A Modified Cloud-Based Cryptographic Agent for Cloud Data Integrity

https://doi.org/10.3991/ijim.v11i2.6553

Basma Hathout
The British University in Egypt, El Shorouk City, Cairo, Egypt
basma.hathout@bue.edu.eg

Samy Ghoniemy
The British University in Egypt, El Shorouk City, Cairo, Egypt

Osman Ibrahim
The British University in Egypt, El Shorouk City, Cairo, Egypt

Abstract—In spite of all the advantages delivered by cloud computing, several challenges are hindering the migration of customer software and data into the cloud. On top of the list is the security and privacy concerns arising from the storage and processing of sensitive data on remote machines that are not owned, or even managed by the customers themselves. In this paper, initially a homomorphic encryption-based Cryptographic Agent is proposed. The proposed Cryptographic Agent is based on Paillier scheme, and is supported by user-configurable software protection and data privacy categorization agents, as well as set of accountable auditing services required to achieve legal compliance and certification. This scheme was tested using different text documents with different sizes. Testing results showed that as the size of the document increases, the size of the generated key increases dramatically causing a major problem in regards to the processing time and the file size especially for large documents. This leaded us to the second part of this research which is: a modified security architecture that adds two major autonomic security detective agents to the multi-agent architecture of cloud data storage. In this paper, we focus on the first agent namely (Automated Master Agent, AMA) that is added to the Multi Agent System Architecture (MASA) layer (cloud client-side) by which any changes happen in the document are mapped in a QR code encoded key print (KP). Experimental results after integrating these agents showed a 100% alternation detection accuracy and a superiority in extracting the KP of large and very large size documents which exceeds the currently available products and leverage the tamper-proof capabilities of cryptographic coprocessors to establish a secure execution domain in the computing cloud that is physically and logically protected from unauthorized access.

Keywords—cloud data storage and processing security, document key print, homomorphic encryption, QR Codes.

1 Introduction

Due to the fast development of cloud computing technologies, there's been a remarkable increase in the cloud services which made the job for securing user's data more challenging and one of the hottest research areas [1]. In 2014, the international data cooperation (IDC) conducted a survey showing that 87.5% from IT executives to chief executive officers (CEOs) believe that the challenge that faces every cloud service lies in its security [2]. One of those challenges that raised due to the storing of data in different and distributed locations is the integrity of the data stored in the cloud [3], which requires robust integrity checker algorithms to ensure that the document didn't tampered prior its retrieval [4]. In addition to the distributed nature, the multitenancy of the cloud environment violates the user's privacy and confidentiality rights since multiple parties can access the stored data [5].

When user(s) outsource data to the service provider's data center, the primary risk that faces this data is its confidentiality and integrity. At this stage of the data life cycle; which is data in transit, it is very important to encrypt the data to ensure confidentiality as well as using security protocols to ensure the privacy [6]. When data arrives to the data center; data at rest in the life cycle, the users lose any physical possession over them. They only use Virtual Machines (VMs) interfaces to have partial control over the data [7], and the cloud service providers are responsible for managing the underlying systems and have constant access to the VM. This threatens the security in terms of its integrity and confidentiality [8]. Such threat is not at question by only unauthorized users trying to access and modify these files, but also by service providers themselves who do so for their own purposes or as a lack of security. For instance, they can discard the users' files that are not accessed very frequently by the users in order to better utilize their stored data on the data centers [9]. Moreover, the multi-tenancy nature of the cloud; which allow service providers to reconfigure resources such as VMs to multiple customers, any misconfiguring to these VMs, does not only affect the corresponding customers, but all the other customers running on this host, as it gives the attacker an entry point to the host machine which result in affecting the underlying platform [8]. In addition to that, since the shared resources are separated virtually and not physically [10], this results in leaking of information by having residual data and/or operations and so violates the confidentiality of the data [11]. Another point is that, since these resources need to be allocated quickly to meet a specific demand, the service providers does not share how these resources are wiped before being reassigned [8].

In addition to the multitenancy and distributed nature, the service providers lack of transparency in providing the customers with the security incidents and measures pose a threat to its security. For example, they do not provide the customers with feedback when a security incident is detected so as to maintain a reputation [12] or what are the security measures they take in order to divert them or how the customer's data are protected during the investigation process [8].

Encryption is the obvious approach for protecting the data at rest. But what about the cloud applications that uses these data to operate on them? Using traditional encryption algorithms wouldn't be feasible any more, as it would require the service

providers to decrypt the data first which wouldn't only subject it to confidentiality risk, but integrity as well [6]. According to a survey done by The European Network and Information Security Agency in [13], among the security risks that debilitate the adoption of cloud computing is the absence of customer's data auditing. As data auditing entitles the service providers to take the appropriate measures to ensure user's data security and gives the users the ability to verify that these measures are up and running, i.e. it gives the users the transparency of how their data is being handled [13].

2 Related Work

Since the emerging of cloud computing paradigm, and its data security and privacy has been studied extensively since its one of its main concerns [14], and since traditional technologies for checking data integrity are no longer applicable for environments with remote data [15], a lot of schemes has been proposed to enhance cloud security in terms of its data integrity. Ref. [16] published a Provable Data Possession (PDP) scheme; that allow users to verify the service possession of the data without having to retrieve it. The major drawback of this scheme is that it only deals with static files [17]. However, this scheme was subsequently enhanced to support dynamic files as published in [18], it did not fully support dynamic data operations such as block modification, deletion and appending, as well as it had a limited number of queries [17] [18] [19]. Although, the original and modified PDP schemes were further extended by allowing insertion, modification and deletion of any blocks as in [20], and to reduce the computational and communication complexity as in [17], they unfortunately have not yet found widespread acceptance in practice due to their computational and communications burdens. Furthermore, in [4] [2] another remarkable scheme was published as an attempt to verify remote data integrity prior its retrieval for large files. However, the authors of this scheme claims that extension into POR-based assurances around data availability guarantees privacy and integrity of stored documents, they notified that computation and communication are serious problems preventing the practicality of POR system. Moreover, like PDP, it bounded the users with limited number of queries. It also does not support dynamic files. Further improvements for the POR scheme described in [4] have been introduced in [21] by providing full proofs of security against arbitrary adversaries. In the improved solution, the authors designed one scheme for public verifiability and another for private verifiability [22]. However, the improved scheme is well designed, it lost its advantage as it stored part of the file and the authenticators on the service provider server which made the document protection not fully guaranteed against alterations. Another research for verifying remote data integrity by computing hash value for the whole file was introduced in [23]. Regardless the benefits of using hash value for the whole file, this scheme is not practical for large files since it requires exhaustive computational time to compute and transfer the hash values [24].

