour to each experiment according to its true classification (in this case the user responsible for the experimental data). After dimensionality reduction, every experiment will map to a location in the two-dimensional space (determined from the input data by the neural network). We can then traverse this space, colouring it according to the dominant colour in each area. Such a plot will show the extent to which areas of the map (distinguished by colour) are separable. If the source data set is noisy, this will result in a noisy plot with no large uninterrupted areas of colour. In contrast, if the data set contains sufficient distinguishing information for each colour class, the plot will tend to show large distinct islands of colour with clear boundaries.

A 2D SOM based on the hold times data for experiments of greater than 400 keystrokes is given in figure 5.2. We can see clearly from the figure that the areas of differing colours, each representing a distinct user, can be separated. Except for a few outliers (which are perhaps more likely to correspond with the experiments which generate classification errors in our further analysis), it is generally possible to separate the map into areas of a single colour.

5.3.2 Machine learning algorithms

For a learning algorithm to be suitable for our information-theoretic GFT analysis, there were several criteria which had to be met. The algorithm had to be a supervised learner, as we would be presenting it with a known classification—the originating user—for each sample during training. The learner had to be designed to cope with a multi-class learning problem with nominal class attributes (i.e., our multiple users with discrete IDs).

It was naturally important that the algorithm should cope well with the nature of our data set. During preliminary investigations several other classifiers which form part of the RapidMiner toolkit were tested, but they failed to produce results worth documenting (typically because they were unsuited to the data set and produced poor results, or on two occasions through limited documentation and a lack of confidence in their deployment). Of those classifiers which gave adequate results, three were selected for more thorough evaluation:

- Bayesian Belief Network classifier (BayesNet)(Bouckaert, 2005)
- K-Star classifier (Cleary and Trigg, 1995)
- RandomForest (WEKA implementation) (Breiman, 2001)

The basic principle of each of these classifiers is briefly outlined in Appendix C

The results of testing the classifiers against the hold times data set (with a minimum number of keystrokes per experiment of 300) are shown in confusion matrices, in Tables 5.0(a), 5.0(b) and 5.0(c). As will all RapidMiner tests in this thesis, the analysis was performed with 10-fold cross-validation to provide an estimate of performance while avoiding over-fitting. Here, we can see that the K-Star classifier appears to have the weakest classification ability for this data set, with the other two algorithms performing comparably well.

As well as being able to manage the type of data being handed to the classifier, the algorithms needed to show reasonable performance and scalability. This was perhaps one of the most important considerations when it came to moving from hold-time measurements to the combined hold and inter-key metric. Given that the data set with the combined metric gives an input vector over thirty times larger, an algorithm scaling $O(N^2)$ would be slower by three orders of magnitude on switching from the hold times vector to the combined metric!

The algorithm which performed fastest was BayesNet, integrated into RapidMiner but provided by the classifier library from the WEKA project (Garner, 1995). It completed our test data set in 8 seconds rather than the 35 seconds and 30 minutes for the RandomForest and KStar classifiers respectively. For this reason it was the chosen classifier for continued experimentation. Its classification accuracy was not the highest of the three, with an error rate of $2.39\% \pm 0.88\%$. The WEKA implementation of the RandomForest classifier gave the most favourable classification accuracies when tested against input thresholded at a minimum of 300 keystrokes per experiment: its error rate was $2.25\% \pm 0.98\%$. Nevertheless, test durations with RandomForest were over four

(a) Hold times, BayesNet, min. keystrokes 300. Classification error: 2.39% ± 0.88%. Run time approx. 8 sec.

	true.A	true.B	true.C	true.D	true.E	true.F	true.G	true.H	true.I	true.J
pred.A	670	1	0	0	1	1	0	0	2	0
pred.B	0	745	0	0	0	0	0	0	0	0
pred.C	6	0	105	1	0	0	0	1	0	1
pred.D	0	1	0	261	2	1	1	0	0	0
pred.E	1	0	0	6	178	2	1	0	2	0
pred.F	0	0	1	0	0	443	0	0	0	0
pred.G	15	0	0	1	1	0	148	0	0	0
pred.H	0	0	0	0	0	0	3	128	0	4
pred.I	0	1	0	1	1	1	1	0	60	0
pred.J	1	0	0	2	0	0	2	2	0	42

(b) Hold times, W-RandomForest, min. keystrokes 300. Classification error: 2.25% ± 0.98%. Run time approx. 35 sec.

	true.A	true.B	true.C	true.D	true.E	true.F	true.G	true.H	true.I	true.J
pred.A	689	0	1	2	1	1	2	0	3	0
pred.B	0	748	0	0	1	0	0	0	1	0
pred.C	0	0	93	0	0	0	2	1	0	1
pred.D	1	0	4	260	0	0	0	1	8	0
pred.E	1	0	0	2	180	0	2	0	2	0
pred.F	0	0	1	0	0	447	0	0	0	0
pred.G	2	0	2	3	1	0	147	0	0	1
pred.H	0	0	3	3	0	0	3	129	0	4
pred.I	0	0	0	2	0	0	0	0	50	0
pred.J	0	0	2	0	0	0	0	0	0	41

(c) Hold times, KStar, min. keystrokes 300. Classification error: 4.39% ± 1.20%. Run time approx. 30 min.

	true.A	true.B	true.C	true.D	true.E	true.F	true.G	true.H	true.I	true.J
pred.A	688	1	0	7	5	2	6	22	3	0
pred.B	0	746	0	0	0	1	0	0	0	0
pred.C	0	0	64	0	0	0	0	1	0	0
pred.D	3	0	15	258	0	0	2	0	0	0
pred.E	0	1	1	1	167	1	1	0	1	0
pred.F	0	0	1	0	0	443	0	0	0	0
pred.G	1	0	3	1	0	0	146	0	0	0
pred.H	1	0	16	0	0	0	0	105	0	1
pred.I	0	0	1	4	11	1	1	0	60	0
pred.J	0	0	5	1	0	0	0	3	0	46

TABLE 5.1: Confusion matrices showing the performance of the BayesNet, RandomForest and KStar classifiers on the thresholded input data.

times longer than those of BayesNet. Empirically, the chances of a test run failing with an 'Out of Memory' error also appeared higher with the RandomForest tests, making it less suited to the combined metric. In contrast, the rapid operation of BayesNet (processing over 2000 samples in under 10 seconds, of which much was set-up and take-down time for the processing environment) meant that this became the learner of choice.

