iButton Enrolment and Verification Requirements for the Pressure Sequence Smartcard Biometric

Neil J. Henderson[1], Neil M. White[1], and Pieter H. Hartel[2]

[1] Department of Electronics and Computer Science, University of Southampton
Southampton SO17 1BJ, United Kingdom
{njh98r,nmw}@ecs.soton.ac.uk
[2] Department of Computer Science, University of Twente, The Netherlands
pieter@cs.utwente.nl

Abstract. With the growing number of smartcard applications there comes an increasing need to restrict access to the card itself. In previous work we proposed the pressure sequence biometric, within which a biometric sensor is integrated onto the card in a low-cost and mechanically compliant manner. Using an off-card verifier we demonstrated *reasonable* discrimination between users. In this paper we consider a number of on-card verification schemes, the best of which offers an equal error rate of 2.3%. On-card computational time requirements were found to be 3.1 seconds for enrolment and 0.12 seconds for verification. Incorporating our implementation into an existing applet used 684 bytes of program space. Whilst data memory requirements are estimated to be 1400 and 300 bytes for enrolment and verification, respectively. These time and size requirements demonstrate our biometric as a practical proposition for the protection of smart cards. Experiments were performed with the iButton's Java Card platform.

1 Introduction

In the rapidly growing world of the Internet and e-Commerce, smartcards offer the potential to protect data. This illustrates just one of the functions of a smartcard driving the motivation to protect and secure access to the smartcard itself. A number of schemes have been proposed whereby some device external to the smart card captures a biometric quantity, which is subsequently verified on a smartcard platform [19]. Off-card sensors suffer from the limitation that the external biometric device must be both present and trusted. This is hard to achieve [7]. In earlier work [8] we proposed the pressure sequence method, a novel biometric, which is both mechanically and economically compliant for incorporation onto the smartcard itself. In this paper we report on the implementation of enrolment and verification functions of our biometric on Dallas Semiconductor's iButton Java Card platform [18], thereby demonstrating the viability of our biometric from the perspective of available computational resources.

I. Attali and T. Jensen (Eds.): E-smart 2001, LNCS 2140, pp. 124–134, 2001.

The pressure sequence method measures the differences with which a user taps a sequence, or rhythm, upon a simple polymer pressure sensor. The recognition of people from a series of taps or pulses has some precedent. For example early telegraphic operators recognised other operators by the way in which they keyed information. Operators developed a distinctive 'fist' or telegraphic style that could be recognised [4]. Indeed much work has been reported on the use of a person's typing style, or keystroke dynamics [9], as a route to identity verification. Rumelhart and Norman [17] offer an explanation for the discriminating basis of keystroke dynamics, suggesting that differences in typing style are due to both the physical characteristics of the hand - such as finger length and agility - and the level of motor control of a person.

The pressure sequence method seeks recognition by the pattern of pressure pulses between fingertip and pressure sensor, resulting to some measure from the biomechanical characteristics of a person's hand and wrist. The human hand is complex and offers scope for discrimination between people. This is demonstrated in its composition of 19 bones, 19 joints and 20 muscles, combining to give 22 degrees of freedom [11].

For the pressure sequence method to gain merit as a verifier, a high level of discrimination must be demonstrated. Since there are strong similarities between the pressure sequence method and that of keystroke dynamics, the discrimination methods of keystroke dynamics have been investigated for their potential to discriminate between people, based only on a sequence of taps on a single pressure sensor. This is justified in that both result from similar neurophysiological and biomechanical mechanisms and that, at a minimum, both consider time intervals between finger taps.

2 Experimental Method

To validate the pressure sequence biometric, 34 students and staff from the Electronics and Computer Science department in Southampton participated in an experiment. Each volunteer was asked to choose a short tapping sequence (typically lasting between 2 and 4 seconds), and to tap that rhythm 30 times. Data collection from each volunteer was collected in one single session in a supervised manner at all times. For further details see our earlier work [8]. Figure 1 shows the analogue waveform of a pressure sequence. It is with these macro features; Pulse Heights, Pulse Widths and Interval Durations, that the pressure sequence method aims to discriminate between a valid and an invalid user. A feature extraction algorithm was devised to pre-process the analogue waveform, generating a single column feature vector comprising of (PulseHeight(1), PulseWidth(1), IntervalDuration(1) , ... , IntervalDuration(n-1), PulseHeight(n), PulseWidth(n)), where n is the number of pulses in a sequence. Table (1) represents the sequence presented in figure 1 as a feature vector.

