INVESTIGATING & IMPROVING THE RELIABILITY AND REPEATABILITY OF KEYSTROKE DYNAMICS TIMERS

Mr Pavaday Narainsamy [1], Dr Soyjaudah Sunjiv[2] & Mr Nugessur Shrikaant[3]

[1]Department of Computer Science and Engineering, University of Mauritius.
n.pavaday@uom.ac.mu
[2]Professor in Communication and Signal processing, University of Mauritius.
ssoyjaudah@uom.ac.mu
[3]Student, Computer Science Department, University of Mauritius.

shrikaant@gmail.com

ABSTRACT

One of the most challenging tasks facing the security expert remains the correct authentication of human being which has been crucial to the fabric of our society. The emphasis is now on reliable person identification for computerized devices as the latter forms an integral part of our daily activities. Moreover with increasing geographical mobility of individuals, the identification problem has become more acute. One alternative, to curb down the increasing number of computer related crimes, is through the use of keystroke biometric technology which represents an enhancement to password mechanisms by incorporating typing rhythms in it.

Time captured being critical to the performance of the identifier, it is primordial that it satisfies certain requirements at a suitable degree of acceptability This paper presents an evaluation of timing options for keystroke dynamics paying attention to their repeatability and reliability as well as their portability on different systems. In actual passwords schemes users enroll using one computer and access resources using other configurations at different locations without bothering about the different underlying operating systems.

KEYWORDS

Security, Authentication, Passwords, Biometrics, Keystroke, Typing rhythms, Timers

1. INTRODUCTION

The increase reliance on computerized devices for our daily tasks has been correlated with an unprecedented increase in computer crimes. These incidents, making the headlines of the press, have caused serious losses over the last years such that secure systems now constitute a priority to agencies and organizations around the globe [1]. Computer security usually involves a number of components among which successful verification of the identity of a person/entity wishing to use the system stands as the essential front line of defense [2]. Effective system administration, auditing, and efficient resource management all hinge on accurate user identification [3–5]. The knowledge based authentication scheme (password) represents the default mechanism for most systems. It is expected to remain the de facto for many more years due to a number of reasons. First, it is straightforward to implement, easy to use and maintain in that if the typed password does not match the stored one, the user is denied access. Additionally their precision can be adjusted by enforcing password-structure policies or by changing encryption algorithms depending on the security level desired. They represent a cheap and scalable way of validating users, both locally and remotely, to all sorts of services [2, 6]. Unfortunately passwords inherently suffer from a fundamental flaw stemming from human

DOI : 10.5121/ijnsa.2010.2305

psychology. On one hand it should be easy to remember and provide swift authentication. On the other for security purposes it should be difficult to guess, composed of special combination of characters, changed from time to time, and unique to each account [7]. These stringent requirements make people adopt unsafe practices such as record their password close to the authentication device, share with friends and use same passwords on multiple accounts or use familiar names. To reduce the number of security incidents, inclusion of the information contained in the "actions" category has been proposed [8, 9].

Society still relies on the written signature to verify the identity of an individual for a wide array of applications. The complexity of the hand and its environment make written signatures highly characteristics and difficult to copy precisely. The handwritten signature has its parallel on the keyboard in that the same neuro-physiological factors that account for its uniqueness are also presents in a typing pattern as detected in the keystroke latencies between two consecutive keystrokes. The typing patterns of a person using a computer keyboard is a distinctive feature that, even when it is not as precise as others in terms of entropy and classification power [10], has the advantage of not requiring costly equipment and software to be implemented. From the measured features, the dwell time and flight times are extracted to a form a pattern used to represent a computer user. For the system to work, it is necessary to obtain timing information with sufficient resolution [11, 12]. Conversely if the resolution is too finicky the variation for a user will be very wide and may increase computation. Consistency, a major feature of data quality [13], is concerned with how similar a valid user's access typing patterns are to his enroll typing patterns. Without a reliable and consistent timing scheme the uniqueness and discriminate capability of the keystroke dynamics scheme will suffer. Moreover as the same computation is needed each time, before authorizing a user, the repeatability of the processing is also of prime importance.

A survey at the papers published reveals that few authors have paid attention to these algorithm independent factors [14]. The purpose of the forthcoming work is three-fold (1) Demonstrate and assess the different software/hardware base timing alternatives available. (2) Evaluate the response of timers to events (internal/external) which can be consistently repeated a number of times. (3) Propose a scheme with improved timing capabilities that should be used where time measurement is a matter of concern. Inherently we solve the issue of getting sufficiently accurate timing information on time-sharing system [11] where access may be through a variety of networks and hard/firm/software.

This paper is structured as follows. We first present a brief on keystroke dynamics with emphasis on timing of keystroke as reported by other researchers in the field. Section 3 reviews background information on measuring events on the computer. Using performed experiments we demonstrate the limitations of present approach used in designing timers for keystroke dynamics authentication. In the subsequent section we describe our technical solution applied. Section 5 details the experimental results obtained with the prototype developed. Conclusion and references are at the end of the document.

2. RELATED WORK

The idea of using keyboard dynamics for authentication is not new [15] and some products that use such characteristics are already on the market [16, 17] while others have been rumored to be ready for release. Furthermore as keyboard characteristics are rich in cognitive qualities and hold great promise as personal identifiers, they have been the concern of a number of researchers. Delving into the details of each approach is beyond our scope but in general each one measures the similarity between an input keystroke-timing pattern and a reference model of the legitimate user's keystroke dynamics. In light of the reading of previous works, the focus/contributions of these studies can be emphasized along these lines: (1) Target string (2)

Features investigated (3) Sample size (4) Classifier (5) Pre/Post Processing (6) Adaptation mechanism.

A striking commonly is that most have only mentioned the capture of the timing associated with the typed text without any focus on how to do it. Among the very few where some mention has been made, most quoted the accuracy only, varying from 0.1millisecond [18] to 1second [19]. Long et al. [20] reported inter key latencies varying from 55 milliseconds to hundreds of milliseconds corresponding to efficient typists and "hunt and peck' users respectively. Brown and Rogers [21], Araújo [22] and Ostry [23] collected data at millisecond accuracy while in both studies of Obaidat and Sadoun [24, 25] samples were collected with an accuracy of tenths of a millisecond (0.0001 second). In some previous work, statistical techniques were relatively successful with only hundredth of second resolution [26]. Shepherd [27] pointed out that, with a resolution of 0.0001 of a second, events are recorded with an accuracy of approximately one percent.