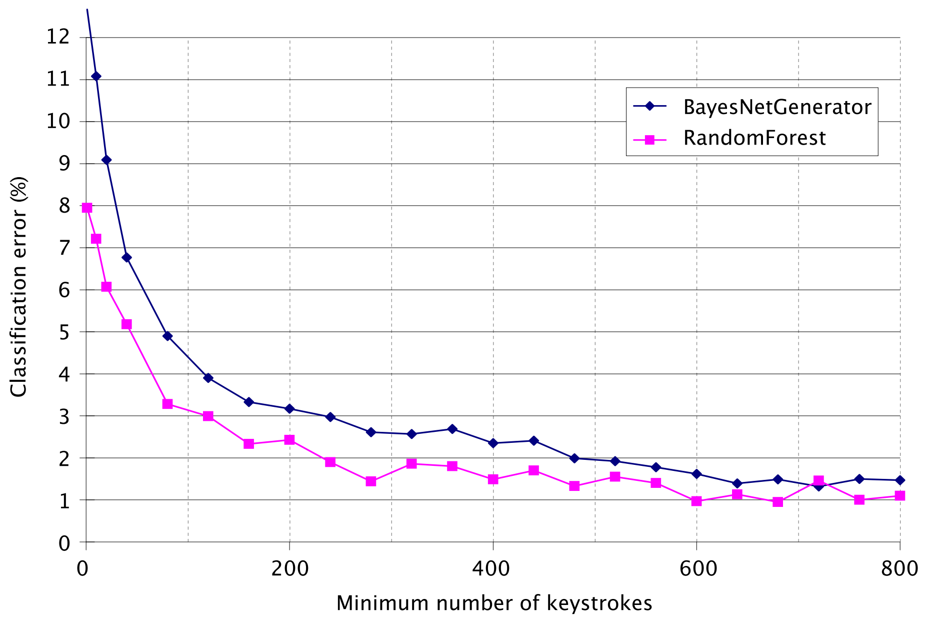


FIGURE 5.3: Hold time classification results for two alternative algorithms

5.3.3 Thresholding by minimum number of keystrokes

Given that there was no facility to apply weights to individual samples during model construction for the classifiers, an alternative approach was necessary to attempt to reduce the influence of the small, less significant samples. In the extreme, a time-slice with only a few keystrokes in it would be much more prone to noise than one containing the means of several thousand: it seems logical that we should seek to reduce the influence of these small, noisy experiment on our overall data set. Similarly it seems reasonable for us to set a lower bound on what we might consider to be a valid 'experiment' within the context of testing. If we were to be given a sample of only a few keystrokes, it is unlikely that attempting to classify it would yield any useful results.

For this reason, preprocessing techniques were used in RapidMiner to reject all samples where the number of keystrokes present was lower than a given threshold. By repeating this over multiple test runs at different threshold levels, it was possible to graph the behaviour across different levels and determine a sensible minimum for analysis. This was attempted with both the BayesNet and RandomForest classifiers to ensure that the behaviour was analysed in as general a fashion as possible (rather than overfitting to the behaviours of a specific algorithm).

Following the generation of this graph (Figure 5.3), the decision was made that for the purposes of all future experiments, we would set a minimum sample size threshold of 300 keystrokes. This value was selected somewhat arbitrarily, although care was taken to choose a threshold which was beyond the 'knee' of the graph. In addition, while the value could have been set higher to obtain more favourable results, consideration was

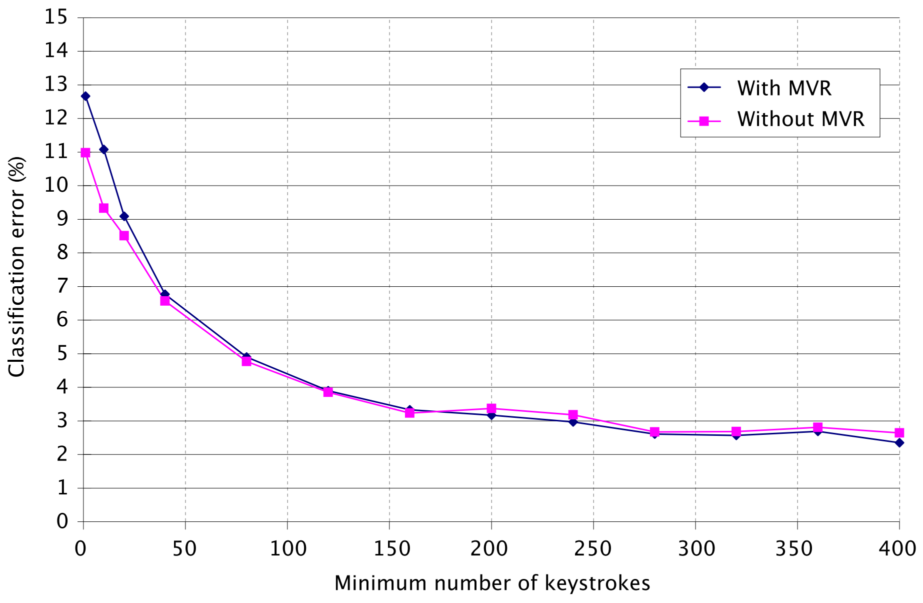


FIGURE 5.4: Effects of missing value replacement on performance

given to the practical value of the assessment. The higher the threshold, the longer the enrolment period for a new user.

While the analysis undertaken was designed to examine the data in a theoretical context, the practical applicability of the metric was also judged to be important. Three hundred keystrokes represents around four sentences' worth of typing. Much of the research carried out in the field so far has relied on much higher sample sizes for analysis: notably, the paper with the closest research focus to this thesis (Downland and Furnell, 2004) discusses how their average time to 'challenge' (upon detection of a suspected impostor) is over 6,000 keystrokes—a figure which they admit is rather high. It is with this in mind that for the purposes of this research, a valid *experiment* is defined as one which contains 300 or more keystrokes.

5.3.4 Effects of missing value replacement

One important consideration when using the learning algorithms from the RapidMiner toolkit is how to deal with missing values. Since it is perfectly possible that many samples will not contain all possible keystrokes, even in hold time analysis (for example, hold times for Q and Z), we must consider how the classifiers will cope with the missing values.

