World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:14, No:8, 2020

Authentication of Physical Objects with Dot-Based 2D Code

Michał Glet, Kamil Kaczyński

***Abstract*—**Counterfeit goods and documents are a global problem, which needs more and more sophisticated methods of resolving it. Existing techniques using watermarking or embedding symbols on objects are not suitable for all use cases. To address those special needs, we created complete system allowing authentication of paper documents and physical objects with flat surface. Objects are marked using orientation independent and resistant to camera noise 2D graphic codes, named DotAuth. Based on the identifier stored in 2D code, the system is able to perform basic authentication and allows to conduct more sophisticated analysis methods, e.g., relying on augmented reality and physical properties of the object. In this paper, we present the complete architecture, algorithms and applications of the proposed system. Results of the features comparison of the proposed solution and other products are presented as well, pointing to the existence of many advantages that increase usability and efficiency in the means of protecting physical objects.

***Keywords*—**Authentication, paper documents, security, anti-forgery.

I. Introduction

PAPER documents still play an important role in communications between governmental agencies, citizens and businesses. Many organizations continue using paper documents for invoicing, creating and signing contracts, as well as for communicating with their business partners and clients. It is easy to imagine a situation in which someone creates a falsified document to discredit an organization, e.g. by providing false information to its customers. Nowadays, falsified documents are easier to create than ever before – a PC with a printer and a text editing software is all it takes. Anyone is capable of creating a document pretending to be a company headletter, with any text published thereon. Such a falsified document may be sent to the company's customers who, being convinced of its genuine character, may draw false conclusions about the alleged sender.

In [1], the authors conducted a study which resulted in determining three different approaches to falsifying documents in order to fraudulently receive money from an insurance company: The first one - Print, Paste and Copy (PPC) – consisting in printing a new text on an empty sheet, and then pasting it onto a part of the genuine document. The next step was to copy the document using a color copy machine. The second approach was known as Reverse Engineered Imitation (REI) forgeries. In this approach, the forger creates a new, editable document based on the genuine copy. In the case of that study, people were imitating genuine invoices, retyping all the text and placing the logos, dates, etc. at the proper locations. The third approach was named Scan, Edit and Print (SEP). The forger scanned the original document and then manipulated the digital image thereof.

There are many works dealing with counterfeiting digital and paper documents. In [2], the authors are focusing on documents which were digitalized, and the digital copies are protected by using a 1D hash algorithm and 2D iFFT encrypting documents in the 2D spatial domain. In [3], one may find a method for identification of the source printer that was used for creating the forged document. The authors claim that accuracy of 76.75% may be achieved. The authors of [4] propose a text-line examination method which may be relied upon in high-volume environments. The method requires that each analyzed document be digitized, and that feature points be collected from the binarized images.

The methods mentioned above are truly effective, but they cannot be used for authenticating documents by parties without special equipment. Our system – DotAuth - is a solution dedicated for individuals who are also widely affected by forged documents. The proposed system does not require any special hardware – all verification steps are performed with the user's smartphone. A dedicated DotAuth app analyzes document contents using the smartphone camera and computer vision algorithms. Such an approach makes it easy to deploy the solution on the mass scale. The system may be used for authentication of paper documents or items with a flat surface – e.g. product packaging. The DotAuth authentication symbol comprises several points located along the line of a circle. The document does not have to be placed in one correct position for the purpose of authentication. The symbol will be calculated correctly regardless of the orientation of the image. The document is divided into several areas with different authentication circles. The user has to scan the entire document or only its selected areas, providing enough data to verify the authenticity of the document. Unlike classic methods, such as watermarks, our solution is much harder to copy into the falsified document, simultaneously being much easier to read with the use of computer vision mechanisms.

A somewhat similar approach may be found in [5]. The authors describe a visual information concealment technique which may be deployed for document authentication purposes. The main disadvantage of this method, compared to the one proposed in this paper, consists in the lack of a fast and easy mechanism for reading the authentication-related information. This renders the method in question unsuitable for commercial applications in which no sophisticated devices performing the

M. Glet and K. Kaczyński are with Military University of Technology, Faculty of Cybernetics, Institute of Mathematics of Cryptology (e-mail: michal.glet@wat.edu.pl, kamil.kaczynski@wat.edu.pl)

authentication process are available.