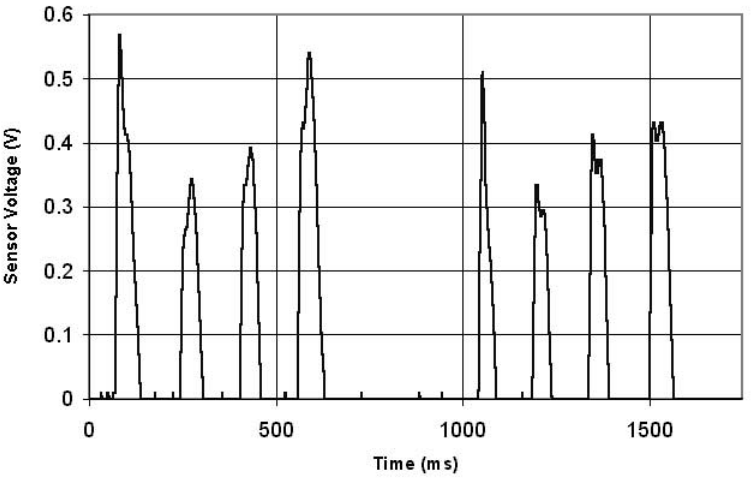

Fig. 1. Unprocessed Pressure Sequence

Table 1. Corresponding Feature Vector

Pulse1 Height	Pulse1 Width	Interval1 Duration	Pulse2 Height	Pulse2 Width	...	...	Interval7 Duration	Pulse8 Height	Pulse8 Width
52.13	115.63	215.5	36.5	105.38	...	...	246.5	48.25	129.38

3 System Architecture

The pre-processing of raw pressure sequence data occurs within a separate biometrics module, comprising of analogue to digital conversion block, alongside feature extraction circuitry. We envisage this module being implemented in a small piece of silicon providing the interface between the analogue sensor and the smartcard's processor. Figure 2 outlines this architecture. This implies that the smartcard's processor will only be presented with the feature vector representation of the pressure sequence, and will not be required to monitor live real-time data.

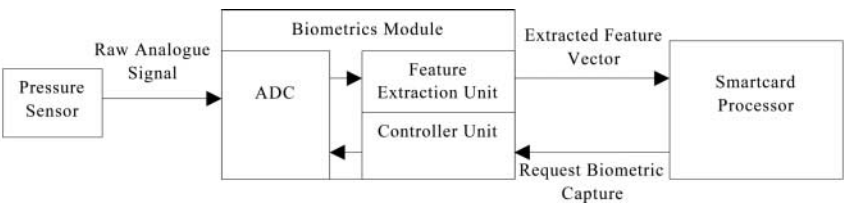

Fig. 2. Biometric System Schematic

This paper assess the viability of using a standard processor for much of the hard work; future work will be to assess the integration of the electronics comprising the biometrics module; for example signal conditioning circuitry, analogue to digital conversion unit and how to protect the wiring on the card.

4 Verification Schemes

A number of authors have reported on the successful use of keystroke dynamics as a route to identification verification [20,14,2,10,1,3,13,16,12,6,21]. The process of verification is essentially the comparison of a live biometric sample to a stored reference template, calculated during the process of enrolment. If the live sample is sufficiently close to the stored reference, then the user's identity will be considered valid. If not, the user will be rejected and denied access to the services or facilities on offer. Due to the similarities, in terms of hand physiology and motor control, responsible for underlying mechanisms of both keystroke dynamics and the pressure sequence method, successful keystroke verification schemes have been assessed (in MatLab V5.3) for their success in recognising a pressure sequences. Table 2 shows our results.