The four different timers, available for use on the windows platform, were mentioned in [12], and details were rounded to 10ms as it is mentioned that higher resolution was not possible contrary to what others have said. QueryPerformanceCounter being updated between each successive API may achieve a resolution in the order of a microsecond [11, 12]. In a recent work, Cho and Hwang [13] introduced the concept of data quality and used artificial clues and rhythms to improved typing consistency of users.

3. BACKGROUND THEORY

This section focuses on the underlying hardware which makes it feasible to use the Personal Computer (PC) for timing purposes. An oscillator supplies cycles of fixed input frequency to the timer device. At each pulse of the oscillator, the counter connected to its output is decremented and when it reaches zero, it sends an interrupt signal to the processor. At that instant, in the aperiodic mode, counting stops while in the periodic mode it is reset and counting continues. An input register stores value to be loaded in the counter register after a reset [28]. The counter can be accessed by both hardware and software. With evolution of computer hardware changes in timing modules have been added as enhancements to the existing architecture.

The 8253/8254 Programmable Interval Timer (PIT) chip basically consists of a 1.193182 MHz crystal oscillator which is used mostly in the periodic mode [29-31]. In 1984, the first Real-Time Clock (RTC), in addition to the 8254 PIT, was included by IBM to keep track of the local time, which is read during boot time and maintained by the OS [32]. The presence of a small battery as an alternate source of power maintains the correct time while the computer is off. The Advanced Programmable Interrupt Controller (APIC) timer, also known as the CPU Local timer, has many of the capabilities of the above mentioned timers and was designed to be used to synchronize multiple processors [31]. In a multiprocessor system, each processor has a local APIC, integrated onto the processor chip with the input frequency being the processor's base front-side memory bus (FSB) frequency [30]. The timer being of 32 bits long is considered to be finer that the PIT's timer of 16 bits [33]. It is therefore programmed to issue interrupts at very low frequencies. The Power Management (PM) timer is an additional system timer that is required as part of the Advanced Configuration and Power Interface specification (ACPI). The PM timer continues to run even in some power saving modes where other timers have been stopped or slowed [28]. Additionally it incorporates a 24/32 bit counter that increments at 3.579545MHz and therefore be programmed to generate an interrupt when its high-order bit changes value [34]. Among the latest addition is the Timestamp Counter (TSC) which started with the fifth-generation CPUs (Pentiums). It is a 64-bit counter of elapsed CPU cycles since power-on [35] and can be captured through the RDTSC (read TSC) assembly instruction. The timestamp counter runs off the CPU oscillator, and rolls back to zero each time the processor is

reset. Nowadays with the increasing use of multi-core/hyper-threaded CPUs, systems with multiple CPUs and 'hibernating' operating systems, concerns have been raised on its accuracy [36] and portability of applications as all processors may not have a similar features [37]. The High Precision Event Timer (HPET) is a new high-resolution timer chip, developed jointly by Intel and Microsoft to meet the timing requirements of multimedia and other time-sensitive applications [31]. 8 HPET blocks may exist in a computer with each block consisting of fixed-rate counters and 3 to 32 independent timers [38]. It has been design such that in the future, the OS may be able to assign specific timers directly to specific applications. With a minimum of 10 MHz frequency and 32-64 bit counter, it can be used to capture timing for up to once in 100 nanoseconds making it more efficient and faster than legacy timers [35]. It supports both periodic and aperiodic modes but it is not supported by operating systems designed before HPET existed therefore exist only on Windows XP, Windows Server 2003, Linux kernel 2.6 and above [39].

Interactions to the underlying hardware are made, on windows platform, through the Now, TimeGetTime, QueryPerformanceCounter, and GetTickCount while on the Linux environment the GetTimeofDay and ClockGetTime are available [35].

The Now property returns an object that is set to the current date and time on the computer. The values are measured in 100-nanosecond units called ticks using the reference 12:00 midnight, January 1, 0001 A.D [40]. The precision of this property depends on the system timer which in turn has a precision of about 10ms in Windows NT 3.5 and later version. It was previously at 55ms in Windows 98 [41]. Even though current date and time on the computer is updated at regular interrupts [42], it is still subject to adjustments by the Windows Time Service [43]. GetTickCount expresses the number of milliseconds that have elapsed since the system was started as a 32-bit unsigned integer value. The time will wrap to zero if the system is run continuously for 49.7 days [44]; hence the need to check for overflows while comparing times. Another variant is the GetTickCount64 which takes much longer to wrap [45] and as the previous one it is based on the system timer interrupt [35]. The resolution is therefore limited to that of the system timer (usually the PIT) [44]. TimeGetTime forms part of the Windows Multimedia Timer Functions [46], and is particularly used in applications that require high-resolution timing [47]. It returns the number of milliseconds elapsed since Windows was started and therefore similar to GetTickCount overflows after 49.7 days [45]. The precision is dependent on the underlying operating system varying from 1ms to 5ms or even more. To improve its precision the TimeBeginPeriod and TimeEndPeriod functions can be used to modify the update interval of the counter. However this in turn affects the global setting as Windows uses the lowest value (that is, highest resolution) requested by any process [48]. Moreover this causes the thread scheduler to switch tasks more often and can reduce overall system performance and additionally prevent the CPU power management system from entering power-saving modes. QueryPerformanceCounter (QPC) and QueryPerformanceFrequency (QPF) are API calls that retrieve the counter value and frequency of the high-resolution performance counter respectively. Using the frequency and the number of counter ticks elapsed, the lapse of time can easily be calculated. However if the return value of QPC or QPF is zero then high performance timers are not supported by the hardware [49]. Therefore the accuracy of the high-performance counter depends on the underlying hardware as well as on the version of windows being used. The PIT is used in Windows 2000, PMT in Windows XP and HPET on Windows Vista [4] respectively. QPC is recommended for high resolution timing [50], even though the former may use the TSC in some systems.

Under the Linux environment the GetTimeOfDay function provide the current time, expressed as seconds and microseconds since the Epoch (00:00:00 UTC on January 1, 1970) [51]. On the other hand the function Clock_GetTime takes a clock identifier as parameter and retrieves the time from the specified clock [52]. The clock identifier can be the system real time clock, the per process/ thread timer from the CPU or the clock monotonic. However the timing process can get pre-empted during timing, producing unwanted delays in the measurements.