In contrast to the above mentioned schemes, [3] proposed a threat model that solves the data privacy issue in cloud computing and addressed the preservation of

data integrity by including digital signature techniques. With attention to what have been introduced [3], [25] proposed a simple mechanism that tried to achieve both the storage security and the user authenticity. This work is not more than a conceptual framework without evidence of practicality. Moreover, [1] implemented a system for cloud data security. Notwithstanding this work is based on well know and trusted algorithms (blowfish and RSA), it couldn't be considered a practical solution for the problem of data integrity because it didn't present neither a solid deployment nor experimental results.

From what was previously discussed, we can conclude that all the schemes that was proposed trying to solve the data integrity issue for remote data focused on static files only; by ignoring the fact that users do not just access the file(s), but they can also update them through different file operations such as insert, delete and modify. Moreover, the few schemes that managed to handle dynamic files faced two problems. First, acceptance problem because of the computational and communication overhead. Second, they limited the number of queries a client can apply on files. Furthermore, the work previously done on data integrity in cloud computing didn't managed to ensure the aspect in question by simply tackling file content.

3 Conventional Data Integrity using Homomorphic Encryption Algorithm

Since the customer lose the possession of the data's storage and management to the service provider, privacy and integrity of the data becomes a security issue that requires proper handling. One of the highly recommended algorithms that was proposed to handle these security issues is homomorphic encryption. It was proposed to allow securing the privacy of data in transmission, storage and processing [26]. Homomorphic encryption is a special type of encryption that "enhance the security measures of untrusted storage systems that stores and manipulate sensitive data" [27]. It gives the ability to perform operations on encrypted data without having to decrypt them first which in an architecture like cloud would be very effective, i.e. the user transmits the encrypted data to the service provider for storage which guarantee the privacy and confidentiality of the data during transmission and while resting in the cloud storage. Moreover, since it applies the user operations while it is encrypted, it keeps the confidentiality and privacy of the data intact during processing [27]. Any Homomorphic schemes consists of four algorithms. Key generation, encryption, decryption and homomorphic property [28]. Homomorphic property can be divided into two types, additive and multiplicative. This property defines the type of operation it can perform on the encrypted data. A lot of schemes has been developed in an attempt to reach a scheme that is efficient, with low expansion rate and minimum computational cost. One of these schemes was Paillier scheme.

3.1 Homomorphic Encryption using Paillier's Scheme

This scheme was introduced in 1999 as a probabilistic asymmetric algorithm for public key additively homomorphic system [26]. This scheme made it as one of the most popular additively cryptosystem, due to its: simplicity, performance, low cost of the encryption process; because of the fact that it allows the encryption of many bits in one operation, efficiency of the decryption process with a constant and low expansion factor that is equals to 2, which is considered to be small compared to other schemes. Moreover, in spite that this scheme falls under the umbrella of additively cryptosystem, it also supported multiplication operation by a constant [29] [30] [31].

Key Generation:

Public key: compute the following:

$$n = pq \tag{1}$$

Where p and q are two large coprime numbers (i.e. GCD = 1) and the GCD is the greatest common divisor that is generated using the following equation: [32].

$$GCD = (pq, (p-1)(q-1)) = 1 \tag{2}$$

Private Key: compute the following:

$$\lambda = \mathrm{LCM}\big((p-1), (q-1)\big) = \frac{(p-1)(q-1)}{GCD(p-1, q-1)} \tag{3}$$

Where λ known as Carmichael's function that is the LCM of all the numbers that are less than or equals to n and are also coprime to n and LCM (least common multiple) of two numbers [33].

Encryption:

$$\mathrm{c} = \mathrm{g^m r^n}. (\mathrm{mod}\ \mathrm{n^2}) \tag{4}$$

Where $m \in Z_n$, $g \in Z^*_{n^2}$ and $r \in Z_{n^*}$.

Decryption:

$$m = L\big(c^{\lambda}\ mod\ n^2\big).\mu\ mod\ n \tag{5}$$

Where $c \in Z^*_{n^2}$, $L = \frac{u-1}{n}$ and μ is a multiplicative inverse that only exists if g is valid and is computed as following:

$$\mu = \left(L\left(g^{\lambda}(mod\ n^2)\right)\right) - 1(mod\ n) \tag{6}$$

3.2 Implementation of the Homomorphic Encryption Algorithm

A homomorphic encryption-based Cryptographic Agent is proposed. The proposed Cryptographic Agent, is based on Paillier scheme for high encryption and decryption efficiency, low expansion factor and supports multiplication operation by a constant. In this agent, the service provider sends to the trusted third party requesting the generation of the public and private key pair. The third party simultaneously send the public key to the cloud service provider and the public and private keys to the client. Immediately after the client receive the key pair, (s)he uses the public key to encrypt the document in question. The encrypted secured document is then sent through a communication link to the cloud service provider data center for storage. When a user requests to operate on the document, the service provider first authenticate this user. In case of authorization, it retrieves the document in question from the data center and passes it to the security scheme together with the public key to apply the user's requested operation on it. The resulted encrypted document is then simultaneously sent back to the service provider data center for storing an updated version of the document and the user, who by turn uses the corresponding private key to decrypt and retrieve the resulted document. On the other hand, in case of unauthorized user, the service provider denies and terminate the user request. Fig. 1 illustrate the system architecture.

