# Identity Security in Biometric Systems based on Keystroking*

Lucjan Hanzlik and Wojciech Wodo
*Faculty of Fundamental Problems of Technology, Wroclaw University of Technology, Wroclaw, Poland*

Keywords: Identity, Security, Keystroking, Biometry, Keylogger, Impersonation.

Abstract: The most valuable element of biometric security systems are the personal features of its users. Characteristics of individuals are unique and must be protected. We focus in this paper on methods of protection of user identity in systems based on keystroking. Our approach assumes giving minimal information to adversaries and the best responsiveness of the system regardless of user representation or possible usage. We consider keystroking not only in the context of keyboard, but also touch screen, pin pad and any other input device that could be used for typing. We present as results several complete security solutions that are applicable for software as well as hardware systems.



## 1 INTRODUCTION

Biometric security systems have been used for many years. They are based on physical and behavioral characteristics inherent in the body. Keystroke dynamics is a behavioral biometric feature defined by our way of typing. Among its advantages, it does not require an additional reader or scanner because every computer is equipped with some kind of keyboard. Another advantage is that keystroke dynamics can be used for continuous authentication of the user; that is, as long as the user is at the workstation, the system can continuously verify his identity. In numerous papers authors consider the effectiveness of authorization or identification systems (Monrose and Rubin, 1997), the decreasing of *False Acceptance Rate* or *False Rejection Rate*, more accurate models of users (Bergadano et al., 2002) and other issues connected with positive usage of system. In this paper we would like to focus on protecting the keystroke identity of individuals. Once stolen, biometric data can be used indefinitely, thus it is important to secure this information.

**Structure and Contribution of Paper.** In Section 2 we describe briefly how keystroking systems work, what kind of data we collect and how we measure these data using toolkits that we developed for intercepting raw data from the keyboard—*keyloggers*. We also identify security threats related to the stealing of user identity and impersonation. Section 3 is devoted to issues connected with protecting an individual's identity. There we present our protection algorithms with implementation details and consider many use cases. A hardware approach to the problem of identity protection is discussed in Section 4. There we also develop our microprocessor device for USB keyboards and consider protection of Smart Card terminals/readers. Analysis of the algorithms' effectiveness is described in Section 5. The final Section 6 contains our conclusion and recommendation for future work.

**Related Work.** There are many papers and implementations based on keystroke dynamics. Authors consider mostly classification algorithms (Zahid et al., 2009) or extraction features from inputted text (Zhong et al., 2012). Discussion about discrimination of digraphs and trigraphs for free text can be found in (Sim and Janakiraman, 2007). An interesting approach to identity verification based on the analysis of the typing rhythms of individuals on different texts is presented in (Bergadano et al., 2003), but to our knowledge the first attempt to implement software protection from a bot attacker is in (Stefan et al., 2012). Work on the security of user identity has been carried out in part by (Klonowski et al., 2012) and (Wodo, 2012).

## 2 KEYSTROKING SYSTEM

Every keystroking system first collects data from the

*This research was supported by European Union POKL 4.1.1 Mloda Kadra and FNP Ventures/2012-9/4 scientific projects


Hanzlik L. and Wodo W..
Identity Security in Biometric Systems based on Keystroking.
DOI: 10.5220/0004526405240530
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 524-530
ISBN: 978-989-8565-73-0

user and checks its compliance during authentication or identification. The process of gathering data could be performed once (e.g. the user may type his login and password a dozen times or a free text sample of demanded length) as well as every time the individual uses the system. Such a solution can be used for continuous authentication to assure that only legitimate users have access to a certain resource, e.g. workstation. Inserted data is processed as digraphs (two consecutive keys) or longer n-graphs depending on the approach (considering context or not).

## 2.1 Data Collection

In the case of keystroking dynamics the user information can be derived from timings coming from a keyboard or other input device, e.g., touch screen or pin pad. It is essential to gain access to raw data in real-time, because we operate on time measured in milliseconds. Any disruption or additional signal processing can alter timings. We focused on gathering the keyboard events *keyUp* and *keyDown* consisting of the key pressed or released and the timestamp of each event. This allows us to calculate *dwell time* and *flight time* timings for each digraph or even each n-graph.

One way to collect such data is to install a hardware keylogger on USB/PS2 port and communicate with it by simulating keyboard events. Another approach—our choice—is to intercept keyboard events in software before the operating system processes them. This solution is highly dependent on the OS; one can, for example, write a new function for the keyboard interrupt, install a daemon/service or hook into the kernel.