Table 2. Pressure Sequence Error Rates with Verification Scheme

Verification Scheme Keystroke Dynamics Reference(s)	EER %	FRR at 1% FAR
ℓ_1 norm - Fixed Threshold [10]	9.7	80
ℓ_2 norm - Fixed Threshold [21,3,16]	5	41
Component-Wise Linear [12]	7.2	32
MICD - Fixed Threshold [1]	5.2	37
Component-Wise non-Linear [12,16]	3.7	20
ℓ_1 norm - User Specific [10]	2.3	10
ℓ_2 norm - User Specific	3	18
Mahalanobis - Fixed Threshold [16]	3.4	42
Mahalanobis User Specific	2.4	10
MICD - User Specific	2.8	11

Figure 3 shows the effect of the acceptance tolerance on the false acceptance and false rejection rates, whilst figure 4 plots the false acceptance rate against the false rejection rate, or the Receiver Operating Curve (ROC), for the ℓ_1 norm Method. This demonstrates the inverse relationship between False Acceptance Rates (FAR) and False Rejection Rates; as the acceptance threshold for a verifier is tightened, the number of false acceptances decreases. The penalty to pay is an increase in false rejections. This property allows the designer of a verification system the flexibility to match the verifier's performance with the security

requirements of an application. The third column in table 2 shows the false rejection penalties incurred in tightening the acceptance threshold to allow only 1% false acceptances.

Table 2 shows that the two best verification schemes; namely the user specific ℓ_1 norm and Mahalanobis Distance, demonstrate impressive discrimination properties. The next section deals with these verifiers in detail, outlining their implementation on the iButton.

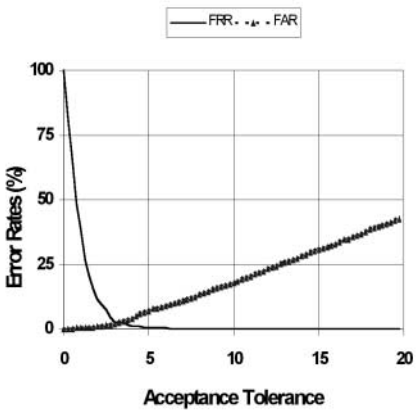

Fig. 3. FAR & FRR with Acceptance Tolerance

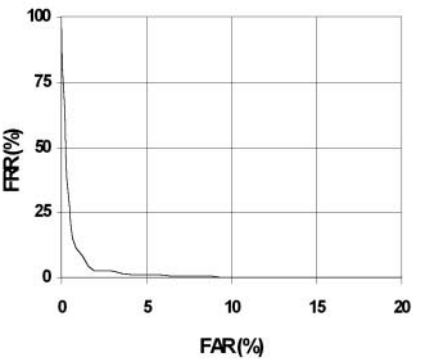

Fig. 4. ROC for L1 Norm with User Specific Threshold

5 Implementing Verifiers on the Java-Enabled iButton

A self-contained verification system embedded onto a smartcard must be capable of performing both the enrolment and verification of a user in a *reasonable* time. We accept one-off enrolment times of a few seconds, and per-transaction verification times of less than one second. To test the suitability of our verification schemes, all 10 schemes above have been translated to Java and executed on a Dallas Semiconductor Java-Enabled iButton [18]. The iButton was chosen as a test-platform, being comparable to the latest generation Java-Powered smartcards, in terms of both processing and memory resources, whilst offering accessible simulation and debug facilities. Applets were compiled using Sun Microsystem's JDK2.1.1 under version 1.10 of Dallas Semiconductor's iButton Development environment (iB-IDE). The resulting Java-Code was run on the JavaCard 2.0 compliant Java-Virtual-Machine (JVM) of the iButton. The enrolment and verification functions of each verifier were implemented in a skeletal applet, containing basic functionality to allow PC Host to iButton communica-

tion and the timing of enrolment and verification functions. The structure of our test applet is as follows:

- Retrieve and store pre-processed enrolment vectors from PC
- Record start time
- Perform enrolment (a number of times)
- Record finish time
- Send total enrolment time and reference vector to PC

Verification times were measured in the same way as enrolment times, with the verification function called in place of enrolment. Times were recorded using the iButton's real time clock, which has a resolution of only 1 second. The applet executes by firstly loading a user's enrolment feature vectors, pre-processed from their enrolment sequences, to the iButton. The enrolment process is then repeated a number of times. Time is recorded using the iButton's clock value, before and after a number of calls to the enrolment process. From these values an average execution time is calculated. Total time required, along with the computed reference vector, is transmitted to the Host PC. Implementing the process in this way avoids lengthy PC to iButton communications which are not involved in the enrolment (or verification process). This simulation assumes that the enrolment samples have already been captured, pre-processed, and are now available to the processor. Data capture and feature extraction pre-processing is the responsibility of the biometrics module, outlined in figure 2.