The process of converting from a zero-value entry in the original matrix to the '?'-missing value notation in the RapidMiner input files is taken care of by our bespoke extraction-conversion software. How the learning algorithm treats the missing entry, however, must be selected: it is possible to opt for Missing Value Replacement, where the toolkit will replace any absent data with the some value (usually an average of other such values).

Therefore, a set of tests were carried out to investigate whether MVR would have a beneficial or detrimental effect on performance.

The results are shown in figure 5.4. The graph shows that while MVR has some small benefit for extremely sparse samples, its effect in the main appears to be to 'dilute' the information from our sample sets. Given that the process also has a small negative impact on temporal performance, and furthermore that its effects on the more sparse combined metric would be more unpredictable, the decision was taken to leave MVR disabled.

5.3.5 Hold time performance

The performance of the learning algorithms when presented with the hold-time-only data sets as an identification problem has been shown in Tables 5.0(a), 5.0(b) and 5.0(c). These confusion matrices are the natural output of the classifiers in the RapidMiner environment.

However, if we alter our operator chain to force the learner to output its internal 'confidence' scores at each iteration of our cross-validator, we can extract estimated class confidences for each class of each experiment classified. That is to say, for every sample generated by a user, we can find out how confident our classifier is that the sample matches that user.

By doing this, we can reconstruct our problem as a large set of binomial classifications. Setting a threshold on this confidence figure means we are able to measure both false acceptance and false rejection: for each per-class confidence in each sample, if the figure is below the threshold, this would be a rejection (and we can check whether the rejection was rightful as we know the true originator of the sample). Conversely, any confidence at or above the threshold would count as an acceptance, and so we are able to check if this is a false positive or genuine.

Repeatedly testing our data set at different thresholds allows us to generate a plot of FRR against FAR, which is a form of ROC (Receiver Operating Characteristic) curve—this can be seen in figure 5.5. This allows us to see the trade-off between allowing false acceptances and incorrectly rejecting legitimate users.

5.3.6 Combined Metric performance

As discussed, BayesNet was selected as our classifier of choice, specifically as it had the ability to deal elegantly with the very large data set size for the combined inter-key and hold times metric. By applying the same techniques to the combined metric data set as to the hold times set, we were able to generate the confusion matrix shown in Table 5.2.

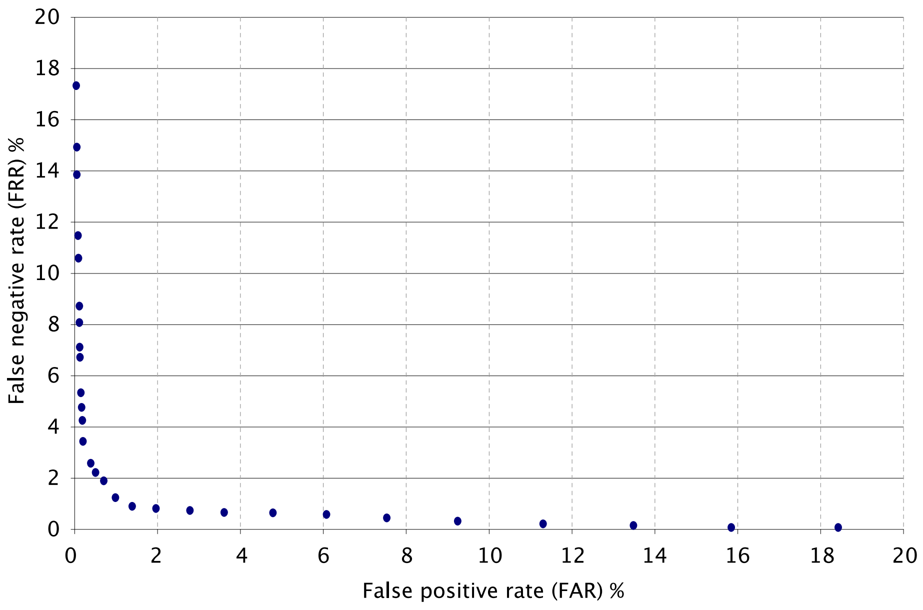


FIGURE 5.5: False Rejection rate vs. False Acceptance rate for hold times, BayesNet, $Threshold = 300$

	true.A	true.B	true.C	true.D	true.E	true.F	true.G	true.H	true.I	true.J
pred.A	659	0	0	0	0	0	0	0	0	0
pred.B	0	731	1	0	0	0	0	0	0	0
pred.C	1	0	98	0	0	0	0	0	0	0
pred.D	1	0	0	136	0	0	0	0	0	0
pred.E	0	0	0	1	54	3	0	0	1	0
pred.F	0	0	0	0	0	151	0	0	0	0
pred.G	0	0	0	1	0	0	82	0	0	0
pred.H	0	0	1	0	0	0	1	53	0	0
pred.I	0	0	0	0	0	0	0	0	50	0
pred.J	0	0	0	0	0	0	0	0	0	36

TABLE 5.2: Combined Metric, BayesNet, min. keystrokes 300. Classification error: 0.53% +/- 0.46% Run time: approx. 2 min

As is clear from this matrix (generated with 10-fold cross-validation) the classification accuracy of the combined metric is very high on this data set. Since we are using cross-validation this reduces the chances of overfitting, and so this suggests that there is a very high level of useful information contained within our matrix format user profiles from BAKER.