The functions SetPriorityClass and SetThreadPriority can be used to change the priority of a process to Low, Below Normal, Normal, Above Normal, High and Realtime. The use of Realtime priority can make the system unstable as it interrupts system threads that manage mouse input, keyboard input, and background disk flushing [53]. In Linux, the process priorities are known as nice values with a range decreasing from the highest (-20) to lowest (19) with the default value being the median value 0. They can be modified by doing a priority call.

4. SET UP

In keystroke authentication, capturing of keystroke timings is vital to the successful operation of the verifier. Usually a toolkit is constructed with an appropriate GUI for users to type their text while in the background their rhythms are stored. Two methods were used to assess the capabilities of the timers. The tight-loop test provided an easy way to compare the repeatability and constancy of the timers within a system. Testing the quality of the computer's time can only be achieved by comparing it with an external reliable source. The use of a robotic arm was not successful as most robots are computer controlled and this would come down to using a computer to assess a computer. Additionally making a robot pressing a key was almost impossible except if additional variation of the sticky key and repeat rate features were considered. Therefore the time elapsed between two successive edges of an electrical square-wave signal of known frequency was used. A function generator (Cathode Ray Oscilloscope) was calibrated to produce the electrical signal fed to the computer via the parallel port.

The tight loop implemented had adjustable loop-count so that the execution time could be varied to investigate its linearity. The test though very simple and easily replicated on various computers without any special equipment or arrangement, required special precautions to avoid erroneous results. The optimizations applied by the system can leave the tight loop unexecuted if the latter does not do useful work. As an example, an empty loop is skipped by C++ unless the result of that calculation is used outside the loop. Therefore a simple computation is done inside the loop, and the result of this calculation is used outside the loop to emphasize the 'usefulness' of the code.

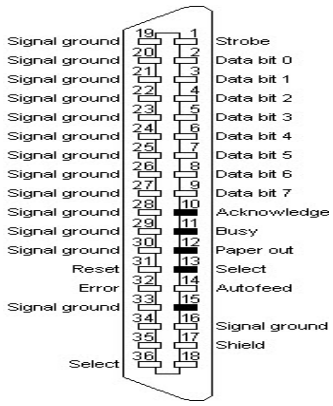


Figure 1. Parallel-Port Cable Pins Specification

To maintain constant CPU speed throughout the timing process, technologies that dynamically adjust CPU frequency (Intel SpeedStep and AMD PowerNow) are disabled. This achieved by choosing the "Always On" power scheme from Power Options in the Control Panel or by turning it off in the BIOS itself [54]. For the external source test, the electrical leads from the function generator are connected to an input line (pin 10,11,12,13 or 14) and any ground pin. An input pulse was applied to change the voltage on the pins at regular intervals and monitored by continuously polling the parallel input port at address *0x379*. Changing the frequency of the

pulse changes the length of time the electrical signal stays high or low in a square wave i.e. the time elapsed between two edges of the wave. To avoid damage to the parallel port and the motherboard the input signal was then set to be of the same amplitude as that supported by other pins of the parallel port.

Performing the experiment only under normal condition is a naïve move, so exploration under stress conditions was also performed to get a picture of worst case scenarios to which keystroke dynamics may be subject to.

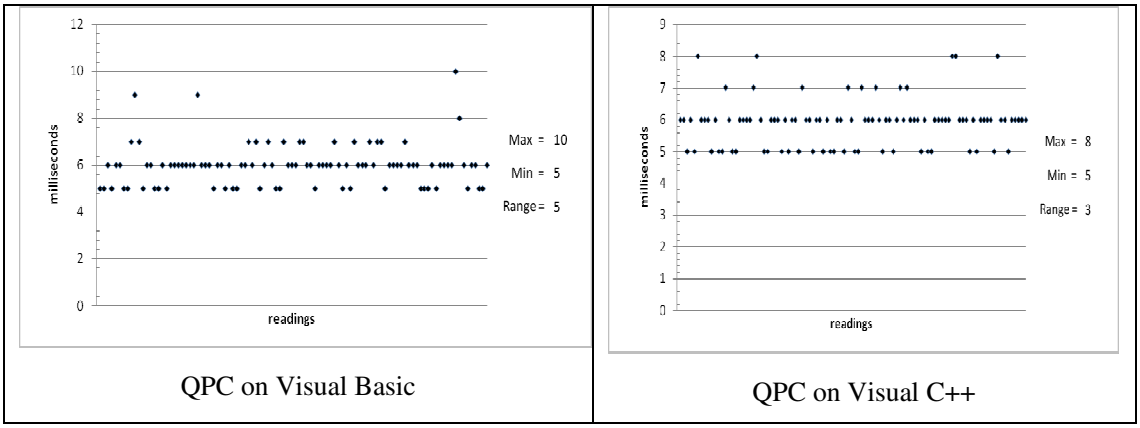
The *Stress* program (developed by Amos Waterland) was used for Linux while for windows it was *SiSoftware Sandra Lite.* They allow straining components such as disk, memory and CPU. Similarly the codes for the experiments were implemented on two programming languages (C++ and visual basic) and different operating systems (Vista Ultimate SP 1, XP SP 2 and Linux 8.04, kernel 2.6.24). The different configurations for the computers used are as follows. Most of the computers are from the Computer Science Laboratory.

- HP Laptop XP: HP Compaq model nc6000 Intel Pentium M processor 1600MHz with 1024 MB RAM
- Lab PC : Intel Pentium 4 CPU at 1.60GHz with 640 MB RAM
- Home PC : Intel Pentium Dual CPU E2180 at 2.00GHz with 2GB RAM

5. RESULTS

The Linux timers were implemented as a simple console program that accepts command-line arguments. Communicating with the parallel port was simple with older versions of Windows (DOS, Win95/98) but with Windows NT and later, the Inpout32.dll was used [56]. This DLL contains the necessary methods to perform port I/O and can be easily linked and used in Microsoft Visual Studio's projects. In Linux, the inb function was to read the input.

For the tight loop test, the loop count value was adjusted so that the execution time is less than 10ms. 100 readings were taken in all cases and the *scatter-graphs* below demonstrate the spread obtained.



QPC on Visual Basic

QPC on Visual C++

Figure 2. Spread in time taken for the tight loop test.

No considerable differences were observed in the variations of the timers in the different languages used, which may be because of the underlying .Net Framework that they share. In terms of performance however, a tight-loop of a given length (loop count) takes less time to

execute in C++ than in VB. One striking feature was the random spread in time taken for the same task when repeated a number of times.