II. DotAuth Principles and Algorithm

There are two types of methods used for authentication of physical products such as documents, goods, drugs, etc. The first method is based on intrinsic properties that are directly related to the product subjected to authentication. In the case of paper documents, the intrinsic properties include, for instance, paper gloss, brightness, opacity and color. The other category of authentication methods relies on extrinsic properties which are added to the product during the marking process. In the case of paper documents, those properties include, for instance, watermarks, printed symbols, chemical marks, etc. An ideal authentication system should be capable of checking both intrinsic and extrinsic properties of a given product. Of course, such a system would be totally impractical, so in a real-life scenario a tradeoff between the degree of security and usability exists. Therefore, when developing a system for widespread use, the following features need to be taken into consideration:

- Ability to use existing production techniques;
- Ability to perform a machine analysis;
- Low cost of integration with the product;
- Analysis performed with the use of widely available hardware;
- Ability to perform the analysis by a person without expert knowledge;
- Unique identifiers for each product;
- Inseparably integrated with the product;
- Cryptographically secure.

Authentication of digital documents or other digital data can be performed using digital signature schemes. Thus, data may be authenticated in the most convenient way without any need of manual steps being taken by the validating party. All authentication-related steps may be performed by the end-user application, e.g. for authenticating PDF files, the user may rely on the built-in Adobe Reader functionality. Unfortunately, the same does not apply to traditional documents. In [6], the authors propose a scheme which uses QR Codes and the digital signature scheme for authenticating paper-based documents. The main disadvantage of that method is its sensitivity to OCR errors, resulting in false negatives obtained during the verification process. Another approach to solve the problem was presented by a company called Jumio [7]. The document verification service performed the analysis by capturing an image of the document in question and by extracting some key identity-related information therefrom. The service needs to store all key information in its database, which may lead to unintended disclosure of such information. The solution is also sensitive to OCR errors, so it may provide false negative results during the verification process.

DotAuth combines different approaches to document verification. Firstly, it uses a proprietary 2D code technology, printing a code on the marked document. Secondly, it allows the user to manually verify if the paper document is equal to its digital equivalent. Lastly, using augmented reality, the DotAuth application may highlight the modified elements of the document or may mark it as genuine. The DotAuth application is communicating with service servers, using data read from the 2D code as part of the communication requests. Details concerning the communication process are described in the following section.

A. DotAuth Code Creation

The DotAuth 2D code (Fig. 1) is composed of points (dots) placed along the perimeters of circles sharing the same center and having different radiuses. The code consists of *n* circles, divided into *k* sectors. Thus, up to *n* points may be placed in each sector. Each point will denote one bit – if a point is present, the bit value equals 1. If no point is present, the bit value is 0. The innermost circle is reserved for synchronizing the code position, ensuring that the data is read in the proper order. The overall capacity of the DotAuth *(n,k)* code is equal to *(n-1)k* bits. The point localized on the innermost circle and the center of that is used for synchronizing the code position – the point on the innermost circle defines the "up" direction of the code.

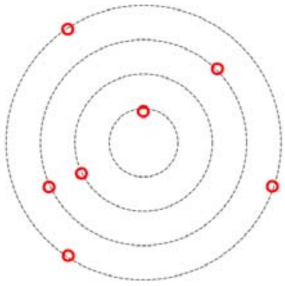


Fig. 1 DotAuth 2D code – dots placed on circles

The process of creating a new DotAuth *(n,k)* code requires that the following steps be performed:

1. Generate *n* circles with a common center with Cartesian coordinates *O=(x,y)*. The radius of the innermost circle is equal to *r;* the radius of the second circle is equal to *2r,* and so on.
2. Divide the circles into *k* sectors.
3. Create sector iterator *i=0.*
4. Place a synchronization dot on the top of innermost circle – at the point with coordinates *(x,y+r).*
5. Take subsequent *(n-1)* bits of the identifier and store it in the vector $v = [v_1, \dots, v_{n-1}]$.
6. Starting from the outermost circle, take v_1 and if it is equal to 1, place a dot at the point with coordinates $\left(x + n \cdot r \cdot \sin\frac{2i\pi}{k}, y + n \cdot r \cdot \cos\frac{2i\pi}{k}\right)$, if v_1 is equal 0, go to step 7.
7. Take the next element from vector v – element v_m, where m is initially equal to 1. If v_m is equal to 1, place a dot at the point with coordinates $\left(x + (n-m) \cdot r \cdot \sin\frac{2i\pi}{k}, y + (n-m) \cdot r \cdot \cos\frac{2i\pi}{k}\right)$, if v_m is equal to 0 do nothing. Increment value m and repeat step 7, until m=*n-1*.
8. Increment sector iterator *i*. If *i*<*k* go to the step 5. Otherwise, the code creation process is completed.