As it has been illustrated, the architecture consists of three different entities: Cloud Client who is responsible for encrypting the document in question, upload it to the service provider data center, request to operate on the document and finally decrypting the retrieved document; the cloud service provider who provide data storage service for storing the user's data, authentication process for the user requests, processing the user's query for operating on the document(s) and managing, securing and controlling the transfer process of the document(s) to the user; trusted third party responsible for generating and distributing the key pair.

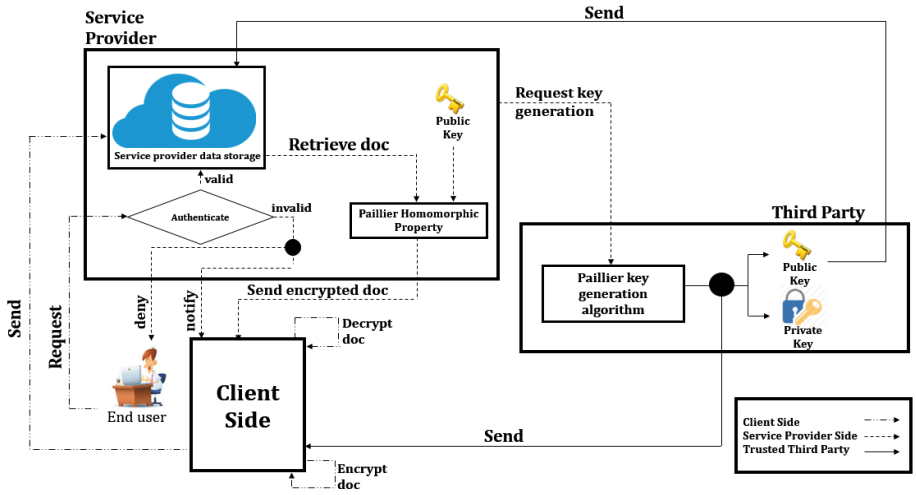


Fig. 1. Cryptographic Agent System Architecture.

3.3 Results and discussion

This agent was tested using text documents. We used 10 samples with different file sizes ranging from 5 KB to 132 KB; as shown in the following table.

Table 1. TEST SAMPLES

Sample Number	File size in KB
Sample 1	4
Sample 2	19
Sample 3	33
Sample 4	48
Sample 5	62
Sample 6	76
Sample 7	90
Sample 8	104
Sample 9	119
Sample 10	132

In choosing our test samples, we assumed that a document has a maximum of 30 lines; representing a page, and based on this assumption, we increased the document's size by increasing the number of lines and get the equivalent size in KB. We started our test samples with a document that is made up of 60 lines (2 pages) and increased the number of lines by 15. We then use the following equation to get the number of lines for a specific element.

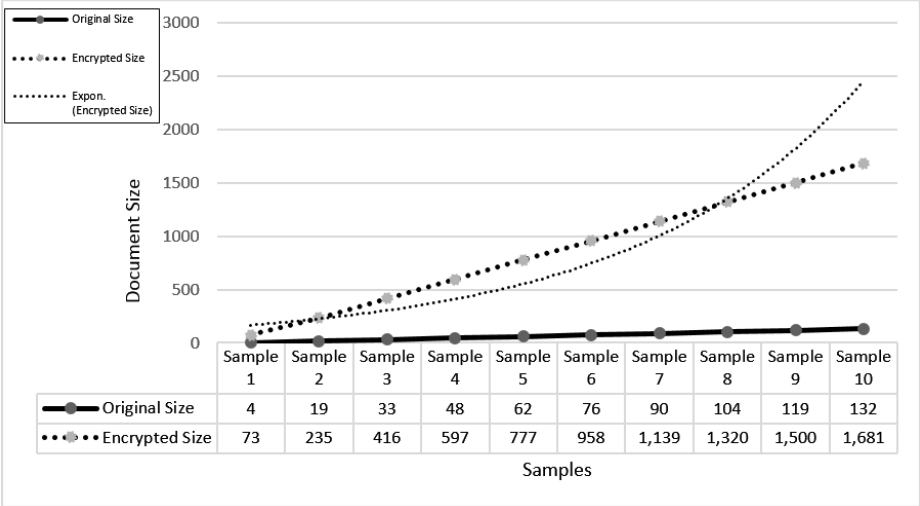
$$a_n = 60 + (n - 1)15, \tag{7}$$

We used this equation to get the number of lines for elements 1, 10, 20, 30, 40 and so on until getting 10 elements representing the 10 samples.

The testing methodology was as follows. First, we tested the homomorphic encryption algorithm and analyzed it in terms of the change in the file size before and after the encryption and the time it took to encrypt the document. We then tested the decryption process by comparing its processing time with respect to the encryption. Moreover, we tested the efficiency of the scheme by testing how efficient the decryption process was in retrieving the original plaintext.

The test results show that as the document's size increase, the resulted encrypted document's size increased significantly, i.e. as shown in the graph in fig. 2, when the document was only 2 pages long and 4 KB in size, the resulted encrypted document expanded to 73 KB and as this size increased to 90 KB and above, the encrypted document's size started to deviate to being greater than 1 MB. Moreover, the results also illustrate that when the expansion rate was calculated for each sample using equation 8, it showed that the encrypted document is almost 12.4 times larger than the original document.

$$expansion\ rate = \frac{encrypted\ size}{original\ size} \tag{8}$$



	Sample 1	Sample 2	Sample 3	Sample 4	Sample 5	Sample 6	Sample 7	Sample 8	Sample 9	Sample 10
Original Size	4	19	33	48	62	76	90	104	119	132
Encrypted Size	73	235	416	597	777	958	1,139	1,320	1,500	1,681

Fig. 2. Original vs Encrypted Document Size.