With the Now and GetTickCount timers, the values were found to be multiples of the system clock interrupt interval (obtained by calling the GetSystemTimeAdjustment function). It was either 10 or 0 with values changing from XP to vista. When tested on PCs of different configurations the variations were wide enough to question their use. For GetTickCount when the system clock's update-interval is not a whole number e.g. 15.625ms, then the return values are rounded to 0, 15 and 16. TimeGetTime behaves like getTickCount if timeBeginPeriod is not enabled. Otherwise, it is able to measure time values that are smaller than the system clock interval.

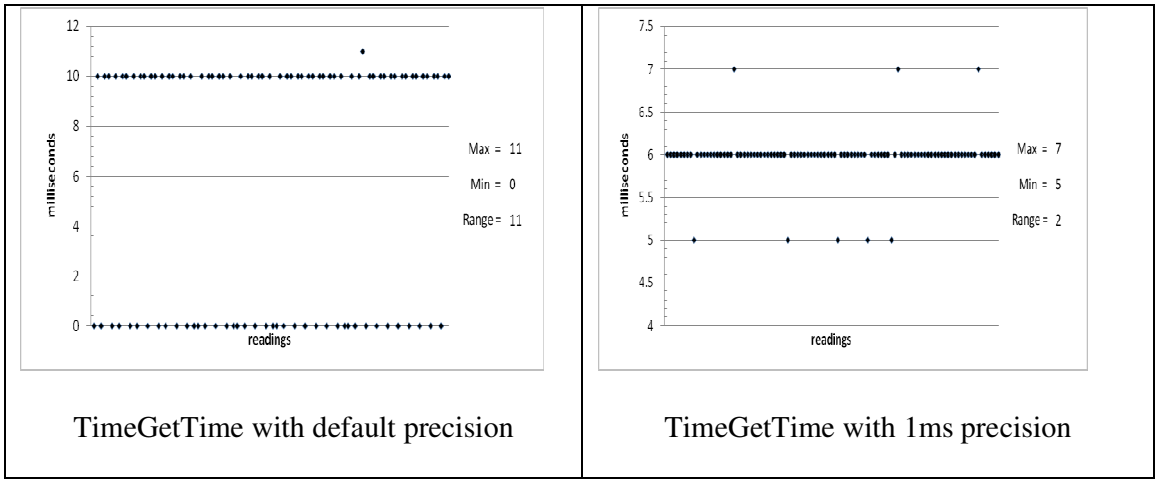


Figure 3. Effect of adjusting precision on performance.

The second graph demonstrates a considerable improvement over the left one with the timeBeginPeriod function enabled. This allows the system to update its time every millisecond at the expense of system performance as mentioned above. The graph below taken from The Performance Monitor program under Windows Administrative Tools clearly indicates these changes.

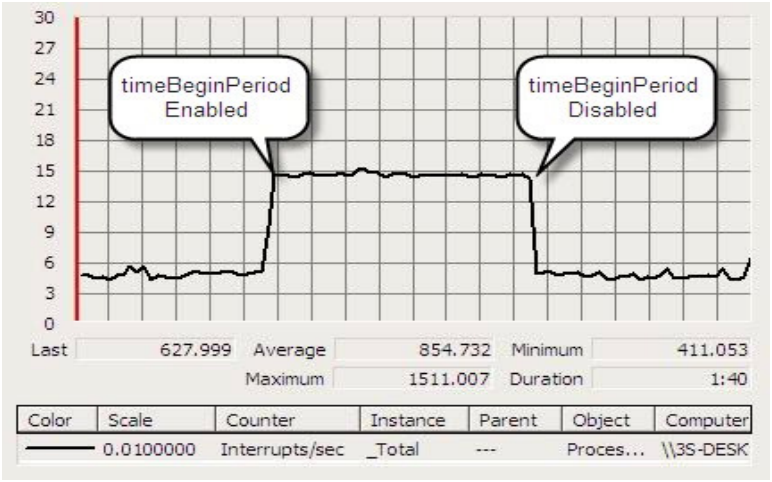
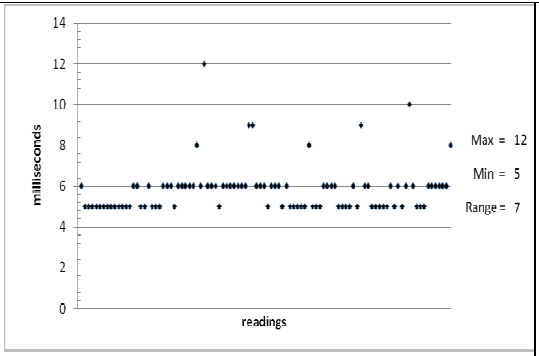


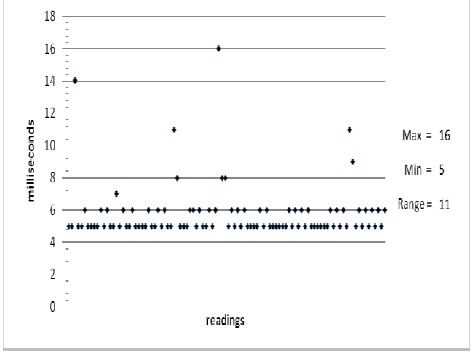
Figure 4. Effect of enabling and disabling timeBeginPeriod on performance.

Figure 4 shows a considerable rise in the number of interrupts per second, from a low value of 411 to a maximum 1511, pointing towards millisecond-update of the TimeGetTime counter when the precision is set to ms.

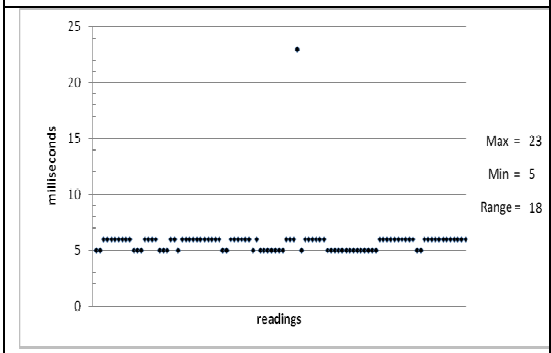
The frequency of the performance counter provides an indication of the underlying hardware clock as shown below with the experiment repeated on computers with different configurations.



