Handwritten Signature Confidentiality through Segmented Curve Generation

Ebenezer R. H. P. Isaac [1] and Dr. J. Visumathi [2]
Department of Comuter Science and Engineering
Jeppiaar Engineering. College, Chennai, Tamil Nadu, India

Abstract—**While Handwritten Signature Verification (HSV) techniques thwart attacks on the security associated with authentication, the signature itself may be prone to attacks with respect to data security. Hence it is vital to store the signatures as encrypted parameters. But storage of the signature as parameters of features would mean that it would not be possible to retrieve the original signature image for cross referencing in case of any error such as a false rejection. Though image encryption may provide end-to-end security, it requires the image to be reproduced at least temporarily on to the memory of the system on which the signature verification is to be performed. This leads to the vulnerability of what is called an image shadowing attack also known as a dead-forensic attack. This article poses a technique that allows the HSV system to store the original reference signatures as a set of encrypted curve equations providing added confidentiality to the signatures as well as facilitating the signatures to be retrievable in case of any discrepancies in the signature verification algorithm.**

Keywords—Newton's Interpolation; Security and Protection; Handwritten Signature Verification

I. INTRODUCTION

Handwritten signature verification is a process of ascertaining whether a given test instance signature belongs to the claimed signing identity provided the assumed true characteristic signature of that identity is trained within the system.

The HSV system is supposed to have a constant connectivity to a database of original signatures signed by the candidates when they are first registered (trained) into the system. Storing the exact signatures as image files would lead to a variety of security vulnerabilities which are mostly associated with the image capture of the handwritten signature. For instance, when images are transferred from one endpoint to another in a network, it may be possible to obtain the image from the intermediate nodes or links by wiretapping. Even if end-to-end encryption is used, when a decryption process is executed on the signature a temporary image is produced on to the hard disk of the system that runs the algorithm. This gives space for an image shadowing attack. Several image recapture software are available today to retrieve temporary images imprinted on the hard disk storage, hence it would be up to the system security of the HSV hardware to keep the image of the signature safe. There are methods to prevent such attacks such as secure shell for end-to-end encryption and encrypted disk storage option during the installation of the operating system that runs the algorithm. But all of this means that the algorithm must heavily rely on the security features of its environment leading to numerous assumptions in its design.

Thus, most HSV systems store the signatures along with their identities in the database as features. Moreover, this method greatly reduces the space required for storing the reference signatures. This may seem to be an appropriate solution, but in some situations it proves to be inappropriate. Consider a classic bank example where a legitimate candidate can have his cheque falsely rejected by a HSV system. If the image of his original signature is available in the database, then it may be possible for the candidate to have his signature verified by a human signature expert in the bank. But when only the features are available, not much can be done to rectify the false rejection.

The solution prescribed in this paper is purely software based and poses minimal effort on database, network and system security. It stores reference signatures as parameters in the form of encrypted curve equations and not as direct features. So, in the case of a false rejection, it would be possible for the authorized verifier to retrieve the original signature of the candidate from the database. To summarize the method, the algorithm segments the given signature into a set of curves and interpolates the signature with the help of Newton's Interpolation formula and stores each curve as a polynomial equation along with its start and end points in the database along with the identity of the signer.

II. RELATED WORKS

Interpolation was already coupled with the concept of security in [1]. In here, Interpolation is utilized for revocation of messages in secret sharing. The paper also proves the efficiency of Newton's interpolation over Lagrange's interpolation. Image interpolation techniques are used to improve visual quality of images. One such technique is proposed in [2]. It shows how edge-directed interpolation (EDI) can be used to improve the clarity of low resolution (LR) images.

Our technique is actually a form of image encryption. An image compression encryption technique saving both space and securing the image was proposed in [3]. The method alters the JPEG compression algorithm and adds an additional encryption code to it. The application of chaotic systems is more predominant in the current trend of image encryption. Reference [4] uses chaotic maps as seeds to generate security keys. It also speculates this function can be recursively applied on the same image to produce different level of encryptions. Chaotic generators can also be implemented with artificial neural networks as expressed in [5]. Despite the emergence of chaos in the study of image encryption, the research in [6] uncovers its flaws by running a cryptanalysis on the Chebyshev chaotic map. In addition to this, Lima et al., have also suggested a technique to overcome those drawbacks. But as expressed earlier, all these methods lead to the temporary reproduction of the image file in memory dumps leading to an image shadowing attack. This type of attack is expressed as a dead forensic attack in [7].