We developed software toolkits to facilitate the process of data collection for Windows XP/7 and the Linux operating systems. The implementation for Windows XP uses *WinAPI's* `SetWindowsHookEx()` function for connect global hook, `WH_JOURNALRECORD` to queue of messages and the `GetSystemTime()` function to deal with timestamps. For Windows 7 we simply used the `GetAsyncKeyState()` method to get all keyboard events. For the Linux operating system we captured raw keyboard data from the event device node for example `/dev/input/event1`.

## 2.2 User's Representation

Generally, our considerations are independent of the adopted model user's representation because we work on raw data that is coming directly from an input device. However, in the case where we want to impersonate a user, we have to take into account way of storing and transforming keystroke data (3.4).

For each digraph we construct a histogram of timings (divided in 1 ms slots for dwell- and flight- time). We reduce outliers and interpolate the gaps among the most common values in order to avoid sequences of the identical timings (it can happen in case of too little data). In the end we normalize these histograms and use them as a probability distribution of particular timings.

## 2.3 Security Threat

There are risks associated with the use of keystroke dynamics in authorization systems. A user's keystroke identity is permanent like other biometric features. It is not an easy task for a person to change his typing rhythm. Thus, once an adversary captures a user's identity he may use it multiple times to gain access to the system. This is a security threat for systems based on authorization by keystroke dynamics since such systems provide no additional security. Other security measures must be applied to make those systems secure.

Another security threat concerns the privacy of users. As shown in (Klonowski et al., 2012) the keystroke identity may be used to uniquely identify a user. An adversary may capture the data flowing to a system and create profiles of its users. Since the adversary is able to create such profiles he may distinguish between users and break the anonymity of systems that are designed to preserve it. Note that this may be also a privacy threat concerning electronic IDs. An adversary may capture the pin pad input while the user is typing his PIN. Thus even if communication and authorization protocols for eIDs preserve privacy, such side channel attacks may leak information about users activities.

# 3 IDENTITY PROTECTION

In this section we would like to focus on protecting the biometric data of legitimate users, which could be intercept during daily tasks. As mentioned in 2.3, once stolen, biometric data can be used indefinitely to impersonate a user because it is mutually correlated with the individual. Our aim is to change the user's original timings and therefore provide no additional information to the attacker about the user's typing rhythm or even that any protection system is in place. We want the adversary to think he is gaining valuable information, but in fact is collecting useless data. The longer this situation persists the better for

the user because if the attacker is unaware of the presence of security software, he will not try to disable it.

All presented solutions are based on a buffer and delay algorithm, and therefore require access to time indicators. Keyboard events (*key down* and *key up*) are intercepted before reaching the application level of the operating system and are stored in memory. A timestamp of each event is recorded, and after an appropriate delay the event is released, similar to (Klonowski et al., 2012).

In our research we have considered two types of timings, i.e., *flight time* and *dwell time*, in order to provide the best possible security.

## 3.1 Constant Delay Algorithm

This kind of algorithm seems to be the easiest to implement. It works by simply gathering keyboard events in a buffer (queue) and releasing them one by one after *d-time* period passes. Parameter *d* regulates the delay and obfuscation threshold. This algorithm works perfectly for timings under threshold *d* generating uniform sequences of timings for typed data. Above this threshold it works in immediate mode (that is, it does not delay timings) to keep the system responsive and does not protect the user. Such an approach provides a very high level of security for at least moderate-skilled typists. It is unfortunately useless for people who cannot maintain a typing rhythm at the minimum level of the set threshold.

According to both our research and (Song et al., 2001) the vast majority of *flight times* are below 400–500 ms. The average delay for *flight time* is about 200 ms. For that reason we recommend setting *d* between 200–250 ms. Such a setting allows one to obfuscate almost every timing. Recall that a delay is generated after every event, i.e. *key down* and *key up*. So in case of typing digraph **th**, we would have the following sequence of events: **keyDown(t), keyUp(t), keyDown(h), keyUp(h)**. Events could be generated in another order because the user can hold a key for a longer time and press another one, therefore we have to consider this scenario as well. Practically, this solution gives us about 400–500 ms time for typing another character of input text providing protection of our identity. If there is a well trained typist or a beginner, parameter *d* could be align to avoid losing fluency of work for the former and protection of the user in the latter. We implement these considerations in Algorithm 1.