Similarly, the method of producing an estimated graph of the FRR against the FRR was applied to the combined metric data set (Fig. 5.6). As expected, the levels of accuracy demonstrated are also very high. The graph also shows the line of the Equal Error Rate (EER), commonly used in assessing the quality of biometric analysers. Where this line intersects with the trend of the plot, this gives as an approximation of the error rate where the FAR and FRR are equal, in this case at around 0.27%. This is a very favourable figure. We can also gain an understanding of the kind of results we should expect to see in a practical situation. With a good quality implementation of a robust

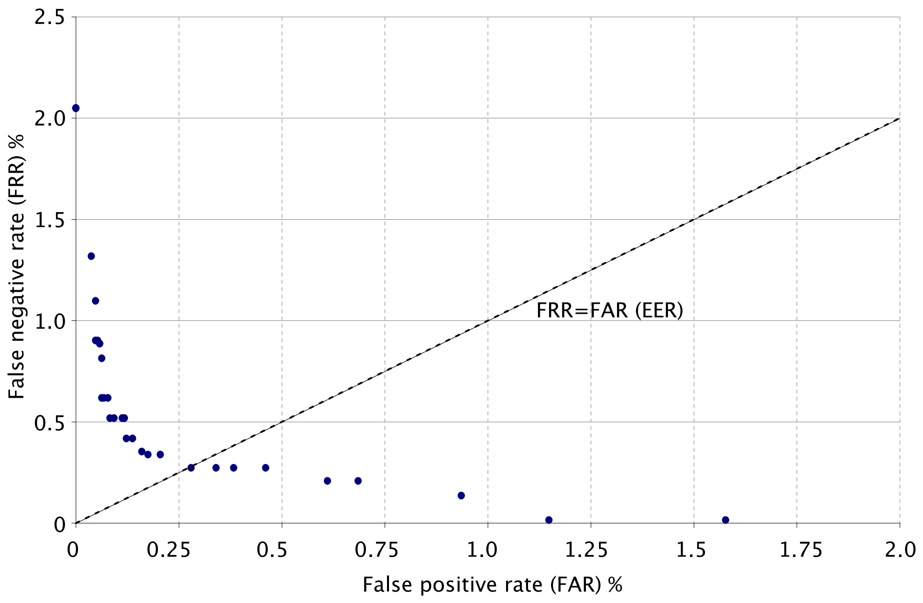


FIGURE 5.6: False Rejection rate vs. False Acceptance rate for Combined Metric, BayesNet, $Threshold = 300$

classifier, using the same type of input data as generated by BAKER, we should expect that—where applied to keystroke streams of 300 keystrokes—tuning for minimal false alarms (i.e., zero or near-zero FRR) should still afford us an impostor detection rate in the region of 98–99%. This figure would certainly be affected by the size of the user base if we were to use the metric for identification rather than verification, since a larger number of users would provide a greater attack surface. Nevertheless, the results from these trials demonstrate clearly that the amount of characteristic information in typing profiles of this size is considerable and that we can achieve very high rates of correct classification.

Chapter 6

Critical Assessment and Conclusions

This chapter presents additional commentary on the results from our previous chapter. This commentary is then followed by a discussion of those aspects of the BAKER project that it has not been possible to complete within the framework of the thesis, but which may prove interesting for the future. Finally, we present the conclusions of this project.

6.1 Comparison to existing literature

The results presented in this thesis from the BAKER study appear to compare very favourably with existing keystroke analysis literature. The research by Downland and Furnell offers a benchmark by which we may examine our results, as theirs is the only other major long-term study of typing in the GFT environment (Downland and Furnell, 2004).

Direct numerical comparisons with their work are difficult as the authors fixed the FRR at zero and estimated the resulting FAR. Nevertheless, their purely statistical approach offered error rates of around 5% and required users to enter some 6000 keystrokes before imposters could be detected with confidence. In contrast, our EER of 0.27% (and an estimated FAR of $< 2\%$ with minimal false alarms), with only a 300 keystroke experiment size, seems very favourable.

Their research went on to suggest that "Further optimisation can be achieved by removing the worst 5 participants from the trial results". Ignoring the obvious criticism—that many results appear to improve when one selectively rejects data that fail to support the hypothesis—we can see that even their "optimised" figures of 1.7%–4.4% are still improved upon by our results, again noting the difference in the number of keystrokes required to constitute an experiment.

6.2 Inferences from experimental results

Our original research question was: 'Is it feasible to develop a software platform which will enable collection and analysis of biometric data based on genuinely free text, and will this provide us with sufficient richness of information for the metric to be useful?'.

We also set ourselves several criteria by which we might measure the level of success of the investigation. To re-iterate, we were looking for:

- totally transparent collection of keystroke data from real users, without compromising privacy and security;
- demonstration of the ability to separate users according to their keystroke data;
- selection of a classifier to demonstrate the practicality of using the biometric live and in real time, in a genuine computing environment.

We can see clearly from the experimental results that we have sufficient evidence to answer positively for the first criterion. The feasibility of developing the software platform was demonstrated beyond doubt, as BAKER successfully operated unobtrusively and gathered a wealth of biometric data while remaining secure and retaining privacy for the end user.

As regards the metric itself, the indication is very much that dynamic keystroke timing patterns offer a rich source of biometric information. Furthermore, it would appear that even in what we would assume to be a noisy environment, we are able to perform effective identification and verification—at least insofar as working with a user base of ten users demonstrates. We were indeed able to separate our users very successfully according to their keystroke data.

Thirdly, we demonstrated the feasibility of using machine learning techniques effectively where the time required for calculation is important. Our ability to perform complete enrolment on the training data and complete 10-fold cross-validation within seconds rather than minutes or hours shows that the techniques are not outside the realm of the live environment, and careful application of the metric could play a useful role in augmenting real time security analysis and intrusion detection systems.

In addition to satisfying these criteria, the inclusion of the long-term trials (over some 18 months) in the testing has gone a long way to answering the question of how stable the typing metric remains, and whether older data can damage accuracy. Since we used stratified cross-validation in assembling the results, we can be sure than a fair mix of data was tested and that there was little or no over-fitting to either recent or older profiles: any of this would have shown up as a significant degradation of the classification accuracy. The high quality of the classification results shows that we are able to use all of these profiles with a good degree of confidence.

When we considered the results of the fixed text experiments, we asked: ‘Do we require additional contextual information, beyond that of unigraphs and digraphs, to assess the identity of a user?’. It would seem that—at least to some degree—the answer is no, we don’t *require* such information to be able to use the metric effectively. The high accuracy of classification achieved with the combined metric (relying on both unigraph and digraph data for hold and inter-key times respectively) seems to provide us with an effective metric without greater context. The differences between this and the fixed-text experiments are perhaps two-fold. Firstly, with the dynamic analysis experiments we are handling samples of 300 or more keystrokes, rather than the 30 or so for the fixed text. It is possible that simply having a larger sampling size reduces the importance of the contextual issue.

Secondly, we are dealing with a different task for the end user, inasmuch as the fixed text experiments require dedicated cognitive effort, whereas with GFT analysis there is no sense in which the user is being forced to follow unnatural typing patterns. It is possible that the effects of contextual difference in the fixed-text experiments were as much cognitive as physiological, and this cognitive effect is likely to be less prominent with the freedom of GFT analysis.