5.1 Specific Implementation Issues

There are two fundamental limitations to running our verifiers on the JavaCard platform; the restriction to 32-bit integers and hence the lack of floating-point data-types; and the lack of elementary mathematical functions, such as square-roots.

The fixed threshold version of one of the most successful verification schemes, the ℓ_1 norm is defined as:

$$\|\mathbf{M} - \mathbf{T}\|_1 = \sum_{i=1}^{d} |m_i - t_i| \tag{1}$$

where $\mathbf{M}$ is the d dimensional reference-vector, generated from the mean of each of the components in the user's enrolment vectors, $\mathbf{T}$ is a d dimensional test vector, and m and t are the components of $\mathbf{M}$ and $\mathbf{T}$ respectively.

The identity of a user will be accepted if:

$$\|\mathbf{M} - \mathbf{T}\|_1 \leq \theta \tag{2}$$

where θ is the acceptance threshold.

Calculation of the ℓ_1 norm is achievable with no loss of precision under the restriction of the 32-bit integer data type. Using a fixed acceptance threshold,

however, resulted in a rather uninspiring equal error rate (EER) of 9.7% (see Table 2), since the same acceptance threshold is used for all users irrespective of their natural variance.

To improve upon this rate, the variation of a user's enrolment vectors from their reference vector can be taken into account. This is performed in the manner of Joyce [10]. If each of the enrolment vectors are $\mathbf{S}_1$, $\mathbf{S}_2$, ..., $\mathbf{S}_n$, respectively, then the ℓ_1 norm between the reference vector, $\mathbf{M}$ and $\mathbf{S}_j$ is calculated for j = 1 to the number of enrolment vectors. The mean deviation $\overline{D}_{Enrolment}$, and standard deviation, $\sigma_{Enrolment}$, of these distances are calculated and used to decide an acceptance threshold on a per user basis. This results in acceptance of the user's identity if:

$$\|\mathbf{M} - \mathbf{T}\|_1 \leq |\overline{D}_{Enrolment} - \tau \cdot \sigma_{Enrolment}| \tag{3}$$

where τ is the globally-set acceptance tolerance.

The problem for the integer-only Java Card system is that calculation of $\sigma_{Enrolment}$, the standard deviation of the distance between enrolment vectors and reference vector, requires calculation of a square root:

$$\sigma_{Enrolment} = \sqrt{\frac{\sum_{j=1}^{n}(\mathbf{S} - \mathbf{M})^2}{(n-1)}} \tag{4}$$

Whilst the immediately obvious method for calculation of square roots, the Newton-Raphson method [15], is well known and used, it suffers from a number of problems. Firstly, it is an iterative method, whose efficiency depends upon the quality of the initial guess. Secondly, applied to integers in its native form, the Newton-Raphson method will infinitely oscillate between two integers, above and below a real non-integer root [5]. Crenshaw [5] has devised a simple algorithm for the calculation of square roots. His method is essentially a search through all possible integers until the integer part of a non-integer root is found. The simplest form of this is to search all possible integer square roots, x, for $\sqrt{N}$ until $x^2 \geq N$. The integer root of N is then the exit value of x, minus one. Crenshaw provides a more efficient implementation, arising from the observation:

$$(x+1)^2 = x^2 + (2x+1) \tag{5}$$

This means that to check each new possible square root, $(2x + 1)$ merely has to be added to the previous square. The Java code implementation for this is presented as follows:

Java Code for Integer Square Root Function

```
public static int SQRT(int a) {
    int square = 1;           // x=1: 1st Integer Square Root
    int delta = 3;            // (2x+1), for x=1

    while(square<=a) {
        square+=delta;        // (x+1)^2 = x^2 + (2x+1)
        delta +=2;            // Next value for (2x+1)
    }
    return (delta/2 - 1);     // square is now > a, so find
                              // previous value of x
}
```

Although faster fixed cycle-length square root functions exist [5], the time required for execution of the enrolment process using this implementation was measured to be 3.1 seconds - well within the bounds of a *reasonable* duration. 20 square roots were required for the enrolment process, each iterating 49 times. Verification required only 0.12 seconds for completion.