**Algorithm 1:** CONSTANT TIME DELAY.

```
   Input: d: 10-bit integer (delay in [ms])
 1 begin
 2     Buffer : Quene ;
 3     while wait for signal T do
 4         T → INPUT ;
 5         if Buffer.Empty() then
 6             T → OUTPUT ;
 7             Set_Timer(d) ;
 8         else
 9             Buffer.Enqueue(T) ;
10     ;
11     proc Timer_Timeout ;
12     begin
13         Buffer.Dequeue() → OUTPUT ;
14         if !Buffer.Empty() then
15             Set_Timer(d) ;
```

## 3.2 Random Noise Algorithm

It is obvious that the solution provided by Algorithm 1 is easily detected by the adversary. In order do enhance stealth mode of our protection system we consider some modification of the previously described algorithm. First of all, we decide for each keyboard event whether it will be delayed or not (with probability 0.5). Next, we generate a set of timing extensions between 15 and 70 ms. Such an approach allows us to create an impression of typing by a real user. This method is applicable only for well-trained typists, because the random noise algorithm will only slightly affect a user's timings, and therefore cannot provide a long delay. In fact, this is not real random noise approach because we do not have the option to shorten timings. This is because we modify delays on the fly, and so this algorithm is more like the moving threshold method. Algorithm 2 includes the above assumptions.

## 3.3 PUF Method

The physical unclonable function (PUF) is a function that is easy to evaluate but hard to predict. In addition, it is easy to create a PUF device but impossible (in practice) to make a copy of it. We show how to use those features to create Algorithm 3 which changes the keystroke identity of a user in a static and unclonable way. The idea of this solution is that a user may create two or more keystroke identities (using PUF based devices) and switch between them (using, for example, passwords or private logins) whenever he

**Algorithm 2:** RANDOM NOISE DELAY.

```
1  begin
2      Buffer : Quene ;
3      while wait for signal T do
4          T ← INPUT ;
5          if Buffer.Empty() then
6              T → OUTPUT ;
7              Timer_Evoke();
8          else
9              Buffer.Enqueue(T) ;
10     ;
11     proc Timer_Evoke ;
12     begin
13         if Random(0,1) ≥ 0.5 then
14             d ← PickAtRandom(15,70);
15         else
16             d ← 0;
17         Set_Timer(d) ;
18     ;
19     proc Timer_Timeout ;
20     begin
21         Buffer.Dequeue() → OUTPUT ;
22         if !Buffer.Empty() then
23             Timer_Evoke() ;
```

likes. Even if an adversary has access to the device he is unable to clone it.

Let $\phi : \{0,1\}^{h+7} \rightarrow \{0,1\}^{h+7}$ be a PUF function and let $H : \{0,1\}^{h} \rightarrow \{0,1\}^{h}$ be a hash function, e.g. SHA1 with $h = 160$. Algorithm 3 takes as input a password *pass* and computes the real time gap between two consecutive keyboard events (stored in a buffer) and changes this time using the password and the PUF function. Due to space reasons we only show the function used to transform the timing between the two consecutive keyboard events.

Note that we do not change the three most significant bits of the timings. Thus, the `Trans_Time` procedure computes a value which differs from the original one by at most 128 ms. However, such small differences change the keytroke characteristics of a user

**Algorithm 3:** PUF BASED TRANSFORMATION.

```
   Input: pass : String, time: 10-bit integer
1  begin
2      proc Trans_Time ;
3      begin
4          pass_H ← H(pass) ;
5          res ← φ(pass_H || time_9 ... time_0);
6          return time_9 time_8 time_7 res_6 ... res_0
```

and make it difficult for the adversary to determine whether the user uses protection against attacks. Notice further that if the adversary has access to the PUF based device of a user he may send arbitrary challenges to the $\phi$ function and store the results. However, because the passwords for each identity are secret, the adversary would have to guess the $pass_H$ value for the user's identities or challenge all possible values, which is computationally infeasible for a large $h$. Thus, unless the user's passwords are known an adversary is not able to clone the user's device and to link the user's identities.

### 3.4 Pretending other User Algorithm

This solution cannot be implemented in real time, because it requires a sequence of characters in order to calculate new timings. The algorithm works as follows: first, it buffers keystrokes until synchronization—signaled by a break in typing ($\geq$ 500 ms). Next, new timings are calculated on the basis of data previously gathered from other users (according to 2.2). Finally, the algorithm flushes the buffered keystrokes with the modified delays. In essence we replace the original timing of a particular digraph with those taken from the distribution defined for another user.

We recommend this solution in cases where real-time communication is not critical and we can afford to capture text first and later inject it (e.g. login form). An adversary will have serious problems identifying this security method because the algorithm pretends to be another individual. Generally this solution provides high security and its quality depends only on the user's data gathered previously.