Unfortunately, the high dimensionality and volume of the full-size matrices gathered by the BAKER software meant that it was not possible to perform full supervised learning and classification using more extended multigraph contexts. We are hopeful that in future the full data set will become analysable; this will probably require some combination of more powerful hardware and optimised, customised or simply better suited training algorithms. Nevertheless, we have demonstrated a lower bound to the performance that would be available from extended analysis, and it seems particularly favourable.

6.3 Scalability issues

The work reported in this thesis has, for reasons of practicality, been limited to a small number of *legitimate* users. It may be interesting to apply the same analytical techniques to data gathered from users who are unregistered, and in particular where they may be deliberately attempting to impersonate legitimate users. We have discussed the risk of specific attacks on static analysers; one might question whether the attack of imitating typing rhythm is still applicable to the GFT metric, given that there is no single pattern to learn.

We should also take care not to overstate the conclusions drawn from our modestly sized group of 10 users. While the separability of the typists in our trial is beyond doubt, we should be clear that the issue of *identification* within a large gallery of users is very much more difficult. We can clearly separate our users using their typing profile, and we can gain good numerical confidence as to whether a user matches the typing profile of the user they claim to be. The question of how well the metric scales to a wide user base (of, say, several hundred or several thousand computer users) remains unanswered, and in particular if we attempted to use GFT analysis to *identify* someone without a prior claim on a particular user account, this would almost certainly be a more difficult problem.

6.4 Further work

The intention with the BAKER project was to gather as much information as possible during the experiment and then to become selective about which data were analysed afterwards. Part of the academic contribution of the thesis was always intended to be the production of a large database of typing statistics which could subsequently be subject to additional analysis. For this to be possible, we hope to perform some aggregation of the individual time-slice matrices, and to re-arrange their contents such that we further safeguard the privacy of the participants. It should, however, be possible to complete this without affecting the nature of the pattern recognition problem, and so at this point the data could be analysed by third parties as well as locally.

Two elements of the BAKER study currently remain un-investigated. Firstly, we have a crude chronology of the psychological state of many of the users, and as yet we have not attempted to test for any correlation between their typing performance and their subjective ‘feelings’ during the study. To search for such a correlation could prove very interesting. Secondly, we have the unanswered question regarding the effect of broader context. To investigate whether the information supplied by contextualising unigraphs and digraphs (and therefore treating them as inner measurements of tri- and quadgraphs) would require more powerful hardware or more carefully designed algorithms, so that the high dimensionality of the data could be handled. While it seems unlikely that the results of the combined metric analysis would be improved upon significantly, it may offer us a means of making the classification more robust under unfavourable conditions (such as short test samples).

The ability to examine more closely the nature of the statistical outliers might also be insightful. In the case of identification, and even with generously tuned verification analysis, it seems that a small minority of samples are reluctant to yield to correct classification. Is this because of some statistical aberration in the natural timing of the keystrokes, or is it in some way related to the nature of the input? Isolating the erroneous cases and examining them to determine why they are outliers could be an interesting topic for future analysis.

Our analysis of whether typing profiles are consistent over the medium-to-long term has shown promise. A potential avenue for future exploration would be to use the existing BAKER data set, organised by gross chronology, and to investigate explicitly how well user profiles remain self-correlated over time. The information already generated by the BAKER study should be sufficient for such analysis to be attempted.

A further area of investigation which may prove interesting is in attempting to prevent impersonation where the legitimate user is not only aware of, but perhaps colludes in the identity fraud. Under these circumstances neither password-based nor token-based authentication is likely to be of any use.

For example, computer systems are becoming an increasingly popular means of delivering academic testing, but typically the systems have no means (beyond password authentication, which in this scenario is ineffective) of verifying the identity of those taking the tests. If during an academic course a user is required to undergo some form of continuous assessment, a typing profile may be constructed. Then when examinations are taken at the end of the course, it would be possible to use passive dynamic analysis to obtain a measure of confidence that the individual taking the test is the same person that followed the course. This area of research, as yet unexplored, might benefit greatly from using the BAKER system, making this a possible area for continuing work.

6.5 In conclusion

In the most part, published literature which pertains to our notion of free text has only appeared within the last few years, notably the study by Bergadano *et al.* (2002) which examined forced-free text, and more recently the work by Downland and Furnell (2004) which showed the first attempt to tackle the dynamic analysis of text outside the forced-free paradigm. Alongside these publications, the BAKER work shows not only novelty but also extremely favourable performance.

Compared to the Downland and Furnell paper, which perhaps has the closest research focus to that of this thesis, the BAKER project took on a much more restrictive scope with respect to the privacy of users. Those authors were able to collect a full log of keystroke information, and then apply arbitrary per-multigraph or even per-keyword thresholding (‘Is this word latency more than 0.7 standard deviations from the mean?’). In contrast, our criterion of ensuring any monitoring would be acceptable to end users at large led us to extract our information from a much more restricted data stream. By denying ourselves the opportunity to record broader chronology, we were forced to return to the most basic principles and design the matrix-based approach from scratch.