The process of code creation may also be described as moving from the center of the circle towards its perimeter and placing points on the circles sharing the same center and having different radiuses. After each step, there is change in angle that equals $\frac{2\pi}{k}$. The capacity of the code may be easily tailored to the needs of each implementation. For example, if we need to store 48 bits of data, then we need to create a DotAuth code with 4 circles divided into 16 sectors. During the creation process, it is necessary to locate at least three points on the outermost circle. It is not possible to decode a DotAuth code with zero, one or two points on the outermost circle.

B. Reading DotAuth Code

The DotAuth code may be read using optical techniques, e.g. by analyzing the image captured with a smartphone camera. The main problem for that type of analysis is to ensure that the elements of the code are read in the proper way. For codes like QR Code [8], there are several types of patterns – alignment pattern, finding pattern and timing pattern enabling the code data to be read without a synchronization error. The DotAuth code may be located e.g. under the body of the document, meaning that the classic techniques are not effective here. The synchronization process is based on two points – the first one is located on the innermost circle of the DotAuth code, while the other one – not visible – is the center of the DotAuth circle. With these two distinct points identified, it is possible to recreate the half-line starting at the center of the circle and passing through the point on the innermost circle. The half-line determines the direction based on which the starting point of the DotAuth code is identified.

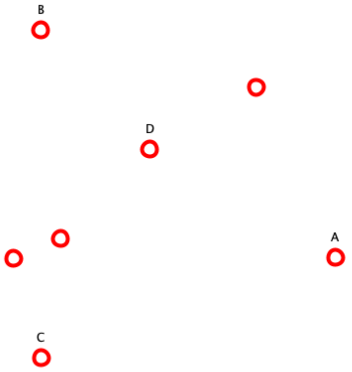


Fig. 2 DotAuth code with distinguished points (A, B, C, D)

The process of decoding the DotAuth code requires specific knowledge about the equation identifying the outermost circle, the number of circles n and the number of sectors k. n and k are implementation-dependent, meaning that they will be constant and may hardcoded, for instance, into the decoding app. In order to recreate the equation of the outermost circle, it is necessary to have at least 3 points that are positioned on this circle. Fig. 2 shows the DotAuth code with three points – $A=(x_A,y_{A,})$, $B=(x_B,y_B)$ and $C=(x_C,y_C)$ located on the outermost circle. Such may be easily recovered with the use of image processing libraries, so we can use their coordinates and substitute them into the circle equation, obtaining the following system of equations.

$$\begin{cases}(x-x_A)^2+(y-y_A)^2=R^2\\(x-x_B)^2+(y-y_B)^2=R^2\\(x-x_C)^2+(y-y_C)^2=R^2\end{cases} \tag{1}$$

After solving (1) it is possible to recover the location of the synchronizing point. Using the image processing technique, we get the coordinates of point $D=(x_D,y_D)$, with the coordinates of the circle center $O=(x,y)$ known as well. Given those coordinates, we can calculate the angle between vector $\overrightarrow{OD}$ and vector $\vec{j}=(0,1)$. We denote that angle as α. Now, having all necessary data, it is possible to recover data from the DotAuth code by performing the following steps:

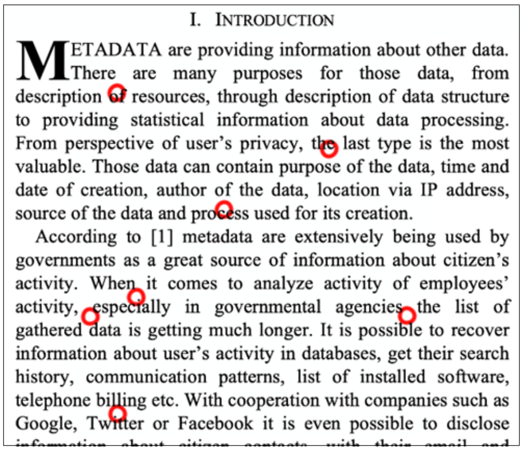
1. Generate n circles with a common center with Cartesian coordinates $O=(x,y)$. The radius of the innermost circle is equal to $r=\frac{R}{n}$; the radius of the second circle is equal to $2r$, and so on.
2. Initiate vector v with $(n\text{-}1)k$ elements.
3. Initiate sector iterator $i=0$.
4. Initiate $m=0$.
5. Starting from the outermost circle, check if there is a dot near the point with coordinates $\left(x+(n-m)\cdot r\cdot\sin(\frac{2i\pi}{k}+\alpha), y+(n-m)\cdot r\cdot\cos(\frac{2i\pi}{k}+\alpha)\right)$, if it exists add element *1* to vector v, otherwise add element *0* to vector v. Increment m.
6. If $m<n\text{-}1$, go to step 5.
7. Increment sector iterator i. If $i<k$, go to step 4. Otherwise, the reading process is completed.

Decoded vector v is the identifier stored in the DotAuth code. When the term “dot near the point” was used, it meant the neighborhood of that point with radius of $\frac{r}{4}$ within that point. No error correction step is performed in DotAuth, but it may be added at the expense of reducing the capacity of the code.

C. Main Properties of DotAuth System

The DotAuth system relies on the DotAuth code and cryptography to deliver a trustworthy solution for authenticating paper documents and other physical objects. The DotAuth code is used as an identifier, and it may also be replaced with any other machine-readable code for special purpose applications. The main advantage of the DotAuth code is that it may be placed under the text, meaning that it may be easily adopted to existing document templates and may even be printed on a document after it has been created.

Fig. 3 shows the DotAuth code placed under a specific text. Using the dedicated app and a compatible smartphone, one may read the identifier and send it to the DotAuth server. The server replies with basic metadata about the identified product and – if a document is being validated – returns a URL allowing to download the encrypted file for further analysis. After decrypting, the file may be analyzed manually or with the use of augmented reality methods. It may be marked as genuine or the modified elements may be highlighted. All documents are stored in an encrypted form on the DotAuth servers. All steps of the analysis process are performed on the user device, preventing sensitive data from leaking. Encryption/decryption keys are generated using an identifier and secret data read from the DotAuth code, as well as using other implementation specific data. The algorithm used in the encryption process is AES-256.



I. INTRODUCTION

METADATA are providing information about other data. There are many purposes for those data, from description of resources, through description of data structure to providing statistical information about data processing. From perspective of user's privacy, the last type is the most valuable. Those data can contain purpose of the data, time and date of creation, author of the data, location via IP address, source of the data and process used for its creation.

According to [1] metadata are extensively being used by governments as a great source of information about citizen's activity. When it comes to analyze activity of employees' activity, especially in governmental agencies, the list of gathered data is getting much longer. It is possible to recover information about user's activity in databases, get their search history, communication patterns, list of installed software, telephone billing etc. With cooperation with companies such as Google, Twitter or Facebook it is even possible to disclose information about citizen contacts with their email and

Fig. 3 DotAuth code under the text

Manual verification of document contents should be available whenever no augmented reality tools are available. It is also possible to come up with a specific verification sequence – first, the user performs manual verification of the document content, then it is possible to perform automatic verification with the AR method. The two-step approach may be useful for long documents, such as contracts.

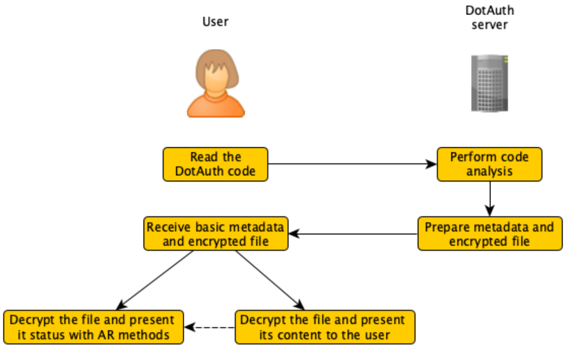

User
DotAuth server
Read the DotAuth code
Perform code analysis
Receive basic metadata and encrypted file
Prepare metadata and encrypted file
Decrypt the file and present it status with AR methods
Decrypt the file and present its content to the user

Fig. 4 Process of analyzing data in the DotAuth system

III. PROPOSED ARCHITECTURE OF THE SYSTEM

The proposed architecture of the DotAuth authentication system comprises dedicated subsystems (Fig. 5) that are intended for performing various, nearly independent data and request processing steps. This kind of architecture poses some difficulties related to developing and managing the system, but is very flexible and suitable for very easy horizontal and vertical scaling in the event of high system load and usage ratios.