The major drawback of applying image encryption in electronic code book mode is depicted in [8]. When this sort of encryption is used on images with lesser detail, e.g. images with more of the same colour in the same level of grey detail, its outline would be clearly visible in the encrypted image. The article also suggests a preprocessing method to overcome this drawback.

As the importance of the proposed method lies in the handwritten signature verification application, it would be better to know about the basics of it. References [9] and [10] are surveys conducted for a better understanding of handwritten signature verification. This includes the various features of HSV and methods proposed in the field.

III. PROPOSED METHOD

As mentioned before, the algorithm converts the given image of the signature into a set of curve parameters. The algorithm can be split up into five phases as depicted in Fig. 1. Colour downscaling and thickness cancellation can be pictured as part of an image preprocessing step.

The handwritten signature is to be fed as input to the system by means of an image scanner. It passes through a sequence of stages to produce a set of polynomial curve equations along with start and end points. This equation set represents the signature of the signer. These equations should be plotted as a two-dimensional graph to reproduce the image of the signature. But for the purpose of data security, only the node under the control of the handwritten signature expert can reproduce the image as a picture.

This algorithm poses only two security assumptions in contrast to the various ones stated in a nominal HSV system.

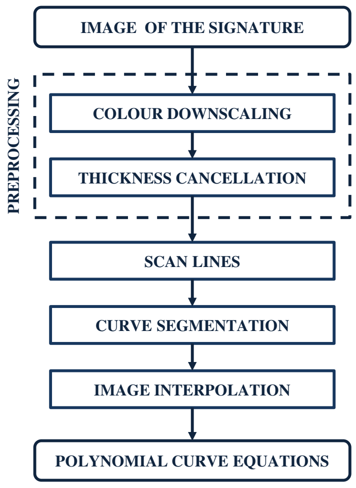

Fig. 1. Proposed approach for the conversion of a handwritten signature image to a set of polynomial curve equations depicted as a flow diagram.

Firstly, the image of the handwritten signature should not be reproduced as a system recognizable image format (such as JPEG, PNG, BMP, etc.) only as a primitive data structure understood by the program running the algorithm, such as a one-dimensional or multi-dimensional array representation. Finally, the system that the handwritten signature expert uses to cross-check false rejections should have encrypted hard disk storage turned on in its operating system. This is because this node is the only one that reproduces the signature as a system recognizable image. This restriction is not necessary with the other nodes that run the algorithm.

IV. IMAGE PREPROCESSING

The first step of the implementation is preprocessing; the image of the signature is first reduced to an image fit for processing for our algorithm.

A. *Colour Downscaling*

The colour of the image is downscaled to pure black and white. It is done by usual means, either by histogram for setting a threshold and then providing the range for decolourization or by any image processing library or API (depending on the programming language adopted) such as the Java Advanced Imaging (JAI) API.

The given image, in the form of RGB, is first converted to a greyscale image. The JAI-API makes short work of this step. Then the colour intensities are analysed pixel wise to obtain an average mark and then from which an appropriate threshold *greythresh* is calculated. This threshold acts as a divider to choose between black and white for each pixel of its given value. In the algorithms discussed in this paper black is considered to take the value 1 and white is to be 0. So if the pixel intensity is lesser than *greythresh*, then pixel is set to be white otherwise black. A pure black and white image is produced as an intermediate output.

B. *Thickness Cancellation*

The thickness cancellation step brings each line within the signature to 1 pixel width. The rationale behind this phase is to make the image to be able to be processed by graph based equations. Such equations cannot handle colour and variable line thickness; hence such details are removed as noise. The algorithm requires the input image to be converted into a pure black and white image. It takes the input image as a two dimensional byte array. Each byte represents a single pixel at the appropriate position with value 1 (for black) or 0 (for white).