This means that, as illustrated with the dotted curve, the encrypted document size would increase exponentially as we keep increasing the original file size. For example, if we have a file that is 1 MB in size, the encrypted size would be approximately 12.4 MB which in an environment that deals with very large sized documents like the cloud would not be practical.

The graph in Fig 3 shows that when the document reaches a size as inadequate as 132 KB, it took approximately 3 minutes to be encrypted. In addition, it shows that the time would continue in increasing exponentially as the size increases as illustrated by the dotted curve. This conclude that the time is directly dependent on the document's size; as when the size increase, the time taken to encrypt it increased as well.

The computation overhead of the decryption process as seen on Fig. 4, is similar to the encryption, in the sense that; as the document size increased, the time taken to decrypt it increased as well. For example, when the original document size was as small as 132 KB and its encrypted version was almost 1.6 MB, the process to decrypt such document took almost 4.6 minutes. The graph also shows that as the size of the sample continue to increase, the processing time will exponentially increase as well; as shown by the dotted curve.

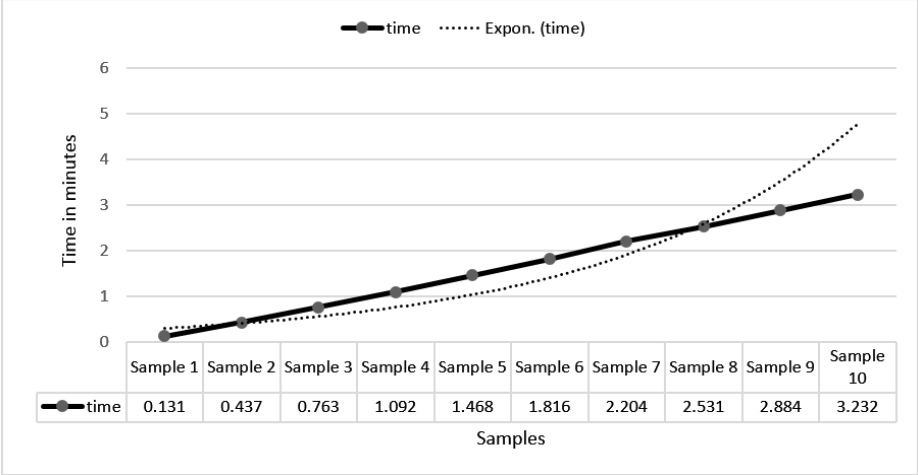


Fig. 3. Encryption Processing Time.

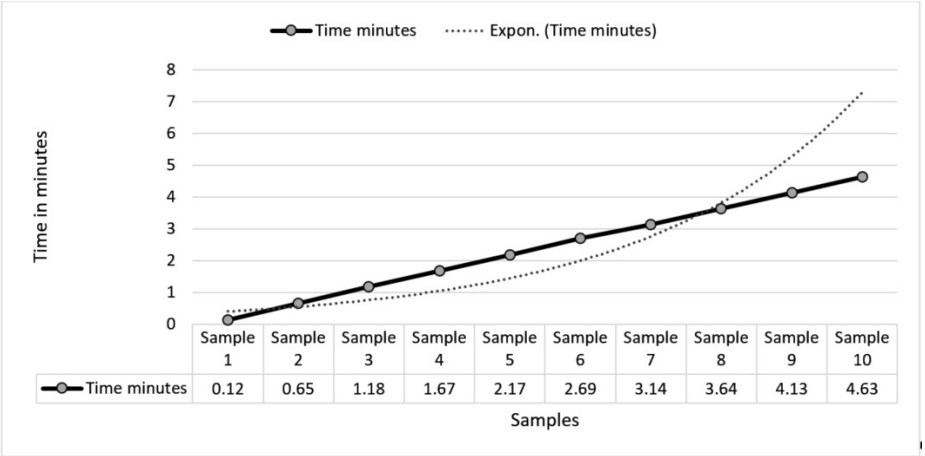


Fig. 4. Decryption Processing Time.

Fig. 5 illustrate the time take by each of the samples during the encryption and decryption processes. The graph shows that the decryption process took more time than the encryption. Moreover, it shows that even through the time for both processes exponentially increase with the increase of the document's size; represented by the dotted and dashed curves for encryption and decryption respectively, the decryption curve is higher than the encryption, with a ratio approximately equals to 1.4. Which means that the decryption process would always take time greater than the encryption by a factor of 1.4.

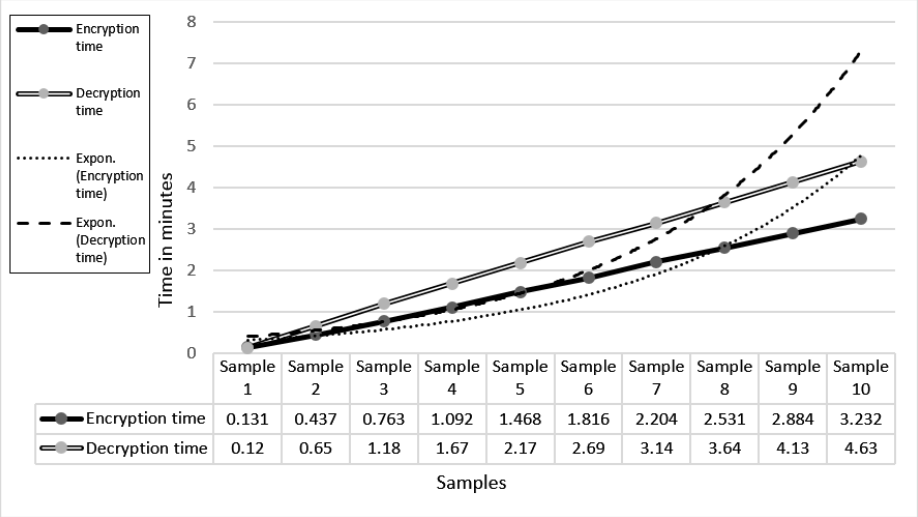


Fig. 5. Encryption vs Decryption Processing Time.