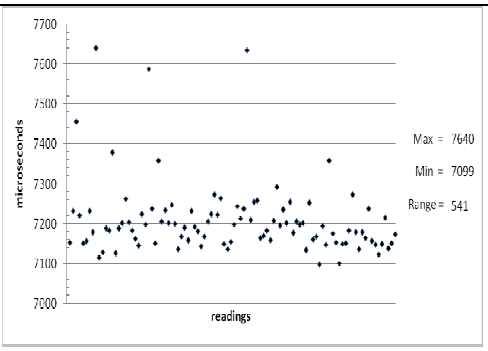
QPC based on 2.0 GHz Time-Stamp Counter [Home PC XP]



QPC based on 14.3 MHz HPET [Home PC Vista]



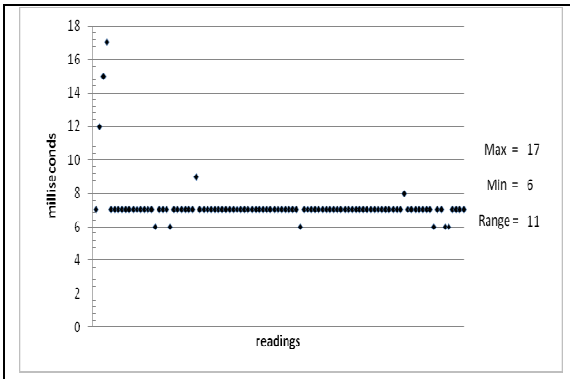
QPC based on 3.57 MHz PM Timer [HP Laptop XP]



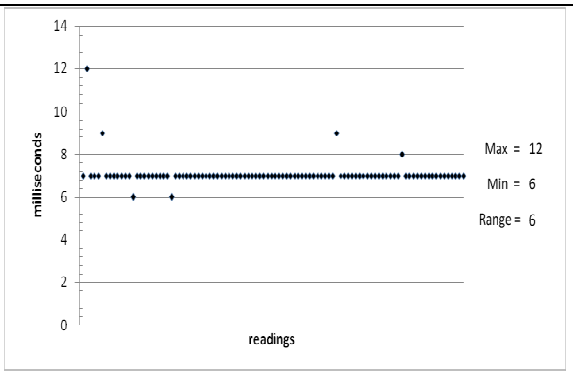
Low variations in values using QPC [Lab PC XP]

Figure 5. Performance obtained using the QPC on computers using different timers.

The graphs below shows the results obtained under the Linux environment. Similar to the windows plateform the results were not free from jitter as peaks much above the average value were obtained.



Gettimeofday [HP Laptop Linux]



CLOCK_MONOTONIC [HP Laptop Linux]

Figure 6. Performance of timers on Linux using the same PC.

The effect of varying priority on results captured is shown in the table below for two different windows configuration configurations. To facilitate understanding only the range variation is shown.

Table 1. Range of Windows timers (ms) with Increasing Priority

	{ Lab PC XP/ Home PC XP}			
Timers	Normal	Above Normal	High	Realtime
DateTime.Now	16/16	16/16	16/16	16/16
GetTickCount	16/16	16/16	16/16	16/16
TimeGetTime	2/6	2/5	2/7	2/6
QPC	0.541/5	0.637/4	0.585/5	0.349/5

Similarly the same investigation was carried out under the Linux operating system. The Nice priorities didn't affect the results (table 2) much whereas the results from FIFO scheduling policy were very impressive as can be seen in table 3. Under FIFO, the latencies were much less than 1ms. This clearly speaks for the need to have control on the FIFO priorities when capturing time.

Table 2. Range of Linux timers with Increasing Priority (Nice Values)

	Linux Nice Values ⟶ { Lab PC /HP Laptop}				
Timers	0 (normal)	-5	-10	-15	-20 (highest)
gettimeofday	7/11	7/21	6/22	6/27	6/21
Clock_gettime	16/6	23/7	24/8	23/6	23/22

Table 3. Range of Linux timers with Increasing FIFO Scheduling Priorities

	Linux FIFO Scheduling Priorities ⟶ { Lab PC /HP Laptop}				
Timers	1	30	50	70	99
gettimeofday	0.222/0.540	0.238/0.530	0.218/0.509	0.220/0.610	0.222/0.524
Clock_gettime	0.220/0.520	0.191/0.541	0.186/0.603	0.258/0.461	0.187/0.622

Users usually enter their passwords while there are other background processes running and this has been boosted with multitasking capabilities of the operating system. This test aims determine the effects on timing when background processes access various components of the system. When the system is under stress, the timing is highly affected. In Windows, increasing the priority of the timing process negated the bad effects of the stress program while for Linux, the timer process under FIFO policy was nearly unaffected.

Table 4. Range of variations of different timers under different stress conditions

Timers	Lab PC Windows Priority Levels					
	Normal			Above Normal		
	CPU	MEM	I/O	CPU	MEM	I/O
DateTimeNow	63	78	16	16	16	16
GetTickCount	47	62	16	16	16	16
TimeGetTime	64	41	10	2	2	2
QPC	34	63	8	0.629	0.887	0.316

Timers	Lab PC with Linux Priorities								
	Nice = 0			Nice = -20			FIFO Priority 1		
	CPU	MEM	I/O	CPU	MEM	I/O	CPU	MEM	I/O
GetTimeOfDay	12	22	24	9	12	24	0.223	0.256	0.235
Clock_GetTime	32	17	21	14	32	26	0.292	0.112	0.235

From table 4 above it is clear that to minimize variations in the data captured and achieve repeatability priorities have to be maximized.

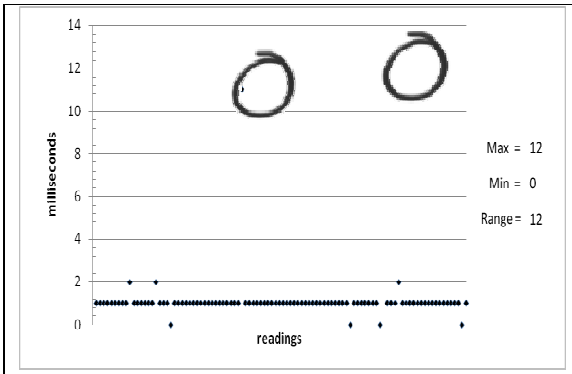
For the electrical signal test the frequency generator was set to output a square wave of frequency 500Hz. With this frequency, the time captured by the computer must ideally be equal to 1ms.

$$500\ Hz \equiv 500\ cycles\ per\ second$$

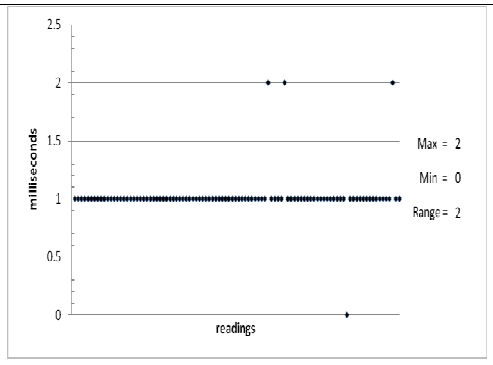
$$500\ cyles \rightarrow 1000\ ms$$

$$Half\ Cycle \rightarrow \frac{1000}{500} \times \frac{1}{2}\ ms = 1\ ms$$

As can be deduced from the background work carried out, the 10 to 15 ms precision of DateTime and GetTickCount yielded very bad results while the others produced some interesting observations.