### 3.5 Binary Representation Probability Algorithm

We use the idea of Algorithm 3, i.e. changing the gap between two consecutive keyboard events. However, in Algorithm 4 we use a probabilistic change. The algorithm takes as input a parameter $p \in (0,1)$ and flips the $i$-th bit of the gap with probability $p^{i+1}$.

Note that while the significance of the bits raises the probability of a change falls.

## 4 HARDWARE DEVICE

In order to provide protection of a user's biometric identity we could implement our security algorithms in external hardware devices (similar to hardware keylogger) or include them in internal micro-

**Algorithm 4:** BINARY REPRESENTATION PROBABILITY.

```
   Input: p: double
1  proc Trans_Time ;
2  begin
3      res ← time;
4      for i ∈ 0,...,9 do
5          r ← Rand(0,1);
6          if r < p^{i+1} then
7              res_i ← res_i ⊕ 1
8      return res_9 ... res_0
```

processor systems. Such an approach guarantees very high security, because an adversary must have physical access to the device in order to disable the protection or gain raw data. This solution eliminates any kind of software attack as well, because the application layer gets already modified data. Command communication between the security unit and the user can be achieved by simulating input device events (e.g. keyboard keystrokes). This special service mode could be invoked by inserting a code sequence, compare (Wodo, 2012).

### 4.1 Keyboard Security Unit

We propose a simple microprocessor device to protect the keyboard. It works as a *man in the middle* unit—plugged in between the workstation and the keyboard. This solution could be implemented in a form of USB/PS2 connector, which transforms the signal and adds appropriate delays.Each of the algorithms presented earlier may be adopted for this kind of device (depending, of course, on the capabilities of the selected microprocessor).

In our opinion such a solution provides the highest level of security because it completely isolates any adversary from any influence on the protection system. In addition, it is much quicker than software application and does not use computer resources. The drawback of this method is that it can be applied only to the externally connected input devices.

### 4.2 Smart Card Security

We stated earlier that the anonymity of eID users may be threatened by an adversary that captures the pin pad input. Note that for protection we must use an off-card solution since the user types his pin on the terminal/reader side. The adversary may be the terminal owner or producer so any countermeasures applied inside the terminal may also be unsafe. One possible solution is for users to use their own pin pad like devices which simulate the input of the user on the real pad. This approach has some disadvantages. For example, users would have to carry additional baggage, and there is also the issue of powering such a device. Another possibility is to use an additional RFiD based Smart Card. A user would have to first insert his eID into the terminal/reader and then use the RFiD card (with a unique number stored in it) to authorize transaction instead of typing his PIN. Note this solution is much cheaper and less complex that the first one. However, is some cases it is desirable to demand that a user types the PIN and that the transaction is not performed by a computer (which might be the case if we use the second solution).

## 5 EFFICENCY ANALYSIS

In this section we present and discuss the results of our tests. We captured and recorded sample data using our keylogger-like program (see 2.1) (about 14000 key events). The timings between corresponding keyboard events (i.e. *key up*, *key down*) are presented in Figure 1. Note that the data was truncated to 300 ms due to the fact that, for our test user, a higher timing corresponds to a pause in typing.

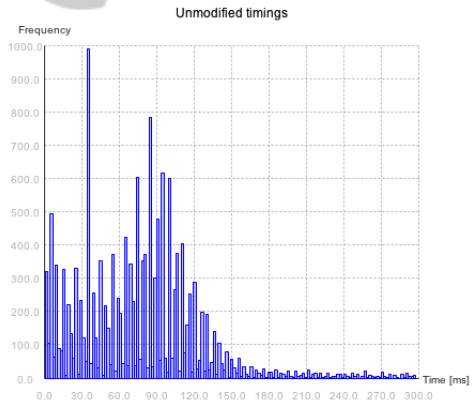


Figure 1: Unmodified Timings.

Then, we transformed the raw data according to algorithms presented in Section 3. Note that the best security level gives the constant time delay algorithm. It can be seen in Figure 2 that events occur with constant delays. Thus, the adversary receives no information about the typing rhythm of the user. However, he is aware that a protection is used by the individual.

As we can observe from Figure 2, random noise delay algorithm obfuscates with certain probability ($p$) timings by adding random delay at the level of dozens of milliseconds. An adversary cannot recog-

nize which timings were delayed, which results in a high security level. Moreover, thanks to changed timings distribution, which is similar to those of a real user, it is much harder to determine that any protection is used in that case. Admittedly this method affects significantly short timings so it is more appropriate for well-trained typist. Manipulating $p$ results in shifting the occurrences of real timings to higher values and decreasing the cardinality of short timings.