Although the final values for the mean distance and its standard deviation between enrolment vectors and the user's reference vector will be truncated to integer levels, the fractional loss is small relative to the distances involved and is not expected to cause any significant reduction in the verifier's accuracy. Table 3, presenting the floating point and integer results from one user, illustrates this point.

Table 3. Comparison of Floating Point and Integer Results

...	Mean Distance	Standard Deviation
Floating Point Result	221	49.29
iButton's Integer Result	222	48

The second most discriminating method; the Mahalanobis distance verifier is defined as follows:

$$(\mathbf{M} - \mathbf{T}) \cdot \mathbf{V}^{-1} \cdot (\mathbf{M} - \mathbf{T}) \leq |\overline{D}_{Enrolment} - \tau \cdot \sigma_{Enrolment}| \tag{6}$$

where $\mathbf{M}$, $\mathbf{T}$, $\overline{D}_{Enrolment}$, τ and $\sigma_{Enrolment}$ are as defined in (1) and (3), above. $\mathbf{V}^{-1}$ is defined as the inverse of a square $d \times d$ matrix whose leading diagonal is composed of the variances from each dimension of the enrolment vectors and all other elements are zero.

This leads us to the problem that multi-dimensional arrays are not supported under Java Card 2.0, and as a result matrix manipulation will be both convoluted

and time consuming. The solution rests with the analytical reduction of the left-hand side of (6) to give:

$$(\mathbf{M} - \mathbf{T}) \cdot \mathbf{V}^{-1} \cdot (\mathbf{M} - \mathbf{T}) = \sum_{i=1}^{d} \frac{(m_i - t_i)^2}{v_i} \tag{7}$$

where v_i is the variance of the component of the enrolment vectors. Equation (7) now offers a route to the direct computation of the Mahalanobis distance verifier. There is one further catch, however. v_i as the variance of each vector component, is the square of the standard deviation for that component. It is hence likely to be of comparable magnitude to that of the component values. The integer division, therefore, of $(m_i - t_i)^2$ by v_i is extremely likely to result in the loss of a significant fractional part of the result, further compounded by the sum across all components. Pre-multiplying each numerator by $10^{RequiredPrecision}$ and post-dividing the sum by the same enables retention of the fractional information. Table (4) provides the floating point, the integer and the pre-multiplied integer results.

Table 4. Comparison of Floating Point, Integer and Pre-Multiplied Integer Results

...	Mean Distance	Standard Deviation
Floating Point Result	17.5	5.56
iButton's Integer Result	10	4
iButton's Pre-Multiplied Integer Result	17	5

Execution of the Mahalanobis enrolment process required 4.5 seconds, whilst verification required 0.16 seconds.

5.2 Further Results

The enrolment and verification times for the other verification schemes were also measured on the iButton. In addition the program size for enrolment and verification functions, combined was recorded. Table (5) presents a comparison of these quantities, along with a repetition of the Equal Error Rates, for convenience.

6 Conclusions

The Java Card 2.0 platform does not support floating-point arithmetic. We show that 32-bit integer arithmetic offered by the iButton implementation of Java Card 2.0 is sufficient to implement on-card verification and enrolment for our

Table 5. Results for all Verifiers

Verification Scheme	Enrolment Time (Seconds)	Verification Time (Seconds)	Program Size (bytes)	EER%
ℓ_1 norm - Fixed Threshold	1.2	0.12	296	9.7
ℓ_2 norm - Fixed Threshold	1.2	0.26	356	5
Component-Wise Linear	1.2	0.19	285	7.2
MICD - Fixed Threshold	1.2	0.15	319	5.2
Component-Wise Non-Linear	2.9	0.29	524	3.7
ℓ_1 norm - User Specific	3.1	0.12	684	2.3
ℓ_2 norm - User Specific	4.2	0.25	707	3
Mahalanobis - Fixed Threshold	2.4	0.16	550	3.4
Mahalanobis User Specific	4.5	0.16	752	2.4
MICD - User Specific	8.5	0.16	704	2.8

pressure sequence biometric. We develop the necessary mathematics and estimate the errors introduced by representing real data as integer data.