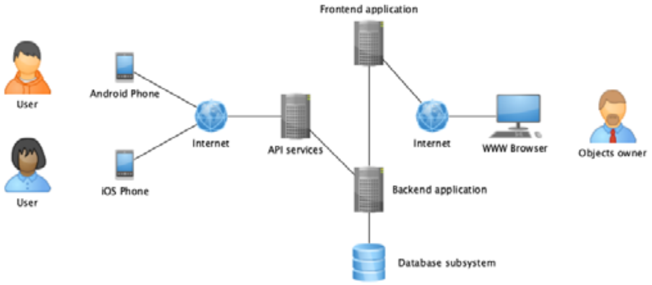


Fig. 5 Overview of the DotAuth authentication system architecture

The DotAuth authentication system is built based on the following subsystems:

1. Database subsystem.
2. Backend application.
3. API services.
4. Frontend application.
5. Mobile applications

A. Database Subsystem

This part of the DotAuth authentication system is responsible for storing system-relevant data. Depending on the utilization of the system and its resources, the database subsystem may rely on SQL database management systems, NoSQL database applications and file storage services. All sensitive data stored in the database subsystem will be maintained in an encrypted form. Business communication with the database subsystem will be fully authenticated and may only be initiated by the DotAuth backend application. All communications with the database subsystem will be encrypted, ensuring proper levels of data privacy and integrity.

B. Backend Application

The backend application is a major subsystem in terms of business processing. This part of the DotAuth authentication system is responsible for the processing of business data and business requests. The backend application receives requests from the frontend application and API services, passing those requests from end users and objects owners. Furthermore, the backend application is responsible for authentication and authorization of requests – it checks whether the rights to access the requested data and to process the resources have been granted. All communications with the backend application will be encrypted and will guarantee privacy and integrity of data.

C. API Services

This part of the DotAuth authentication system exposes, to the Internet, the API services that are intended to be used by DotAuth mobile applications. All communications between the mobile applications and the server side of the system (backend application, database subsystem) pass through the API services. API services expose the business methods that are relied upon directly by the mobile applications to perform server-side business operations – e.g. to retrieve information about the object that is being authenticated. API services will be protected against unauthorized access. All communications with API services will be encrypted and will provide proper levels of data privacy and integrity.

D. Frontend Application

This part of the DotAuth authentication system is intended to be used by object owners (business end users who intend to mark objects, e.g. documents). The frontend application provides a web-based graphical user interface that may be used to perform business actions in the DotAuth system. The main business action allows to upload a document (object), marks it with the authentication data and downloads the new version. Additionally, the frontend application performs activities allowing, inter alia, to manage the objects (documents), objects (documents) marked with the DotAuth code or company users. All communications between the end user and the frontend application will be encrypted and will offer proper levels of data privacy and integrity.

E. Mobile Applications

This part of the DotAuth authentication system is intended to be used by users (business end users who intend to authenticate objects, e.g. documents). The DotAuth system will offer mobile applications for two major mobile operating systems – Google Android and Apple iOS. The main business action supported by the mobile applications consists in authenticating the objects (documents). This action is triggered by the user who has scanned the DotAuth code from a specific object. Then, the mobile application performs AI/AR image processing and performs computations described in the previous sections in order to read the identifier stored in the DotAuth code. Then, via API services, the mobile application communicates with the DotAuth backend application, and the data returned thereby is relied upon to decide whether the object in question may or may not be authenticated. All communications between the mobile applications and API services will be encrypted and will ensure proper levels of data privacy and integrity.

IV. High Capacity DotAuth code

The DotAuth code described in the previous sections has a relatively low capacity – low amounts of data may be stored therein. This amount is suitable for the presented DotAuth authentication system but may not be suitable for some other applications. In such cases, a high capacity version of the DotAuth code may be used.