Normal Priority

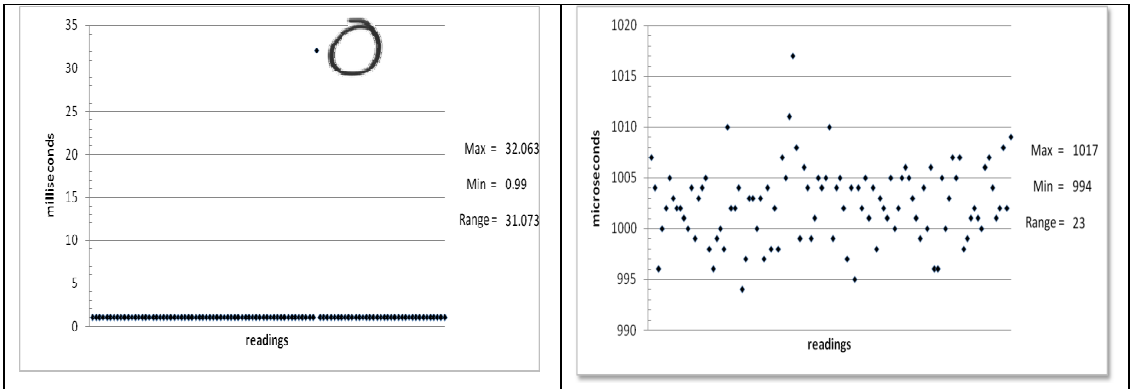


High Priority

Figure 7. Result obtained using TimeGetTime for the frequency generator input.

The TimeGetTime got most of the readings right (1 ms) but the readings also included strange values such as 12 and 11 (circled on the figure 6 above). Fortunately when the priority was set to high (right graph) refine results were obtained but still with few erratic value.

The experiment was repeated on the same PC using this time the QPC and the results was seen to be more consistent but with only one unusual readings which was however surprising high.



QPC under Normal Priority

QPC under high Priority

Figure 8. Performance of timers on windows XP using the same PC.

The jitter was considerably reduced under high priority to only 23 microseconds. The vertical axis was based on a microsecond scale to indicate the minor variations seen. Similarly when applied for Linux, increasing the priority minimizes the variations obtained. The table 6, below shows the values captured for the experiments performed on the same PC in the lab.

Table 6. Results obtained when using experiment was repeated on Linux

	Normal Priority – Captured value	FIFO priority- Captured value
GetTimeofday	Min 0 ms, Max : 7 ms	Min 610 µs Max 1007 µs
Clock_GetTime (CLOCK_MONOTONIC)	Min 0 ms: Max : 5 ms	Min 103 µs Max 1086 µs

6. CONCLUSION

The experiments performed shows that great care should be taken when using the computer as a time measuring device. The performance of the timers is not the same on all computer systems and worst of all is not repeatable. Previous work has hyped the inclusion of keystroke dynamics to enhance the actual password based schemes without impacting on its desired properties which are consistency, portability, repeatability and acceptability on all systems. Currently once a user has registered with a password, independent of the operating system and the underlying hardware, he is swiftly authenticated the next time he supplies the same password. However our research work has shown that with inclusion of typing rhythms this may not be the case as the timing captured depends on a number of factors. Moreover if the computer is already involved in other activities then he may not be able to login in as performance is degraded further.

Additionally most antivirus are automatically launch at start up without user intervention (same for screen saver) and will therefore impede on the performance of the processor. Background processes negatively affect timing on computer and running a CPU intensive alongside the timer

process causes the measurements to vary by several orders of magnitude. Similar behavior was noticed during execution of memory or disk intensive tasks.

Increasing the priority of the timer used has considerably minimized the fluctuations captured but even then some jitter is still present. Among the timers that have been used by previous researchers in the field, the QPC performs much better and this is improved further by increasing the priority at the expense of system instability However as the latter's accuracy depends on the underlying hardware the fast CPU can use the high-performance counter present in some systems. Great care should however be taken so that power-saving features do not change the processor speed.

To facilitate comparisons of performance between different algorithms, different classifiers and features, it is primordial that all users adopt the same timer which provides the minimum fluctuations in values. For the windows environment the QPC is recommended with highest priority enable. Our investigation has shown that the variation then falls to a very low value or the order of several microseconds. Although it can be considered to be more efficient that other timers it cannot achieve the 0.1 microsecond accuracy claimed by Wassenberg [35]. The Now and GetTickCount have too coarse precision (10 to 16ms) to be useful for accurate time measurements.

In Linux, changing the scheduler policy to FIFO provided remarkable low-jitter results for both GetTimeofDay and Clock_GetTime. Moreover the changes due background processes were minimum with FIFO priority set. The question that remains is how to get constant, accurate and reliable results given the jitter is inevitable in non-realtime operating systems. To this end real-time solutions available on Windows and Linux were tested. IntervalZero's Real-Time Extension[54] is a self-contained real-time subsystem that bypasses the Windows scheduler to provide the desired determinism for Windows. Ingo Molnar's Preemptible Linux patch[55] allows the entire kernel to be preempted as it replaces most spinlocks with mutexes that support priority inheritance as well as moving all interrupt and software interrupts to kernel threads.