3.4 Paillier Security analysis

For asymmetric schemes, their security depend on the firmness of the mathematical problem upon which the schemes are built. On one hand, some would argue that these problems are hard to solve and the only way to break them, if there's a prior knowledge about the keys, which makes these schemes secure. On the other hand, assessing the security level of these schemes regarding only the assumption that their mathematical problem is hard to solve isn't correct. This is because it is believed that are other ways to break a system. For instant, having any knowledge about the ciphertext can threaten the security of the scheme. Moreover, most of the schemes where proven to be secure under a model called random oracle model; which is an idealized model that tests the schemes under unrealistic assumptions, and so doesn't fully assess the scheme's practicality [28]. Paillier scheme has proven security against indistinguishable chosen plain text attack (IND-CPA). This type of security notion states that; the ciphertext does not reveal any information about the plaintext other than its length. This prove was done under the decisional composite residuosity (DCR) assumption; which states that: it is computationally challenging to decide whether an integer z is an n-residue modulo n^2 or not, given that n is a composite integer. Moreover, due to the homomorphic property, this scheme is not protected against adaptive chosen ciphertext attack (IND-CCA2) which is considered to be the highest level of security for an encryption algorithm [32] [34].

4 Modified Data Integrity Framework

The results and analysis presented in sections 3.3 and 3.4 showed that however homomorphic encryption facilitates the idea of being able to operate on documents while encrypted, its large expansion rate, high computational time and storage overhead; especially for large sized documents, make the use of homomorphic encryption in cloud environment impractical way for document's privacy and integrity. These problems are alleviated in the modified data integrity framework as it is discussed in the following section.

The proposed security architecture adds two major autonomic security detective agents to the multi agent architecture of cloud data storage. The first agent is the Automated Master Agent (AMA), while the second agent is the Automated Detection Agent (ADA). The earlier is responsible for extracting the document's signature, converting this signature to its equivalent QR code representation and finally concealing this QR code in the document. This happen through the Document Signature Extraction and Hiding (DSEH) process; shown in Fig. 6, that consists of three cooperative sequential sub-agents. These agents are; *(1) Key Print (KP) Agent* that is used for extracting the ASCII pattern for any given document which employed to extract the key print based on the document's content. *(2) QR Code Encoding (QRCEnc) Agent*, which receives the key print from the previous agent, preprocess it by applying three logic operations, namely binary conversion, binary addition and hexadecimal conversion, then it generates the corresponding encoded QR code. (3) *Hiding Agent (HA)* which is used for concealing the generated QR code in any graphical placeholder or in the document's header.

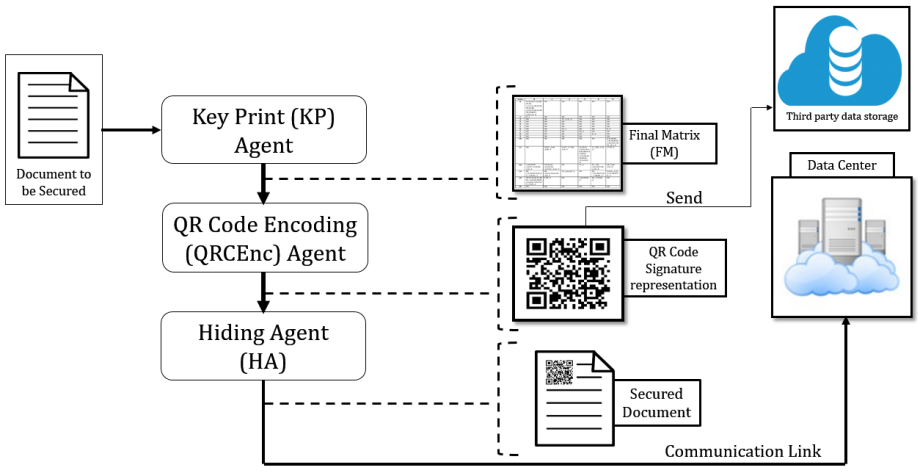


Fig. 6. Document Signature Extractions and Hiding (DSEH) Process.

In this model as shown in Fig. 7, cloud client uploads the document that could be a text, word or a Portable Document Format (PDF) file. The uploaded document then goes through the Document Signature Extraction and Hiding (DSEH) process where

the document signature is extracted, converted to QR code that is simultaneously sent to the third party, as an independent verifier, and hidden in the document itself in either a graphical placeholder or its header. The resulted secured document is then outsourced over the communication system to be stored on the cloud service provider's data center. When a user sends a request to retrieve a document the server authorizes this user. In case of authorized user, the cloud server act accordingly by sending the requested document to the third party, otherwise it sends a notification to the client and deny the access to that document. When the third party receives the document, it verifies its integrity. If verified it sends the requested document to the user, otherwise it notifies the client and sends a log file to the service provider.