Though many papers include this step within their suggested method, few however describe the details involved in the step. We hence show below a simple thickness cancellation algorithm.

1. Get the black and white image as input.
2. Code the image as 2 of 2D arrays, *a* and *b*, with the value 1 for black or 0 for white.
3. Scan the 2D array along the *x*-axis of *a*. For each pixel $a(x, y)$, if $a(x, y) = 1$ and $a(x+1, y) = 1$, then set $a(x, y)$ to 0.

4. Scan the 2D array along the *y*-axis of *b*. For each pixel *b*(*x*, *y*), if *b*(*x*, *y*) = 1 and *b*(*x*, *y*+1) = 1, then set *b*(*x*, *y*) to 0.
5. Set *c* = *a* + *b*, where '+' is a pixel-wise OR operation
6. Produce the black and white image from *c*.

Two temporary copies of the input image are produced. Byte array *a* is read row by row from leftmost pixel to the rightmost pixel. Each time, if the current pixel and the next immediate pixel in the same row is black, the current pixel is cleared to white. By doing this, the width of each line in the image would be reduced all the way to one pixel. A similar process is done in byte array *b*, but it is read column wise top to bottom. Each time, if the current pixel and the next immediate pixel in the same column are black, the current pixel is cleared to white. Thus *b* would contain an image with each line's height reduced to one pixel. A pixel-wise logical OR operation between both byte arrays *a* and *b* would return the value for *c*, the final pure black and white image with thickness of each line cancelled to 1.

V. PRIMARY SEGMENTATION

A. *Scan Lines*

This algorithm poses only two security assumptions in contrast to the various In order to extract the curve equations from the given handwritten signature image, it must be separated to a set of lines for which it must be "scanned" from the raster image. The entire algorithm pictures the image as the first quadrant of a two dimensional graph. The scan lines phase of the algorithm is as shown below. This algorithm converts the pre-processed signature into a set of lines, each represented by an ordered sequence of points (*x*, *y*). From the following description onwards, we may use the term pixel and point interchangeably. A detailed description of this method is given in Algorithm 1.

The algorithm reads the image point by point. Each time a black cell is encountered (value = 1) a new line is started with this cell as its first point. The algorithm checks which neighbour has the next point along the current line. It recursively traces the neighbours of the cell to form a line. This part of the algorithm is known as the LineTrace procedure (Algorithm 2). When the end of the current line is reached, the current line is added to the list of lines, *lineList* that make up the image and the algorithm carries on the reading the image from where it left off. The control is certain to reach points that had already become part of previous lines. To avoid retracing the same line, the set of visited cells are kept in track by the variable *visited*. If there are more than one cell to trace along the current point, then the algorithm is said to have reached a

Algorithm 1

```
1:  function SCANLINES($bwImage$) : List⟨List⟨Point⟩⟩
2:      List⟨Point⟩ $visited$;                ▷ Set of visited points
3:      List⟨List⟨Point⟩⟩ $lineList$;              ▷ List of lines
4:      for each Point $p$ in $bwImage$ do
5:          if $(p = 1)$ AND $(p \notin visited)$ then
6:              LineTrace($p, visited, lineList$);
7:          end if
8:      end for
9:      return lineList;
10: end function
```

Algorithm 2

```
1:  procedure LINETRACE($p, visited, lineList$)
2:      Point $prev$;                          ▷ Last visited point
3:      List⟨Point⟩ $cLine$;       ▷ Current line being produced
4:      Stack⟨Point⟩ $bStack$;       ▷ Stack of branching points
5:      add $p$ to $cLine$;
6:      add $p$ to $visited$ set;
7:      $bStack \leftarrow$ branchSearch($p$, null , $bStack$);
8:      $prev \leftarrow p$;
9:      while !$bStack$.isEmpty() do
10:         $p \leftarrow bStack$.pop();
11:         if $p \notin visited$ then
12:             add $p$ to $visited$;
13:             if $p$ is not adjacent to $prev$ then
14:                 add $cLine$ to $lineList$;
15:                 initiate new $cLine$;
16:             end if
17:             add $p$ to $cLine$;
18:             $bStack \leftarrow$ branchSearch($p$, $prev$, $bStack$);
19:             $prev \leftarrow p$;
20:         end if
21:     end while
22:     if !$cLine$.isEmpty() then
23:         add $cLine$ to $lineList$;
24:     end if
25: end procedure
```