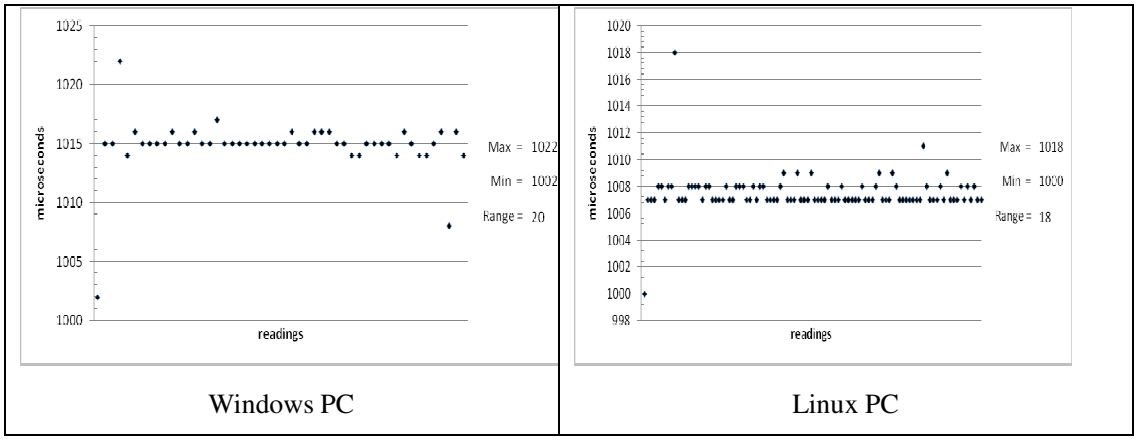


Figure 9. Performance of RealTime timers on Windows and Linux respectively.

As shown in figure 9 above, for the 1 ms second pulse experiment the range variation is much smaller than in Linux and Windows thereby boosting the use of these timers in time critical activities. The use of these timers for keystroke dynamics remains an avenue to be explorer further.

ACKNOWLEDGEMENTS

The authors would like to thank heartily those who have helped in the setting up of the different experiments and also those who have provided valuable suggestions. Our kind appreciation for

those who worked hard on making the robot strike on the keyboard although the study has not produced any valuable input for this paper.

REFERENCES

[1] R. Richardson (2003), *Computer crime & security survey* . Technical report, Computer Security Institute, CSI and Federal Business of Investigations, FBI, 2003.

[2] CP, Pfleeger (1997), "*Security in Computing*", International Edition, Second Edition, Prentice Hall International, Inc,

[3] D.L. Jobusch & A.E. Oldehoeft (1989), "*A Survey of Password Mechanisms: Weaknesses and Potential Improvements, Part 1*," Computers & Security,Vol. 8, pp. 587–604.

[4] C.P. Pfleeger (1993), *Security in Computing,* Prentice - Hall, Upper Saddle River, New Jersey.

[5] J.C. Spender (1987), "*Identifying Computer Users with Authentication Devices (Tokens)*," Computers & Security, Vol. 6, pp. 385–395.

[6] S. Garfinkel & E. H. Spafford (1996), *Practical UNIX Security*. O Reilly, 2nd edition, April 1996.

[7] S. Wiedenbeck, J. Waters , J. Birget, A. Brodskiy & Nasir Memon (2005), *Passpoints: Design and Longitudinal Evaluation of a Graphical Password System*, International Journal of Human-Computer Studies, 63(1-2), pp. 102-127.

[8] A Mészáros, Z Bankó, L Czúni(2007) , *Strengthening Passwords by Keystroke Dynamics,* IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Dortmund, Germany.

[9] D. Chuda & M. Durfina(2009), *Multifactor authentication based on keystroke dynamics,* ACM International Conference Proceeding Series; Vol. 433, Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, Article No.: 89

[10] A. Mandujano & R. Soto(2004), *Deterring Password Sharing: User Authentication via Fuzzy c-Means Clustering Applied to Keystroke Biometric Data*, Proceedings of the Fifth Mexican International Conference in Computer Science.

[11] J.R.Young, & R.W. Hammon (1969), Method and apparatus for verifying an individual's identity. Patent Number 4805222 U.S. Patent and Trademark Office, Washington, DC.

[12] R Joyce & G. Gupta (1990), *Identity Authentication Based on Keystroke Latencies*, Volume 33 Number 2 Communications of the ACM, pp. 168-176.

[13] S. Cho & S. Hwang(2006), *Artificial Rhythms and Cues for Keystroke Dynamics Based Authentication*, D. Zhang and A.K. Jain (Eds.): Springer-Verlag Berlin Heidelberg , ICB 2006 , LNCS 3832, pp. 626 – 632.

[14] N. Pavaday (2009), Transfer report, Submitted to University of Mauritius.

[15] R. Gaines et al (1980), *Authentication by Keystroke Timing: Some Preliminary Results*, technical report R-256-NSF, RAND.

[16] www.psylock.com

[17] www.biopassword.com

[18] J.A. Robinson, V.M. Liang, J.A.M. Chambers, & MacKenzie(1998), *Computer User Verification Using Login String Keystroke Dynamics*, IEEE Transactions on Systems, Man and Cybernetics —Part A, Volume 28 (2), pp 236-241

[19] O. Coltell, J. M. Badia, & G. Torres(1999), *Biometric identification system based in keyboard filtering*, In IEEE 33rd International Carnahan Conference on Security Technology, pp 203-209.

[20] J. Long, I . Nimmo-Smith and A. Whitefield (1983), *Skilled typing: A characterization based on the distribution of times between responses*, Cognitive Aspects of Skilled Typewriting,ed. W. E. Cooper (Springer, Berlin,) pp 145-196.

[21] M. Brown & S.J. Rogers (1993), *User identification via keystroke characteristics of typed names using neural networks*, International, Journal of Man-Machine Studies vol 39, pp 999-1014.

[22] L. C. F. Araújo et al. (2004), *User Authentication through Typing Biometrics Features*, D. Zhang and A.K. Jain (Eds.): ICBA 2004, LNCS 3072, pp. 694-700.

[23] D.J. Ostry (1983), Determinants of interkey times in typing, Cognitive Aspects of Skilled Typewriting, ed. W. E. Cooper (Springer, Berlin),pp 225-246,.

[24] M.S. Obaidat (1995), A verification methodology for computer systems users, Proceedings of the ACM Symposium on Applied Computing, pp 258- 262.

[25] M.S. Obaidat & B. Sadoun (1997), *Verification of computer users using keystroke dynamics*, IEEE Trans. Systems, Man and Cybernetics Part B, Vol 27, No2, pp 261-269.

[26] J. Umphress & G. Williams (1985), *Identity verification through keyboard characteristics*, International Journal of Man-Machine Studies, Vol 23, pp 263-273.

[27] S.J. Shepherd(1995) , *Continuous authentication by analysis of keyboard typing characteristics*, European Convention on Security Detection, pp 111-114.