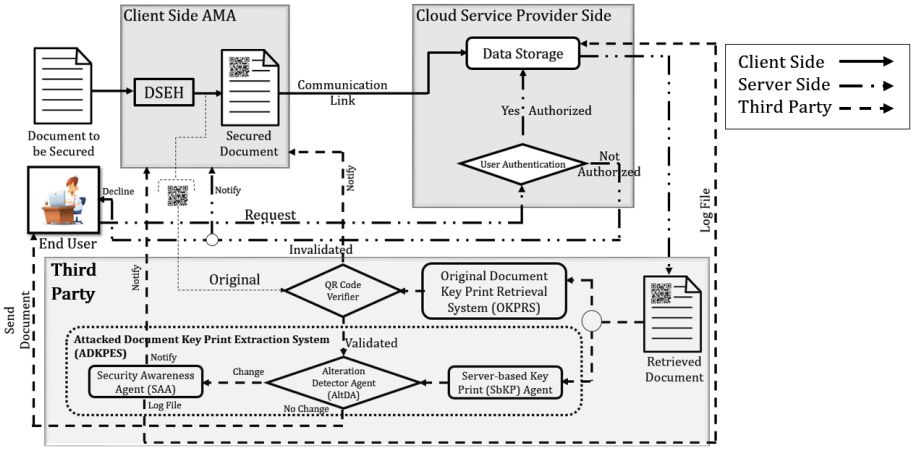


Fig. 7. Overall Modified System Architecture.

4.1 Key Print Agent

This agent works on generating digital signature for an uploaded document. The signature depends on the content of the document which makes it distinctive by never having two documents with the same signature. This agent starts by defining two 16 x 6 matrices *Initial Matrix (IM)* and *Final Matrix (FM)*. *IM* is the first of the two matrices defined by the *Key Print Agent*. It generates the 96 graphical codes which include letters (upper and lower case), numbers and special characters. The *FM*, which is the output of this agent, and the second of the two matrices that would be the input for the *QR Code Encoding Agent*. When this matrix is first created, its cells are initialized with a default string "NA" and its final state would be the document key print generated from its content. After creating these matrices, a hash table is then created. This table have a <key, value> combination which represent each of the 96 codes as keys and their corresponding position in the *IM* is regarded as the value associated with the key.

After a document is uploaded, it is read in a line by line manner, where the *Character Positions per Line (CPL)* is extracted followed by the line number. The same

process is repeated until reaching the end of the document. By then, a table with all the characters in the document with its corresponding positions and line numbers are represented. This table together with the hash table mentioned earlier are used to fill the *FM* in the following manner: the extracted positions would be placed in the *FM* based on the cell that corresponds to this character's index as provided in the IM. This index is represented by the row and the column numbers in the *IM*. For example, as shown in Fig. 8 (which further clarifies this process of updating the Final Matrix) in the hash table the position of character 'a' in the *IM* is in row number 1 and column number 4, so when placing the *CPL* of this character in the *FM*, it would be placed in row number 1 and column number 4.

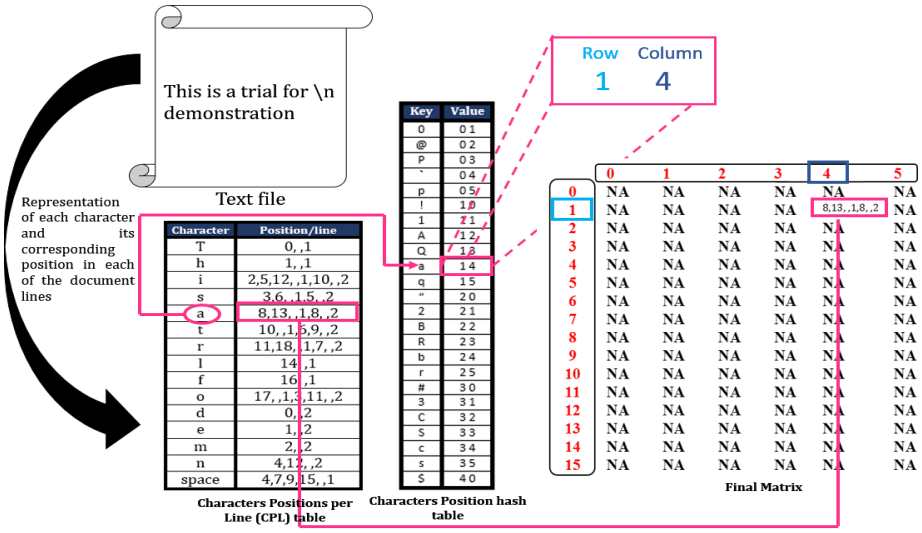


Fig. 8. Process of Filling FM

In the *CPL* representation, a line separator is used in order to distinguish between the positions and the line number. The separator is assigned a constant value; character (0) which is the decimal representation of "null". Fig. 9 shows an example where the first two numbers represent a Character's Positions, followed by the separator and then the line number.

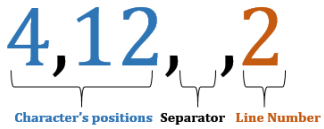


Fig. 9. Character per Line (CPL.)

4.2 QR Code Encoding (QRCEnc) Agent

This agent takes the *FM* coming from the previous agent as an input and generates the corresponding QR code. This agent goes through the *FM* and for each cell three operations are applied prior to the generation of the QR code. The three operations are shown with example in Fig. 10

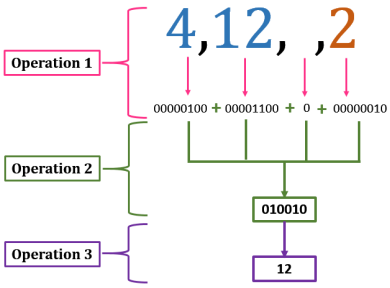


Fig. 10.QR Code Agent Operations.

And are as follows:

Operation 1: for each cell in the *FM*, it converts the numbers representing the *CPL* to its equivalent in binary format. When a cell has "NA" it is ignored as it signifies that this graphical code was not found in the content of the document.

Operation 2: the resulting binary equivalents are added where each cell is now represented by only one binary number, which is the result of the addition.

Operation 3: The final binary resultant is then converted to its equivalent hex format, and is then appended in a string with the output of the other cells.

Fig. 11 shows the output of operation 3 which is the final hex string representing the document's signature and its equivalent generated QR code.