The PUF based algorithm, as can be seen from Figure 2, changes the occurrence of keyboard events in a significant but deterministic way. Note that a different passwords would give different results. It is obvious that such characteristics give the adversary no substantial information about the user or whether he is using any protection. However, a context analysis of the data discloses that protection is used. Note that it may happen that for a given PUF function two occurrences of a digraph (e.g. **ok**) differ by about 1–5 ms in the original message, while after the transformation the difference can be much higher. Thus, this solution may significantly change the variation of some digraphs, making it easy for an adversary to detect this kind of protection.

Note that the binary representation probability algorithm flattens the histogram. As can be observed from Figure 2 (probability $p = 0.5$) with the increase of the probability $p$ the timing distribution becomes more similar to uniform distribution. Obviously, the adversary is able to detect that the user is using protection if we use a higher probability $p$. However, it should be harder for the adversary to gain any information about the identity of the user as the histogram becomes flatter.

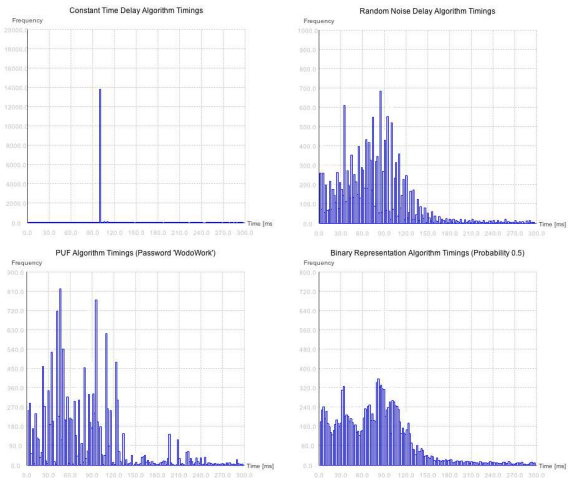


Figure 2: Modified Timings.

# 6 CONCLUSIONS

In this paper we consider security algorithms working mostly in real time and what makes them difficult to implement in real environments (responsiveness of the system). If we could record the whole stream of data and replay it with modified timings we would be able to create any sequences of keyboard events we want.

The presented solutions provide, in our opinion, a high security level, but could effect using keystroking as a method of verifying identity. In that case it is a tradeoff between usability and security. For instance the constant-time algorithm destroys the possibility of distinguishing users, on the other hand the PUF based algorithm transforms only one user identity into another. We consider protection of identity regardless of the user's representation model which means that the quality of intercepted information depends on assumptions of adversary (e.g. is any protection used, what kind of algorithm is used).

We highly recommend using a hardware-based solution, if possible, because it has the following advantages: (1) It cannot be disabled remotely; and (2) It can perform more effective calculations without burdening the CPU of the workstation.

# REFERENCES

Bergadano, F., Gunetti, D., and Picardi, C. (2002). User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur.*, 5(4):367–397.

Bergadano, F., Gunetti, D., and Picardi, C. (2003). Identity verification through dynamic keystroke analysis. *Intell. Data Anal.*, 7(5):469–496.

Klonowski, M., Syga, P., and Wodo, W. (2012). Some remarks on keystroke dynamics - global surveillance, retrieving information and simple countermeasures. In *SECRYPT*, pages 296–301.

Monrose, F. and Rubin, A. D. (1997). Authentication via keystroke dynamics. In *ACM Conference on Computer and Communications Security*, pages 48–56.

Sim, T. and Janakiraman, R. (2007). Are digraphs good for free-text keystroke dynamics? In *CVPR*.

Song, D. X., Wagner, D., and Tian, X. (2001). Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, pages 25–25, Berkeley, CA, USA. USENIX Association.

Stefan, D., Shu, X., and Yao, D. D. (2012). Robustness of keystroke-dynamics based biometrics against synthetic forgeries. *Computers & Security*, 31(1):109–121.

Wodo, W. (2012). Kradziez tozsamosci i podszywanie sie pod innych uzytkownikow w systemach biometrycznych opartych o keystroking. In *Interdyscy-*

*plinarnosc badan naukowych 2012*, pages 461–466, Wroclaw, Poland. Oficyna Wydawnicza Politechniki Wroclawskiej.

Zahid, S., Shahzad, M., Khayam, S. A., and Farooq, M. (2009). Keystroke-based user identification on smart phones. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 224–243, Berlin, Heidelberg. Springer-Verlag.

Zhong, Y., Deng, Y., and Jain, A. K. (2012). Keystroke dynamics for user authentication. In *CVPR Workshops*, pages 117–123.