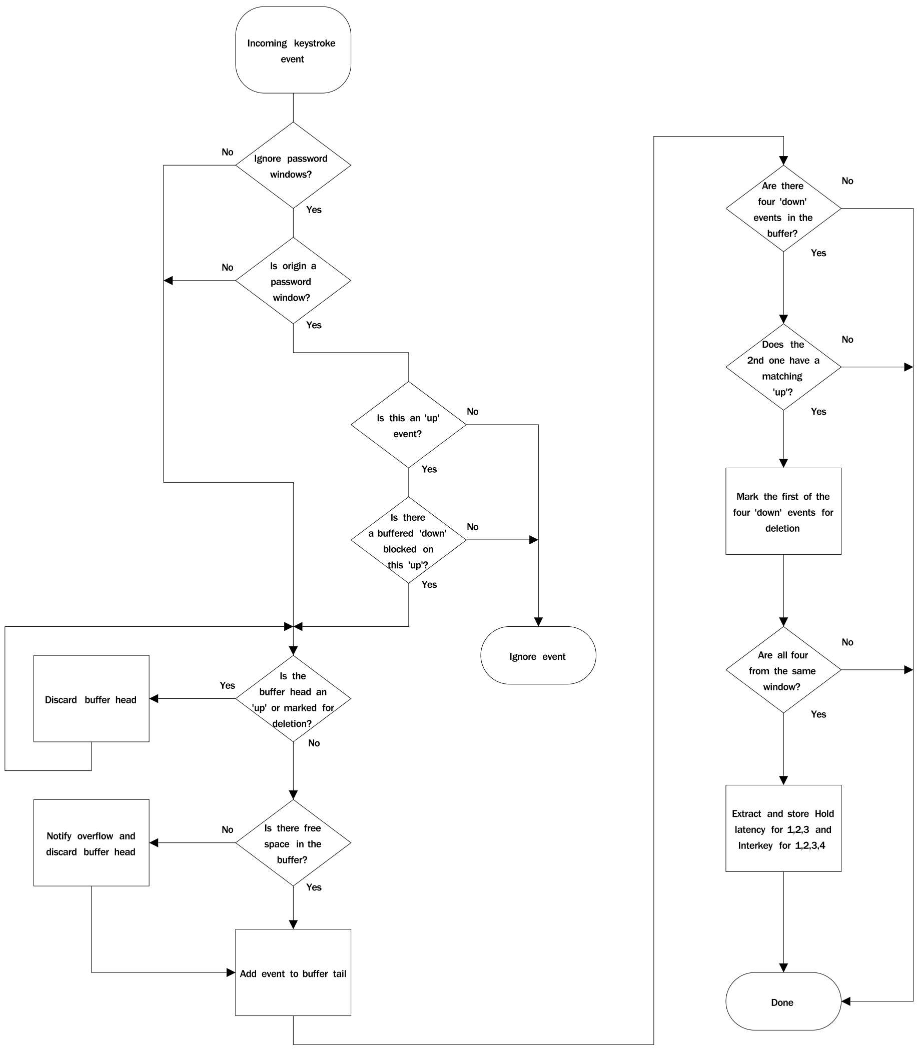
In doing so, we have developed an approach to dynamic keystroke analysis which not only appears to outperform methods in the existing literature with respect to classification, but which seems to do this with a much smaller sample size requirement.

Furthermore, as part of the contribution to research in the field, the work documented in this thesis has produced a database of the most extensive and long-term keystroke analyses to date. There remains much unexamined material generated by the BAKER project, and it is hoped that the analysis of these data will continue in earnest beyond the submission of this thesis.

In undertaking this research, we set out to answer the question, 'Is it feasible to develop a software platform which will enable collection and analysis of biometric data based on genuinely free text?' Without doubt, it is; this is exactly what BAKER does. We also asked 'does this provide us with sufficient richness of information for the metric to be useful?' The experimental results make the answer clear—most definitely it does.

Appendix A

BAKER Grouping Algorithm



Incoming keystroke event
Ignore password windows?
No
Yes
Is origin a password window?
No
Yes
Is this an 'up' event?
No
Yes
Is there a buffered 'down' blocked on this 'up'?
No
Yes
Ignore event
Is the buffer head an 'up' or marked for deletion?
Yes
No
Discard buffer head
Is there free space in the buffer?
No
Yes
Notify overflow and discard buffer head
Add event to buffer tail
Are there four 'down' events in the buffer?
No
Yes
Does the 2nd one have a matching 'up'?
No
Yes
Mark the first of the four 'down' events for deletion
Are all four from the same window?
No
Yes
Extract and store Hold latency for 1,2,3 and Interkey for 1,2,3,4
Done

Figure A.1: Simplified flowchart representing the Grouping algorithm

Appendix B

BAKER Database Tables

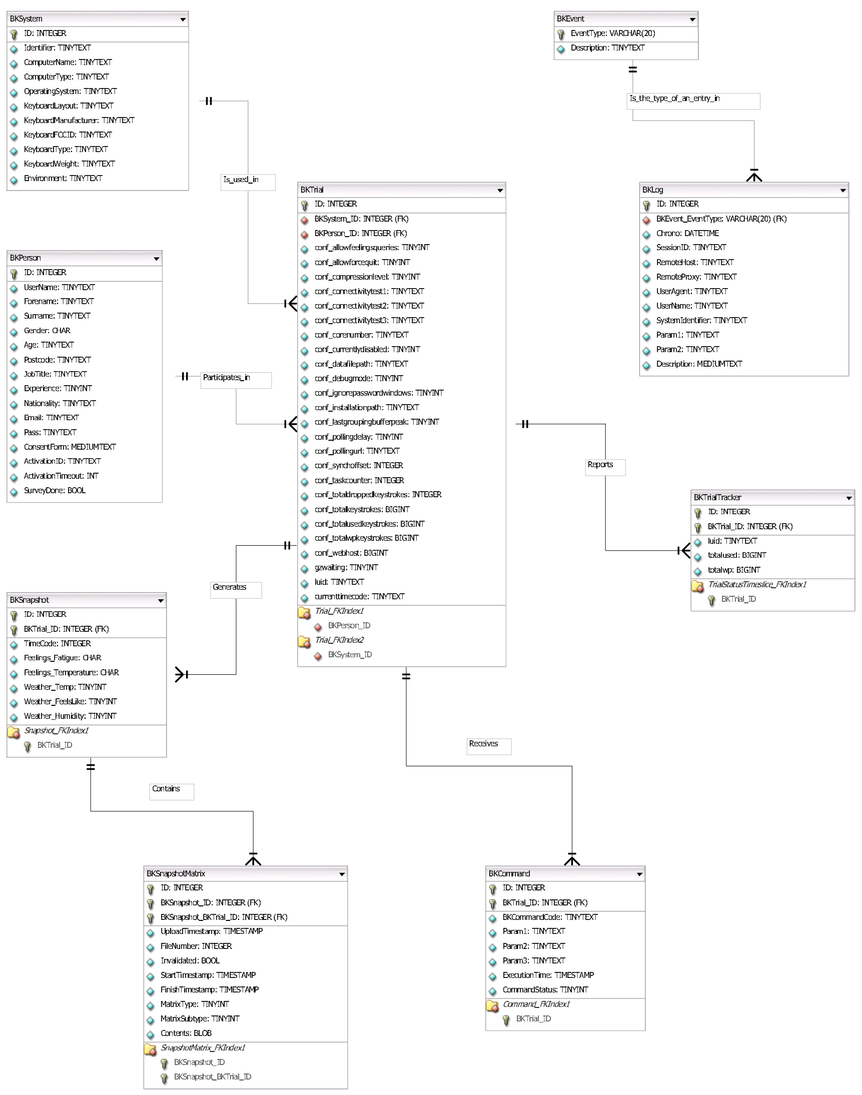


Figure B.1: ER diagram showing the tables of the BAKER database

Appendix C

Summary of Classifier Algorithms

C.1 BayesNet

The BayesNet classifiers offered by the RapidMiner machine learning environment allow the generation and training of Bayesian Belief Networks. A Bayesian Belief Network is a directed acyclic graph, with nodes representing features or classes. Tables of conditional probability determine the relative strength of the links between nodes, with each node having a table which defines the probability distribution amongst its links given the values of its parent nodes (the conditional probability calculations being based on Bayes' Theorem).