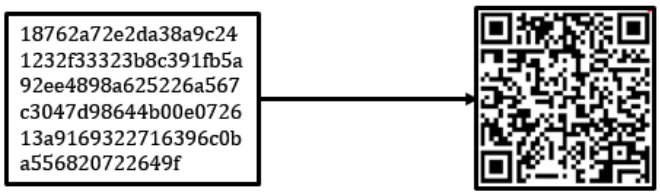


Fig. 11. Hex string and its corresponding QR Code.

4.3 Results and discussions

The proposed modified model was tested using two sets of documents. The first set was the previously mentioned samples in Table 1. While the second set was additional 11 samples of different file type and size; in terms of number of characters per document and the number of lines, shown in Table 2.

Table 2. Automated Master Agent (AMA) Specific Samples.

File Type	Number of characters	Number of line	Sample Number
Text file	4800	15	Sample 1
	15600	51	Sample 2
	123600	678	Sample 3
	1203600	7480	Sample 4
Word Document	4800	60	Sample 5
	15600	195	Sample 6
	123600	1545	Sample 7
	1203600	15045	Sample 8
Pdf	4800	47	Sample 9
	15600	156	Sample 10
	123600	1211	Sample 11

Two aspects were targeted from this test. First was the verification of the *QRCEnc Agent* and second was the validation of the same agent in terms of the processing time, compared to the Cryptographic Agent.

The test results shows that, the system was successful in generating the QR code for all the test samples of both sets. It showed that the model was able to generate key print, even when the size of the document reached 1,203,600 characters and 15045 lines (Sample 8). Fig. 12 shows the output for samples 1, 5 and 10 of the second set.

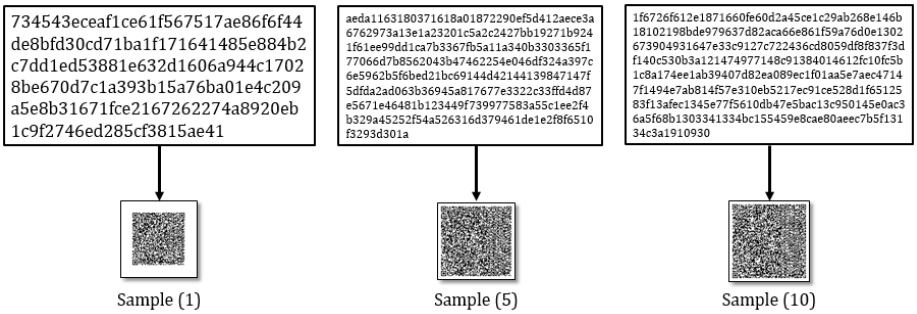


Fig. 12. Samples of generated QR code.

Moreover, the results also showed that the time taken to extract the key print for the samples in Table. 1 was inadequate compared to the results of the Cryptographic Agent as shown in Fig. 13. For example, sample 10 in the Cryptographic Agent took almost 3 minutes to be encrypted, while with the AMA, it only took 0.02 minutes for a key print to be generated and converted to its corresponding QR code.

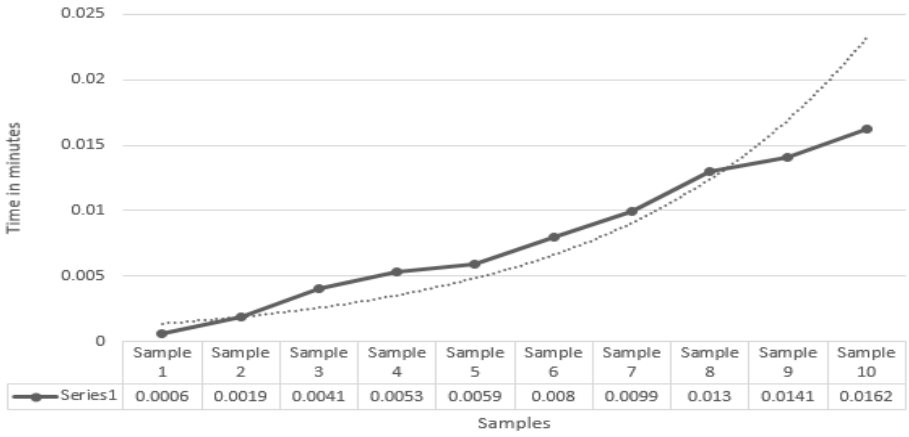


Fig. 13. Automated Master Agent (AMA) processing time.

5 Conclusion

The results of both models conclude that, for the Cryptographic Agent, due to the use of large key size (which was 2048 bits), we ended up having an expansion rate that is approximately equals to 12.4; which when dealing with very large size documents, like the ones used in a cloud environment, would end having very large encrypted documents that would require a lot of computational time, to either operate or decrypt them (which was proven by the results). On the other hand, the AMA proved superiority in extracting the KP of large and very large documents and the time it required to do so was very small compared to the results of the Cryptographic Agent.

6 References

[1] Rani, D. and Ranjan, R. (2014). Enhance data security of private cloud using encryption scheme with RBAC. *International Journal of Advanced Research in Computer and Communication Engineering*, 3(6), pp.7330-7337.

[2] Itkar, S. and Raut, V. (2014). A Survey on Data Integrity of Cloud Storage in Cloud Computing. *International Journal of Advance Foundation and Research in Computer*, 1(2), pp.58-65.

[3] Attas, D. and Batrafi, O. (2011). Efficient Integrity Checking Technique for Securing Client Data in Cloud Computing. *International Journal of Electrical & Computer Sciences*, 11(5), pp.14-45.

[4] Juels, A. and Kalishki, B. (2007). PORs: Proofs of Retrievability for Large Files. In: *14th ACM Conference on Computer and Communications Security*. ACM.