A. Basic Idea

The idea of a high capacity DotAuth code is very simple and is based on the same DotAuth code principle. In the standard (low capacity) version of the DotAuth(*n*,*k*) code, there are *n* circles and each of them is divided into *k* sectors. Each sector is used to store data. Only one bit of data may be stored in a single sector of a single circle. This process is based on the dot notation presented in previous sections. Each sector of each circle has the same radian length. But the length of sectors (arc length) of circles with a bigger radius is greater than in circles with smaller radiuses. Sector arc length depends on circle radius and on the value of parameter k of the DotAuth code. It may be computed with the use of (R is the radius of the circle):

$$sector\_arc\_length(R,k) = \frac{2\cdot\pi\cdot R}{k} \tag{2}$$

The high capacity DotAuth code uses this length to determine how much data may be stored in a single sector of a given circle. This is the main difference between the normal DotAuth code and its high capacity version. In the high capacity version, one sector may store more than one bit of data only. Sector capacity depends on the length of the sector arc and on the diameter of the dot that is used to store data. That is why the high capacity DotAuth code is associated with three parameters:

1. n – number of circles
2. k – number of sectors in every circle
3. d – dot diameter

The high capacity DotAuth code will be marked as hcDotAuth(*n*,*k*,*d*). The overall capacity of the hcDotAuth(*n*,*k*,*d*) code may be computed based on (where *r* is the radius of the innermost circle):

$$hcDotAuth_{capacity(n,k,d)} = \sum_{j=2}^{n} \left\lfloor \frac{2\cdot\pi\cdot j\cdot r}{d} \right\rfloor \tag{3}$$

The overall capacity of the code depends largely on the diameter of the dot. Codes with small dots it will offer much more capacity than those with big dots. Table I shows the capacity of the hcDotAuth code for constant *n* and *k* values and for different *d* parameter values.

TABLE I
Capacity of the hcDotAuth Code vs. *d* Parameter Value

Circle number	d/r ratio (dot diameter/radius of the innermost circle)				
	1	0,8	0,6	0,4	0,2
2	12	15	20	31	62
3	18	23	31	47	94
4	25	32	41	62	125
5	31	39	52	78	157
6	37	47	62	94	188
7	43	54	73	109	219
8	50	62	83	125	251
9	56	70	94	141	282
10	62	78	104	157	314
Total capacity:	334	419	560	844	1692

B. Code Creation Principle

The process of creating hcDotAuth(*n*,*k*,*d*) code is very similar to that described in Section II *A* and requires that the following steps are performed:

1. Generate *n* circles with a common center with Cartesian coordinates *O=(x, y)*. The radius of the innermost circle is equal to *r;* the radius of the second circle is equal to *2r,* and so on.
2. Divide circles into *k* sectors.
3. Create sector iterator *i = 0.*
4. Place a synchronization dot on the top of the innermost circle – at the point with coordinates *(x, y+r).*

5. Take subsequent $l = hcDotAuth\_capacity(n, k, d)$ bits of the data and store it in vector $v = [v_1, \ldots, v_l]$.
6. For every circle $c = \overline{2, n}$, starting from the outermost circle $c = n$:

a. Compute capacity l_c of circle number , $l_c = \left\lfloor \frac{2\cdot\pi\cdot c\cdot r}{d\cdot k} \right\rfloor$.

b. Take the first l_c bits of v, put them in vector $w = [w_1, \ldots, w_{l_c}]$, and remove them from vector *v*.

c. For every w_j for $j = \overline{1, l_c}$ check if w_j is equal to 1. If yes, place a dot at the point with coordinates $\left(x + c\cdot r\cdot \sin(\frac{2\cdot i\cdot\pi}{k} + \frac{\pi}{k\cdot l_c}\cdot(j-1)), y + c\cdot r\cdot \cos(\frac{2\cdot i\cdot\pi}{k} + \frac{\pi}{k\cdot l_c}\cdot(j-1))\right)$. If no, leave an empty space.

d. Compute $c = c - 1$, and if $c > 1$, go to step a.

7. Compute $i = i + 1$ and if $i < k$, go to step 6. Otherwise, the code creation process is completed.

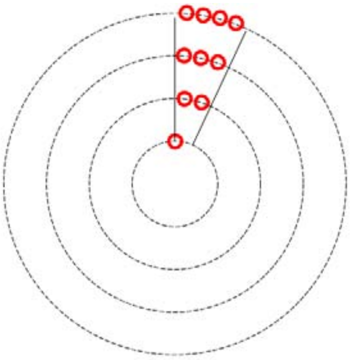


Fig. 6 hcDotAuth 2D code – synchronization dot and dots in one sector for a sample vector consisting of ones only.