We measure the execution times and space requirements required by our implementation of verification and enrolment and show that they are within reach for a typical Java Card platform.

We present an architecture for a complete on-card pressure sequence biometric. In a previous paper we presented a method of integrating the physical sensor the plastic substrate of the smart card. In this paper we show that the processing capabilities required can be provided by a standard smart card. Future work includes an investigation into integrating the analogue electronics and the wiring on a standard smart card.

References

1. S. Bleha, C. Slivinsky, and B. Hussien, *Computer access security systems using keystroke dynamics*, IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990), no. 12, 1217–1222. 127
2. S. A. Bleha and M. S. Obaidat, *Computer user verification using the perceptron algorithm*, IEEE Transactions on Systems, Man and Cybernetics **23** (1993), no. 3, 900–902. 127
3. M. Brown and S. J. Rogers, *User identification via keystroke characteristics of typed names using neural networks*, International Journal of Man-Machine Studies **39** (1993), 999 – 1014. 127
4. W. L. Bryan and N. Harter, *Studies in the physiology and psychology of the telegraphic language*, Psychological Review **4** (1897), 27–53. 125
5. J. Crenshaw, *Integer square roots*, Embedded Systems Programming (1998), no. 2. 130, 131
6. J. D. Garcia, *Personal identification apparatus*, US Patent 4 621 334, November 1986. 127

7. G. Hachez, F. Koeunne, and J-J. Quisquater, *Biometrics, access control, smart-cards: A not so simple combination*, Smart Card Research and Advanced Applications (Bristol, UK), IFIP/TC8 Fourth Working Conference on Smart Card Research and Advanced Applications, Kluwer Academic Publishers, September 2000, ISBN: 0-7923-7953-5, pp. 273–288. 124
8. N. J. Henderson and P. H. Hartel, *'pressure sequence' - a novel method of protecting smart cards*, Smart Card Research and Advanced Applications (Bristol, UK), IFIP/TC8 Fourth Working Conference on Smart Card Research and Advanced Applications, Kluwer Academic Publishers, September 2000, ISBN: 0-7923-7953-5, pp. 241–256. 124, 125
9. A. Jain, R. Bolle, and S. Pankanti, *Biometrics: Personal identification in networked society*, Kluwer, Boston, 1998, ISBN 0792383451. 125
10. R. Joyce and G. Gupta, *Identity authentication based on keystroke dynamics*, Communications of the ACM **33** (1990), no. 2, 168–176. 127, 130
11. E. R. Kandel, *Principles of neural science*, Elsevier, North Holland, 1981. 125
12. J. Leggett and G. Williams, *Verifying identity through keystroke characteristics*, International Journal of Man-Machine Studies **28** (1988), 67–76. 127
13. F. Monrose and A. D. Rubin, *Keystroke dynamics as a biometric for authentication*, Future Generation Computer Systems **16** (2000), 351–359. 127
14. M. S. Obaidat and B. Sadoun, *Verification of computer users using keystroke dynamics*, IEEE Transactions on Systems, Man and Cybernetics - Part B - Cybernetics **27** (1997), no. 2, 261 – 269. 127
15. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipies in c*, Cambridge University Press, Cambridge, United Kingdom, 1993. 130
16. J. A. Robinson, V. M. Liang, J. A. M. Chambers, and C. L. MacKenzie, *Computer user verification using login string keystroke dynamics*, IEEE Transactions on Systems, Man and Cybernetics - Part A - Systems and Humans **28** (1998), no. 2, 236 – 241. 127
17. D. E. Rummelhart and D. A. Norman, *Simulating a skilled typist: A study of skilled cognitive-motor performance*, Cognitive Science **6** (1982), no. 1, 1–36. 125
18. Dallas Semiconductor, http://www.ibutton.com/, March 2001. 124, 128
19. Siemens, http://www.silicon-trust.com/, March 2001. 124
20. D. Umphress and G. Williams, *Identity verification through keyboard characteristics*, International Journal of Man-Machine Studies **23** (1985), 67–76. 127
21. J. R. Young and W. Hammond, *Method and apparatus for verifying an individual's identity*, US Patent 4 805 222, February 1989. 127