[28] VMWARE INC., 2008. *Timekeeping in VMware Virtual Machines*, Information Guide Available from:http://www.vmware.com/pdf/vmware_timekeeping.pdf.

[29] D. JURGENS *8253/8254 PIT - Programmable Interval Timer*. Available from: http://heim.ifi.uio.no/~stanisls/helppc/8253.html

[30] Y. CHIEN-HUA (2007), SUN MICROSYSTEMS INC.,. *Solaris OS Hardware Virtualization Product Architecture*, (1.0, 11/27/07). Available from: http://www.sun.com/blueprints/1107/820-3703.pdf

[31] WINDOWS HARDWARE DEVELOPER CENTRAL, 2002. *Guidelines for Providing Multimedia Timer, Microsoft Support,* Microsoft Corp. http://www.microsoft.com/whdc/system/sysinternals/mm-timer.mspx

[32] INTEL CORP, 2007. *Intel ICH Family RTC Accuracy and Considerations under Test Conditions,* (292276-008). http://www.intel.com/Assets/PDF/appnote/292276.pdf

[33] INTEL CORP. *Intel 64 and IA-32 Architectures Software Developer's Manual*, Volume 3A: System Programming Guide, Part 1. Available from: http://www.intel.com/design/processor/manuals/253668.pdf

[34] HEWLETT-PACKARD CORPORATION, INTEL CORP., MICROSOFT CORP., PHOENIX TECHNOLOGIES LTD. & TOSHIBA CORP., 2006. *Advanced Configuration and Power Interface Specification*, (3.0b). http://www.acpi.info/DOWNLOADS/ACPIspec30b.pdf

[35] WASSENBERG J., 2007. *Timing Pitfalls and Solutions* Available from: http://www.stud.uni-karlsruhe.de/~urkt/timing_pitfalls.pdf

[36] ADVANCED MICRO DEVICES INC., 2007. *TSC Dual-Core Issue & Utility Fix*, AMD Technical Bulletin, http://developer.amd.com/assets/TSC_Dual-Core_Utility.pdf.

[37] INTEL CORP., *Intel 64 and IA-32 Architectures Software Developer's Manual*, Volume 3B: System Programming Guide, Part 2. http://www.intel.com/design/processor/manuals/253669.pdf

[38] INTEL CORP., 2004. *IA-PC HPET (High Precision Event Timers) Specification*, (Revision 1.0a, 2004) http://www.intel.com/hardwaredesign/hpetspec_1.pdf

[39] BOVET D.P., CESATI, M., 2006. *Understanding the Linux Kernel.* 3rd ed. O'Reilly

[40] MICROSOFT DEVELOPER NETWORK. *DateTime Structure*, MSDN .NET Framework Class Library. http://msdn.microsoft.com/en-us/library/system.datetime.aspx

[41] MICROSOFT DEVELOPER NETWORK. *DateTime.Now Property*, MSDN .NET Framework Class Library. http://msdn.microsoft.com/en-us/library/system.datetime.now.aspx

[42] MICROSOFT HELP AND SUPPORT, 2007. *Task Scheduler Behavior at DST Transitions and Other Events* (rev. 1.2). http://support.microsoft.com/kb/325413

[43] THE NTP PUBLIC SERVICES PROJECT, 2008. *Windows Time Service*. http://support.ntp.org/bin/view/Support/WindowsTimeService

[44] MICROSOFT DEVELOPER NETWORK. *GetTickCount Function*, MSDN Time Functions. http://msdn.microsoft.com/en-us/library/ms724408(VS.85).aspx

[45] MICROSOFT DEVELOPER NETWORK. *GetTickCount64 Function*, MSDN Time Functions.http://msdn.microsoft.com/en-us/library/ms724411(VS.85).aspx

[46] MICROSOFT DEVELOPER NETWORK. *Multimedia Timer Reference*, Windows Multimedia. http://msdn.microsoft.com/en-us/library/ms712713(VS.85).aspx

[47] MICROSOFT DEVELOPER NETWORK. *TimeGetTime*, Multimedia Functi http://msdn.microsoft.com/en-us/library/ms713418(VS.85).aspx

[48] MICROSOFT DEVELOPER NETWORK. *About Multimedia Timers*, Windows Multimedia. http://msdn.microsoft.com/en-us/library/ms704986(VS.85).aspx

[49] WINDOWS EMBEDDED HARDWARE DEVELOPER CENTRE. *HOWTO: Use a Performance Counte*r, .NET Compact Framework HowTo. http://msdn.microsoft.com/en-us/library/aa457094.aspx

[50] WALBOURN C., 2005. *Game Timing and Multicore Processors*, Microsoft Corp. [online] http://msdn.microsoft.com/en-us/library/bb173458.aspx

[51] UBUNTU MANUAL, 2008. *gettimeofday - get the date and time*. [online] http://manpages.ubuntu.com/manpages/jaunty/en/man3/gettimeofday.3posix.html

[52] UBUNTU MANUAL,2008. clock*_getres, clock_gettime, clock_settime - clock and time functions* http://manpages.ubuntu.com/manpages/jaunty/en/man3/clock_gettime.3.html

[53] MICROSOFT DEVELOPER NETWORK. *Scheduling Priorities*. [online] http://msdn.microsoft.com/en-us/library/ms685100.aspx

[54] INTEL CORP. *Enhanced Intel SpeedStep Technology How To Document* http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm

[54] www.intervalzero.com

[55] *www.linuxfordevices.com/.../Linux.../ELJOnlineBRRealTime-and-Linux-Part-2-the-Preemptible-Kerne/*

[56] www.logix4u.net

Short Biography

This work represents a continuation of the work carried out under the Fulbright research grant U105315 allocated to the first author, for working on his research thesis.

Mr. N. Pavaday is now with the Computer Science, Faculty on Engineering, University of Mauritius, having previously done his research training with the Biometric Lab, School of Industrial Technology, University of Purdue West Lafayette, Indiana, 47906 USA, (phone: +230-4037727 e-mail: n.pavaday@uom.ac.mu).

Professor K.M.S.Soyjaudah is with the same university as the first author. He is interested in all aspect of communication with focus on improving its security. He can also be contacted on the phone +230 403-7866 ext 1367 (e-mail: ssoyjaudah@uom.ac.mu)

Mr Srikant Nugessur was a student in the faculty of Engineer and has helped in the setting up of the experiments. He is now working as a programmer in a private company (email: shrikaant@gmail.com.