branching point and must choose to carry on one of the branches and then come back to the other branch (or branches) after completing the current line and so on. To keep track of the pending branch points, a stack is used. So whenever the current line ends, the branch stack is popped to provide the first point for a new line. Once the branch stack is empty, the algorithm reverts back to the ScanLines algorithm. The ScanLines algorithm repeats the LineTrace procedure for each unvisited cell until the entire signature image is covered.

B. *Search Optimization – Smart Search*

We pose a special method called smart search to search for neighbouring pixels. A typical exhaustive search for branching lines would compare all 8 neighbouring pixels for each pixel. A smart search would reduce this to 5 comparisons per pixel. Whenever the control moves from one pixel to the next, the previous pixel is kept into account. Initially, the previous pixel is null; the program must refer all 8 neighbouring pixels for a branch. When it moves from one pixel to another, it avoids searching the pixels it would've already compared.

Fig. 2. describes smart search by indicating the search control for each point from where the previous point to the current point and where to search for the next point. This applies both to locate the first point of a line as well as locating branches. For example, if the previous point is to the immediate left of the current point, then the neighbouring points to search for the next black pixel is to the top, top-right, bottom-right and the bottom of the current point (as in Fig. 2a). This means that it is already understood that the top-left, left and bottom-left have already been traversed before in order for the previous point to reach the current point.

The illustration in Fig. 3 would aid to provide a better understanding of the smart search (Fig. 3b) and its comparison with the exhaustive search (Fig. 3a). The illustration shows how many times the search algorithm compares the respective cells to detect the next point continuation of the line with the assumption that the search starts with the leftmost point on the line.

The number on the cell shows how many times that cell is traversed by the algorithm. An exhaustive search algorithm would always checks all neighbouring cells to seek out the next point on the line. The smart search searches technically avoiding the most of the cells that already may have been traversed according the rules as shown in Fig. 3. The exhaustive search takes up to a total of 122 cell checks to reach the end of the line. But, the suggested smart search takes only a total of 83 cell checks. This is in fact a major improvement when considering the running the algorithm for images of large sizes.

C. *Curve Segmentation*

Since we are dealing with statistical curve equations, there should not be more than one outcome for a given input. In our case, we consider x as input and y as outcome. Hence there should not be more than one value of y for a given value of x. For instance, in Fig. 4a, at point x_i there are two outcomes y_{i1} and y_{i2}. Once they are split as shown in Fig. 4b and 4c, there would be only one y_i for any x_i within that line.

The output of ScanLines algorithm should be used as input for this step (Algorithm 3).

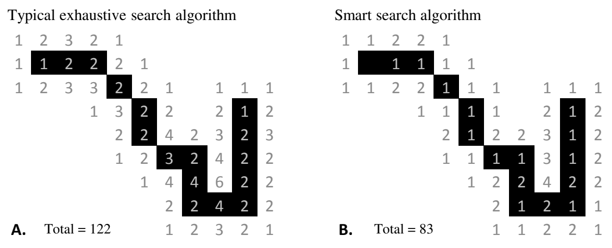

Fig. 3. Smart Search and Exhaustive Search Comparison

Hence it is assumed that the set of points within a line is ordered in terms of its connectivity. The reason why this step is called ‘primary’ segmentation is because the next step of curve generation (image interpolation) would require the lines to be further subdivided according to the needs of the respective formula used for the curve production. Curve

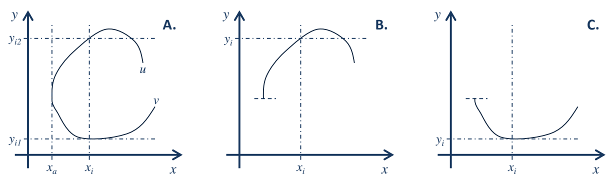

Fig. 4. Primary Segmentation of a Curve

segmentation makes sure that these lines can be processed by image interpolation.