C. Code Reading Principle

The process of reading the hcDotAuth(n,k,d) code is very similar to that described in Section II *B*. After recovering the synchronization location point $D = (x_D, y_D)$, the circle center $O = (x, y)$, the radius of the outermost circle R and angle α, the following steps need to be performed:

1. Generate *n* circles with a common center with Cartesian coordinates *O=(x,y)*. The radius of the innermost circle is equal to $r = \frac{R}{n}$; the radius of the second circle is equal to *2r*, and so on.
2. Compute code capacity $l = hcDotAuth\_capacity(n, k, d)$.

Initiate vector *v* with l elements.

Initiate sector iterator $i = 0$.

For every circle $c = \overline{2, n}$, starting from the outermost circle $c = n$:

e. Compute capacity l_c of circle number c, $l_c = \left\lfloor \frac{2\cdot\pi\cdot c\cdot r}{d\cdot k} \right\rfloor$.

f. Create vector $w = [w_1, \ldots, w_{l_c}]$.

g. For every $j = \overline{1, l_c}$ check if there is a dot near ($|\Delta d| < \frac{d}{4}$) coordinates $\left(x + c\cdot r\cdot \sin(\frac{2\cdot i\cdot\pi}{k} + \frac{\pi}{k\cdot l_c}\cdot(j-1) + \alpha) + \Delta d, y + c\cdot r\cdot \cos(\frac{2\cdot i\cdot\pi}{k} + \frac{\pi}{k\cdot l_c}\cdot(j-1) + \alpha) + \Delta d\right)$. If yes, set $w_j = 1$. If no, set $w_j = 0$.

h. Add data from vector w to the end of data in vector v. Compute $c = c - 1$ and if $c > 1$, go to step a.

Compute $i = i + 1$ and if $i < k$, go to step 5. Otherwise, the code reading process is completed.

Decoded vector *v* contains data stored in the hcDotAuth(*n*,*k*,*d*) code. There is no error correcting step in the hcDotAuth(*n*,*k*,*d*) process. It may be added at the expense of reducing the capacity of the code.

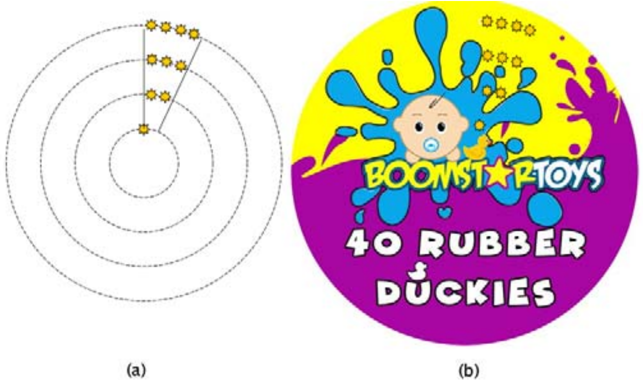


Fig. 7 hcDotAuth 2D code with custom symbols – a sample of the hcDotAuth code with one sector filled with ones (a) and the same hcDotAuth code printed a sample toy [9] label (b)

D. Custom Dots

The high capacity version of the hcDotAuth(*n*,*k*,*d*) code makes it possible to use other graphical symbols as well. Those graphical symbols may be related to the specific application of the hcDotAuth(*n*,*k*,*d*) code. For example, company marks could be used to secure company documents, or star marks could be used to mark toy labels. The only requirement is that the symbol used must fit within a circle with its diameter equal to *d*. Because of this feature, the hcDotAuth(*n*,*k*,*d*) code is highly flexible, meaning that it may be customized to satisfy the needs of specific applications and may be easily hidden within other graphical symbols.

E. Main Drawbacks and Advantages

The main drawbacks of the hcDotAuth(*n*,*k*,*d*) code include the following:

- Creation process of the hcDotAuth(*n*,*k*,*d*) code is more complicated than creation process of the DotAuth(*n*,*k*) code
- Reading process of the hcDotAuth(*n*,*k*,*d*) code is more complicated than reading process of the DotAuth(*n*,*k*) code
- If more graphical symbols are added to the existing graphical layout, readability may deteriorate.