Training a simple Bayesian Network is usually performed in two stages, firstly the structure learning, in which the network layout is generated, and secondly the probability distribution stage where the tables of conditional probability are populated. Given the very high complexity of our data sources, the network generation stage would take an exceptionally large amount of time and memory, so we opted to produce random networks of bounded size and proceed immediately to the iterative training phase for populating the conditional probability tables; this gave us very favourable results on our data set as well as a short training time.

C.2 K-Star classifier

The K-Star classifier is an instance-based classifier from the family of k-nearest neighbour algorithms. For each test instance presented to the classifier, it will traverse the set of training data looking for the nearest k data points in the multi-dimensional space, and determine the class of the test case to be the one which is the most common in those k training instances.

The implementation of K-Star allows the operator to choose a guideline, known as the *blend factor*, which determines how close a training instance must be to the test case for the classifier to consider it. In our case, we left this factor (expressed as a percentage) at its default value of 25%.

While the method of this classifier is very simple, it has the disadvantage that for large data sets it tends to be slow, as the entirety of the training data must be searched for neighbours every time a test case is presented. This limitation is clear in our experimental results, where we note that it is the slowest of the three classifiers when used on our large data set.

C.3 RandomForest

RandomForest is an ensemble classifier that works by producing a large number of decision trees (each a directed acyclic graph, much like the Bayesian Network graphs). Each individual tree has nodes representing feature data and links between nodes, with a classification at each leaf node in the graph.

The decisions tree are grown based on a random subset of the training data, and the classification is carried out by combining the output of the various trees. To produce the classification result for a test case, the algorithm queries each tree in turn and then selects the modal class as the result.

The classifier has the advantage that it handles a large number of input variables well, which is useful given the nature of our data set. It also handles well cases where the data are subject to missing values, which is clearly important in the case of our sparsely populated matrices. Nevertheless, the complex nature of our input data meant that this classifier took significantly longer than the BayesNet approach.

Appendix D

BAKER Source

This source is available in a separate volume.

References

Andrews, G. R. and Schneider, F. B. (1983). Concepts and notations for concurrent programming. *ACM Computing Surveys*, **15**(1), 3–43.

Bergadano, F., Gunetti, D., and Picardi, C. (2002). User authentication through keystroke dynamics. *ACM Transactions on Information and System Security*, **5**(4), 367–397.

Bleha, S., Slivinsky, C., and Hussien, B. (1990). Computer-access security systems using keystroke dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(12), 1217–1222.

Bleha, S. A., Knopp, J., and Obaidat, M. S. (1992). Performance of the perceptron algorithm for the classification of computer users. In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pages 863–866.

Bouckaert, R. (2005). Bayesian network classifiers in weka. Technical report, Department of Computer Science, Waikato University, Hamilton, NZ.

Brainard, J., Juels, A., Rivest, R., Szydio, M., and Yung, M. (2006). Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 168–178.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Brown, M. and Rogers, S. J. (1993). User identification via keystroke characteristics of typed names using neural networks. *International Journal of Man-Machine Studies*, **39**(6), 999–1014.

Card, S. K., Moran, T., and Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, **23**(7), 396–410.

Cho, S., Han, C., Han, D., and Kim, H. (2000). Web-based keystroke dynamics identity verification using neural network. *Journal of Organizational Computing and Electronic Commerce*, **10**(4), 295–307.

Cleary, J. G. and Trigg, L. E. (1995). K*: An instance-based learner using an entropic distance measure. In *Proceedings of the 12th International Conference on Machine Learning*, pages 108–114.

Denning, P. J. (1992). Passwords. *American Scientist*, **80**, 117–120.

Dowland, P., Furnell, S., and Papadaki, M. (2002). Keystroke analysis as a method of advanced user authentication and response. In *Proceedings of the IFIP TC11 17th International Conference on Information Security: Visions and Perspectives*, volume 214, pages 215–226.

Downland, P. and Furnell, S. (2004). A long-term trial of keystroke profiling using digraph, trigraph and keyword latencies. In *Proceedings of IFIP/SEC 19th International Conference on Information Security*, pages 275–289.

Furnell, S. and Clarke, N. (2005). Biometrics: no silver bullets. *Computer Fraud and Security*, **8**, 9–14.

Furnell, S., Dowland, P., Illingworth, H., and Reynolds, P. (2000). Authentication and supervision: A survey of user attitudes. *Computers and Security*, **19**(6), 529–539.

Furnell, S. M., Morrissey, J. P., Sanders, P. W., and Stockel, C. T. (1996). Applications of keystroke analysis for improved login security and continuous user authentication. In *Information systems security: facing the information society of the 21st century*, pages 283–294. Chapman & Hall, Ltd., London, UK.

Gaines, R. S., Lisowski, W., Press, S. J., and Shapiro, N. (1980). Authenication by keystroke timing: Some preliminary results. Technical report, RAND Corporation, Santa Monica, CA.

Garcia, J. (1986). Personal identification apparatus. Patent No. 4 621 334, U.S. Patent and Trademark Office.

Garner, S. (1995). WEKA: The Waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64.

Gunetti, D. and Picardi, C. (2005). Keystroke analysis of free text. *ACM Transactions on Information and System Security*, **8**(3), 312–347.

Guven, O., Akyokus, S., Uysal, M., and Guven, A. (2007). Enhanced password authentication through keystroke typing characteristics. In *Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications*, pages 317–322.