The algorithm is very simple. The points of each line are read in order. The direction of change along x-axis (increase/decrease) is monitored for every successive point taken with respect to the previous point. If there is a change,

then the set of points read so far in the list must be appended to ta new line up to the point that resulted the change in direction. Then instantiate a new line and continue the algorithm till the end of the current line. Follow these steps for each line in the set of lines. The final output is a set of lines represented by ordered points such that there would be no more than one value of y for a given value of x for each line.

Algorithm 3

```
1: function CURVESEGMENTATION($prevLineList$) : List⟨List⟨Point⟩⟩
2:     List⟨List⟨Point⟩⟩ $lineList$;
3:     List⟨Point⟩ $cLine$;
4:     for each $line$ in $prevLineList$ do
5:         for each point $p$ in $line$ in order do
6:             monitor the direction of change along $x$-axis;
7:             if there is a change in the direction of the observed point then
8:                 add $cLine$ to $lineList$;
9:                 initiate a new $cLine$;
10:            end if
11:            add $p$ to $cLine$;
12:        end for
13:        add $cLine$ to $lineList$;
14:        initiate a new $cLine$;
15:    end for
16:    return $lineList$;
17: end function
```

VI. IMAGE INTERPOLATION

A. *Newton’s Interpolation*

Interpolation was first introduced to estimate unrecorded information using the existing ones in statistics. The method of interpolation used in this project is Newton’s Interpolation through Divided Differences [11]. In order to interpolate a given set of points (x, y), where $y = f(x)$, it is necessary to first define the divided difference of $f(x)$, which is given by (1).

$$\Delta^i f(x_{n-1}) = \begin{cases} f(x_n) & \text{if } i = 0, \\ \dfrac{\Delta^{i-1} f(x_n) - \Delta^{i-1} f(x_{n-1})}{x_{n-1+i} - x_{n-1}} & \text{if } i > 0 \end{cases} \quad (1)$$

where *n* is the number of points to be interpolated to a single curve. From this equation, the Newton's Divided Difference Formula can be expressed as in (2).

$$y = y_0 + \sum_{i=0}^{n-1}\left[\Delta^{i+1} f(x_0) \prod_{j=0}^{i}(x - x_j)\right] \quad (2)$$

B. *Image Interpolation using Newton's Formula*

Using equation (2), the given image of the reference signature can be converted to a set of polynomial curve equations. The polynomial equation can be represented as *y* = *f*(*x*) in which *f* (*x*) is expressed by a sum of *x* raised to different integral powers each having a real coefficient. The polynomial can be depicted as a list (or array) with each entry considered as a coefficient and the positional index of the list as the respective power of *x*. A curve under this context consist of a tuple of start and end points along with the polynomial equation. Lines (ordered set of points) are taken one by one in the given set. Algorithm 4. shows how the set of segmented lines can be converted to polynomials using Interpolation.

Algorithm 4

```
1: function IMAGEINTERPOLATION($lineList$) : List⟨Curve⟩
2:     List⟨Point⟩ $cLine$;
3:     List⟨Double⟩ $eqn$;
4:     List⟨Curve⟩ $curveList$;
5:     for each $line$ in $lineList$ do
6:         while end of $line$ is not reached do
7:             read up to $N$-points from the $line$ to $cLine$;
8:             $eqn \leftarrow$ interpolate($cLine$);
9:             repeat
10:                read next point $p$ from $line$;
11:                if $p$ lies in $eqn$ then
12:                    add $p$ to $cLine$;
13:                end if
14:            until $p$ does not lie in $eqn$
15:            add to $curveList$ ⟨start of $cLine$, end of $cLine$, $eqn$⟩;
16:        end while
17:    end for
18:    return $curveList$;
19: end function
```

Up to *n*-points from the line is first read to form the polynomial equation using Newton's Interpolation. Then read the rest of the line point-by-point checking whether those points lie on the formed equation until the line ends or until the point next in the line does not lie on the polynomial equation. The start point is the first of the *n*-points that was taken to form the initial curve and the end point is the last point that followed the formed equation (not the one that deviated from it). The tuple <Start pt., End pt., Equation> added to the curve list. If the line end is not reached, then again read up to *n*-points starting to read from the point that did not follow the current equation continuing the algorithm. If the line end is reached, then the next line in the segmented line list is taken.