The main advantages of the hcDotAuth(*n*,*k*,*d*) code include the following:

- Much higher capacity of the hcDotAuth(*n*,*k*,*d*) code allows to store not only identifiers, but also other data required to satisfy the needs of a specific application.
- The hcDotAuth(*n*,*k*,*d*) code allows to use custom graphic symbols instead of dots.
- As custom graphic symbols may be used, the hcDotAuth(*n*,*k*,*d*) code may be graphically integrated with

the current designs and applications.

- The system may contain data used, for instance, to perform simple offline authentication of the marked objects.
- Error correction outputs and other security algorithms may be added to the stored data.

V. Conclusion

In this paper, we have proposed a complete system for authentication of paper documents and other physical objects with a flat surface. The main part of the system has the form of a 2D code – DotAuth code – which may be placed under the text, rendering our solution suitable for existing document templates. We have provided complete algorithms for creating and reading the 2D code developed and come up with its high capacity version that may be used in other applications, also those of a special character. The codes developed are characterized by some very unique features that make them suitable for scenarios in which other 2D codes fail – e.g. they may be used to mark existing objects with a code that is seamlessly integrated with the existing layout and graphical design, or to place marks under the text in existing documents. We have described the architecture of the DotAuth system, capable of supporting all functionalities related to authentication and creation of the marked objects.

The architecture of the DotAuth system and the 2D code are the foundations of a user-friendly process used for authenticating products and documents, facilitating its widespread use. In our future work, we will focus mainly on analyzing security-related aspects of DotAuth/hcDotAuth codes and on validating DotAuth readability under different environmental conditions. Specifically, we will develop a version of the DotAuth code employing error correcting techniques to decrease the misread data rate. We will also provide a sample payload format – for the identifier and the local secret – used in the authentication process. Our future work will also tackle such issues as choosing the color of the code elements in order to adapt them to the medium on which they are used, maximizing legibility of text to the user, as well as maximizing readability of the code to vision recognition algorithms.

The DotAuth/hcDotAuth code proposed paves way to a way of thinking about 2D codes and their utilization. Currently, 2D codes are not integrated and form separate elements that fail to match, in many cases, the rest of the object (document) – the DotAuth/hcDotAuth code may be added without disturbing the graphical style and layout of the object concerned. Additionally, the 2D code proposed may be easily adapted to applications other than those concerned with object authentication. As long as a flat surface is available, the DotAuth/hcDotAuth code relying on custom symbols (dots) to store custom types of data may be used. Such a high degree of customization may be particularly useful in scenarios in which brand recognition is being established.

References

[1] Lindblom, B.S., Gervais, R.: Scientific Examination of Questioned Documents. pp. 238–241. Taylor and Francis, Boca Raton, FL (2006).

[2] Cheddad, A., Condell, J., Curran, K., & Mc Kevitt, P. (2009). A secure and improved self-embedding algorithm to combat digital document forgery. *Signal Processing*, *89*(12), 2324-2332.

[3] Elkasrawi, S., & Shafait, F. (2014, April). Printer identification using supervised learning for document forgery detection. In *2014 11th IAPR International Workshop on Document Analysis Systems* (pp. 146-150). IEEE.

[4] Van Beusekom, J., Shafait, F., & Breuel, T. M. (2013). Text-line examination for document forgery detection. *International Journal on Document Analysis and Recognition (IJDAR)*, *16*(2), 189-207.

[5] Huang, S., & Wu, J. K. (2007). Optical watermarking for printed document authentication. *IEEE Transactions on Information Forensics and Security*, *2*(2), 164-173.

[6] Warasart, M., & Kuacharoen, P. (2012, May). Paper-based document authentication using digital signature and QR code. In *4th International Conference on Computer Engineering and Tecnology (ICCET 2012).*

[7] Jumio homepage, https://www.jumio.com/trusted-identity/netverify/document-verification/, last accesed 02/05/2020

[8] Soon, T. J. (2008). QR code. *Synthesis Journal*, *2008*, pp. 59-78.

[9] Image retrieved at 02/17/2020 from https://www.freelancer.com/contest/Label-for-toy-1403253-byentry-23008342.

Michał Glet – is an Assistant at the Faculty of Cybernetics at the Military University of Technology in Warsaw, Poland. His research interests include cybersecurity, cryptography, cryptoanalysis, malware analysis, software reverse engineering, and software development.

Kamil Kaczyński – Military University of Technology, R&D assistant, Cryptography, Steganography, Blockchain, Cryptanalysis, Steganalysis, Mobile applications, Internet of Things (IoT)