Haider, S., Abbas, A., and Zaidi, A. K. (2000). A multi-technique approach for user identification through keystroke dynamics. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 2, pages 1336–1341.

Held, R. and Durlach, N. (1993). Telepresence, time delay and adaptation. In S. R. Ellis, M. K. Kaiser, and A. J. Grunwald, editors, *Pictorial communication in virtual and real environments (2nd ed.)*, pages 232–246. Taylor & Francis, Inc.

Henderson, N. J., White, N. M., and Hartel, P. H. (2001). iButton enrolment and verification requirements for the pressure sequence smartcard biometric. In *E-SMART '01: Proceedings of the International Conference on Research in Smart Cards*, pages 124–134.

Ilonen, J. (2003). Keystroke dynamics. Lappeenranta University of Technology, Available at `http://www.it.lut.fi/kurssit/03-04/010970000/seminars/Ilonen.pdf`, last verified 10/08/2004.

Jones, L., Antón, A., and Earp, J. (2007). Towards understanding user perceptions of authentication technologies. In *WPES 07: Proceedings of the 2007 ACM workshop on Privacy in electronic society*, pages 91–98.

Joyce, R. and Gupta, G. (1990). Identity authentication based on keystroke latencies. *Communications of the ACM*, **33**(2), 168–176.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, **43**(1), 59–69.

Lee, H. and Cho, S. (2007). Retraining a keystroke dynamics-based authenticator with impostor patterns. *Computers and Security*, **26**(4), 300–310.

Leggett, J. and Williams, G. (1988). Verifying identity via keystroke characteristics. *International Journal of Man-Machine Studies*, **28**, 67–76.

Leggett, J., Williams, G., Usnick, M., and Longnecker, M. (1991). Dynamic identity verification via keystroke characteristics. *International Journal of Man-Machine Studies*, **35**(6), 859–870.

Magalhaes, P. S. and Santos, H. D. (2005). An improved statistical keystroke dynamics algorithm. In *Proceedings of the 2005 IADIS Virtual Multi Conference on Computer Science*.

Mahar, D., Napier, R., Wagner, M., Laverty, W., Henderson, R. D., and Hiron, M. (1995). Optimizing digraph-latency based biometric typist verification systems: inter and intra typist differences in digraph latency distributions. *Internatonal Journal of Human-Computer Studies*, **43**(4), 579–592.

Matsumoto, T., Matsumoto, H., Yamada, K., and Hoshino, S. (2002). Impact of artificial gummy fingers on fingerprint systems. In *Proceedings of SPIE: Optical Security and Counterfeit Deterrence Techniques IV, 2002*, volume 4677, pages 275–289, Bellingham, WA, USA. Society of Photo-optical Instrumentation Engineers (SPIE).

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06).*

Miller, B. (1994). Vital signs of identity. *IEEE Spectrum*, **31**(2), 22–30.

Monrose, F. and Rubin, A. (1997). Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 48–56. ACM Press.

Monrose, F., Reiter, M. K., and Wetzel, S. (1999). Password hardening based on keystroke dynamics. In *Proceedings of the 6th ACM conference on Computer and communications security*, pages 73–82. ACM Press.

Monrose, F., Reiter, M. K., and Wetzel, S. (2002). Password hardening based on keystroke dynamics. *International Journal of Information Security*, **1**(2), 69–83.

Moody, J. (2004). Public perceptions of biometric devices: The effect of misinformation on acceptance and use. In *Proceedings of the Informing Science and Information Technology Education Joint Conference*.

Morris, R. and Thompson, K. (1979). Password security: a case history. *Commun. ACM*, **22**(11), 594–597.

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

Obaidat, M. S. and Macchiarolo, D. T. (1993). An on-line neural network system for computer access security. *IEEE Transactions on Industrial Electronics*, **40**(2), 235–242.

Obaidat, M. S. and Macchiarolo, D. T. (1994). A multilayer neural network system for computer access security. *IEEE Transactions on Systems, Man, and Cybernetics*, **24**(5), 806–813.

Obaidat, M. S. and Sadoun, B. (1997b). A simulation evaluation study of neural network techniques to computer user identification. *Information Sciences: an International Journal*, **102**(1–4), 239–258.

Obaidat, M. S. and Sadoun, B. (1999). Keystroke dynamics based authentication. In A. K. Jain, editor, *Biometrics: Personal Identification in Networked Society*, chapter 10, pages 213–230. Kluwer Academic Publishers.

Robinson, J. A., Liang, V. M., Chambers, J. A. M., and MacKenzie, C. L. (1988). Computer user verification using login string keystroke dynamics. *IEEE Transactions on Systems, Man, and Cybernetics*, **28**(2), 236–241.

Rodgers, J. and Nicewander, W. A. (1988). Thirteen ways to look at the correlation coefficient. *The American Statistician*, **42**(1), 59–66.

Schneier, B. (1999). Inside risks: the trojan horse race. *Communications of the ACM*, **42**(9), 128.

Schuckers, S. (2002). Spoofing and anti-spoofing measures. *Information Security Technical Report*, **7**, 56–62.

Siponen, M. T. and Oinas-Kukkonen, H. (2007). A review of information security issues and respective research contributions. *SIGMIS Database*, **38**(1), 60–80.

Tan, B. and Schuckers, S. (2006). Liveness detection for fingerprint scanners based on the statistics of wavelet signal processing. *Computer Vision and Pattern Recognition Workshop*, page 26.

Uludag, U. and Jain, A. (2004). Attacks on biometric systems: A case study in fingerprints. In *Proceedings of SPIE-EI 2004, Security, Steganography and Watermarking of Multimedia Contents VI*, volume 5306, pages 622–633.

Umphress, D. and Williams, G. (1985). Identity verification through keyboard characteristics. *International Journal of Man-Machine Studies*, **23**, 263–273.

Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organising map. *IEEE Transactions on Neural Networks*, **11**(3), 586–600.

Walker, B. (1977). *Computer security and protection structures*. Dowden, Hutchinson and Ross Inc.

Woodward, J. D. (1997). Biometrics: Privacy's foe or privacy's friend? In *Proceedings of the IEEE*, volume 85, page 1487ff.

Yu, E. and Cho, S. (2004). Keystroke dynamics identity verificationits problems and practical solutions. *Computers and Security*, **23**(5), 428–440.