Thus each line provided in the set of lines may correspond to one or more polynomial equations each specified with start and end points. The *n* value determine the maximum degree of the polynomial equation. The value of *n*-points was experimentally calculated to be seven due to a trade-off between time and space efficiency. The result of this experiment is included in the following section.

VII. RESULTS AND DISCUSSION

Thus each line provided in the set of lines may correspond to one or more polynomial equations each specified with start and end points. Fig. 5a shows a sample original signature given as input to the system. When this image is converted to a set of polynomial curves and then reverted back to an image by drawing the curves, it appears as shown in Fig. 5b. The signature using this method is composed of 176 curves when the value of *n*-points is taken to be seven.

Java ArrayList object of the tuple <Polynomial, Start pt., End pt.> was serialized into a file. The file size of this structure was 11 KB while its equivalent PNG file was 12 KB (Fig. 5a).

The size of the output data structure and the runtime of the algorithm depend on the value of *n*-points, i.e. the number of points taken for interpolation at a time.

Newton's divided differences provide a recursive approach for interpolation; the greater the number of points to be interpolated at a time, the more the formula recurs. The run time of the algorithm increases exponentially with the increase in the number of points taken at a time. On the other hand taking only few points at a time for interpolation increases the number of lines needed to represent a single signature and hence lesser space efficiency on the whole scale. To equalize the trade-off between the runtime and space efficiency, the number of points to be interpolated at a time (*n*-points) was experimentally calculated. The space efficiency can be monitored by observing the increase in the number of lines with respect to the increase in *n*-points. Observing the runtime efficiency of the algorithm is however more complex. This requires monitoring the computation of lebla operations along with the summing and product operations. To simplify this task, we accounted only the number of lebla operations taken for a given value of *n*-points. For each value of *n*, the total number of lebla operations and the number of curves required to represent the image was observed. The experiment was carried over for the same image as shown in Fig. 5a. An XY

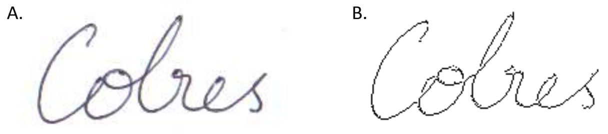

Fig. 5. Sample input original signature (a) and output retrieved signature (b)

graph was plotted with two curves as shown in Fig. 6.

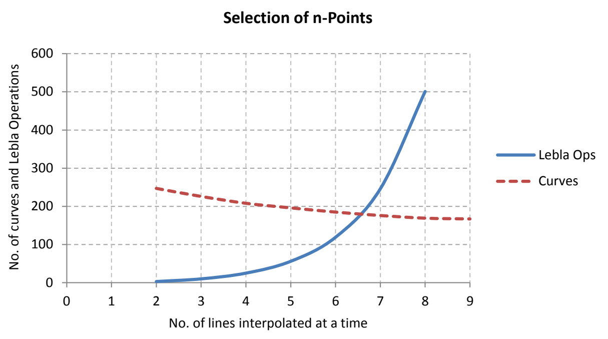

Fig. 6. Smart Search and Exhaustive Search Comparison

In the aspect of this article, both space and time efficiency is accounted for equal importance, so no scaling options for the curves were given. The upper bound of the point of coincidence should be the most efficient value for *n*-points. From Fig. 6, this value was observed to be seven equalizing the trade-off between space and time efficiency. This value was found to confirm with most other signatures as well.

The proposed method is found to have major advantages. No intermediate image is to be produced during the retrieval of the signature. This means that the image can be directly produced in the form of a two dimensional array of bytes representing a first quadrant graph producing no recognizable standard image format on the disk of the working environment which would otherwise make room for a dead forensic attack. End-to-end encryption can be done for added security of the set of polynomials. The following are some of the notable features are preserved during the transformation: spatial difference; absolute coordinates; inclination; curvature; signature path length; path tangent angles; aspect ratio; cut vertices. This forms a superset of most global and local features involved in HSV.

There are some minor drawbacks of this method. The algorithm provides insignificant compression as JPEG compression provides more favourable space efficiency (7 KB for Fig. 6a). Although the method preserves many fundamental features of a handwritten signature, it comes with a cost of eliminating the features grey strokes, gradient and line thickness which some HSV systems find important.

VIII. CONCLUSION

Through this paper, we have proposed a method to secure the image of reference signatures used for handwritten signature verification. The implementation of the method has been illustrated with suitable algorithmic structures. The results were discussed analysing both its merits and demerits. Apart from the proposed method, some may feel that it would be better to use end to end encryption along with hard disk storage encryption facility. Encrypting data for storage each time for every intermediate output would render the entire HSV system with less than its overall performance potential. We believe that our approach brings about a simpler yet effective solution to the problem of handwritten signature confidentiality.

We hope to find out more curve representation techniques and compare with the one at hand and also possibly overcome the demerits; providing better compression by incorporating a suitable compression algorithm as part of our proposed method and a way to retain gradient details of the original signature.

REFERENCES

[1] N. Kogan and T. Tassa, "Improved efficiency for revocation schemes via newton interpolation," ACM Transactions on Information and System Security, vol. 9, pp. 461–486, 2006.

[2] S. Yu, R. Li, R. Zhang, M. An, S. Wu, and Y. Xie, "Performance evaluation of edge-directed interpolation methods for noise-free images," in Proceedings of the 5th International Conference on Internet Multimedia Computing and Service, Huangshan, China, 2013, pp. 268–272.

[3] L. Zhiqianga, S. Xiaoxin, D. Changbin, and D. Qun, "JPEG algorithm analysis and application in image compression encryption of digital chaos," in Proceedings of the 2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control. Washington, DC, USA: IEEE Computer Society, 2013, pp. 185–189.

[4] Y. Zhou, L. Bao, and C. Chen, "A new 1D chaotic system for image encryption," Signal Processing, Elsevier, vol. 97, pp. 172–182, April 2014.

[5] A. Kassem, H. Al Haj, Hussein, Y. Harkouss, and R. Assaf, "Efficient neural chaotic generator for image encryption," Digital Signal Processing, vol. 25, pp. 266–274, February 2014.

[6] J. Lima and L. Novaes, "Image encryption based on the fractional fourier transform over finite fields," Signal Processing, Elsevier, vol. 94, pp. 521–530, January 2014.

[7] D. S.M., M. C.R., L. D.M., and W. A, "When cryptography meets storage," in Proceedings of the 4th ACM international workshop on Storage security and survivability, ACM, 2008, pp. 11–20.

[8] I. Elashry, O. Faragallah, A. Abbas, S. El-Rabaie, and F. Abd El-Samie, "A new method for encrypting images with few details using rijndael and RC6 block ciphers in the electronic code book mode," Information Security Journal: A Global Perspective, Taylor and Francis, vol. 21, no. 4, pp. 193–205, 2012.

[9] S. Pal, M. Blumenstein, and U. Pal, "Off-line signature verification systems: a survey," in Proceedings of the International Conference and Workshop on Emerging Trends in Technology, ACM, February 2011, pp. 652–657.

[10] S. Garhawal and N. Shukla, "A study on handwritten signature verification approaches," International Journal of Advanced Research in Computer Engineering and Technology, vol. 2, no. 8, pp. 2497–2503, August 2013.

[11] P. Kandasamy, K. Thilagavathi, and K. Gunavathi, Numerical Methods, 3rd ed. India: S. Chand and Company Ltd., 2005, ch. Interpolation with unequal intervals, pp. 257–270.