[5] Gellman, R. (2009). *Privacy in the Clouds: Risks to Privacy and Confidentiality from Cloud Computing*. [online] World Privacy Forum. Available at: http://gato-docs.its.txstate.edu/vpit-security/policies/WPF_Cloud_Privacy_Report.pdf.

[6] Mather, T., Kumaraswamy, S. and Latif, S. (2009). *Cloud security and privacy*. Beijing: O'Reilly.

[7] Wei, L., Zhu, H., Cao, Z., Dong, X., Jia, W., Chen, Y. and Vasilakos, A. (2014). Security and privacy for storage and computation in cloud computing. *Information Sciences*, 258, pp.371-386. https://doi.org/10.1016/j.ins.2013.04.028

[8] Pearson, S. and Yee, G. (2013). *Privacy and security for cloud computing*. London: Springer. https://doi.org/10.1007/978-1-4471-4189-1

[9] Goyal, R. and Sidhu, N. (2014). Third Party Auditor: An Integrity Checking Technique for Client Data Security in Cloud Computing. *International Journal of Computer Science and Information Technologies*, 5(3), pp.4526-4530.

[10] Youssef, A. and Alageel, M. (2012). A Framework for Secure Cloud Computing. *International Journal of Computer Science*, 9(4), pp.487-500.

[11] Cloud Security Alliance, (2011). *Security Guidance for Critical Areas of Focus in Cloud Computing*. V 3.0. Orlando, FL: Cloud Security Alliance.

[12] Wang, C., Chow, S., Wang, Q., Ren, K. and Lou, W. (2013). Privacy-Preserving Public Auditing for Secure Cloud Storage. *IEEE Transactions on Computers*, 62(2), pp.362-375. https://doi.org/10.1109/TC.2011.245

[13] Rasheed, H. (2014). Data and infrastructure security auditing in cloud computing environments. *International Journal of Information Management*, 34(3), pp.364-368. https://doi.org/10.1016/j.ijinfomgt.2013.11.002

[14] Liu, C., Ranjan, R., Yang, C., Zhang, X., Wang, L. and Chen, J. (2015). MuR-DPA: Top-Down Levelled Multi-Replica Merkle Hash Tree Based Secure Public Auditing for Dynamic Big Data Storage on Cloud. *IEEE Transactions on Computers*, 64(9), pp.2609-2622. https://doi.org/10.1109/TC.2014.2375190

[15] Yu, Y., Au, M., Mu, Y., Tang, S., Ren, J., Susilo, W. and Dong, L. (2014). Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage. *International Journal of Information Security*, 14(4), pp.307-318. https://doi.org/10.1007/s10207-014-0263-8

[16] Ateniese, G., Burns, R., Herring, J., Kissner, L. and Song, D. (2007). Provable Data Possession at Untrusted Stores. In: *ACM Conference on Computer and Communications Security*.

[17] Khalkar, R. and Patil, S. (2013). Data Integrity Proof Techniques in Cloud Storage. *International Journal of Computer Engineering & Technology*, 4(2), pp.454-458.

[18] Ateniese, G., Pietro, R., Mancini, L. and Tsudik, G. (2008). Scalable and Efficient Provable Data Possession. In: *4th International Conference on Security and Privacy in Communication Netowrks*.

[19] Wang, Q., Wang, C., Ren, K., Lou, W. and Li, J. (2011). Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(5), pp.847-859. https://doi.org/10.1109/TPDS.2010.183

[20] Erway, C., Papamanthou, C. and Tamassia, R. (2009). Dynamic Provable Data Possession. In: *16th ACM Conference on Computer and Communication Security*. ACM, pp.213-222. https://doi.org/10.1145/1653662.1653688

[21] Shacham, H. and Waters, B. (2008). Compact Proofs of Retrievability. In: *14th International Conference Theory and Application of Cryptology and Information Security: Advances in Cryptology*. Berlin Heidelberg: Springer, pp.90-107. https://doi.org/10.1007/978-3-540-89255-7_7

[22] Boneh, D., Lynn, B. and Shacham, H. (2004). Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4). https://doi.org/10.1007/s00145-004-0314-9

[23] Deswarte, Y., Quisquater, J. and Saïdane, A. (2004). Remote Integrity Checking. In: *Integrity and Internal Control in Information Systems VI*, 1st ed. Berlin, Germany: Springer US, pp.1-11. https://doi.org/10.1007/1-4020-7901-x_1

[24] Al-Saiyd, N. and Sail, S. (2013). Data Integrity in Cloud Computing Security. *Journal of Theoretical and Applied Information Technology*, 58(3), pp.571-581.
[25] Wadhwa, A. and Gupta, V. (2014). Framework for User Authenticity and Access Control Security over a Cloud. *International Journal on Computer Science and Engineering*, 6(4), pp.138-141.
[26] Sharma, I. (2013). *Fully Homomorphic Encryption Scheme with Symmetric Keys*. Post-graduate. University College of Engineering.
[27] Chakraborty, N. (2013). Cloud Security Using Homomorphihc Encryption. In: *National Conference on Advances in Computing, Networking and Security*. Excel India Publishers, pp.112-115.
[28] Sen, J. (2013). Homomorphic Encryption: Theory & Application. In: *Theory and Practice of Cryptography and Network Security Protocols and Technologies*, 1st ed. InTech, pp.1-31. https://doi.org/10.5772/56687
[29] Fontaine, C. and Galand, F. (2007). A Survey of Homomorphic Encryption for Nonspecialists. *EURASIP Journal on Information Security*, 2007, pp.1-10. https://doi.org/10.1155/2007/13801
[30] Maimut, D., Patrascu, A. and Simion, E. (2012). Homomorphic Encryption Schemes and Applications for a Secure Digital World. *Journal of Mobile, Embedded and Distributed Systems*, 4(4), pp.224-232.
[31] Damgard, I. and Jurik, M. (2000). *A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System*. Aarhus, Denmark: Basic Research in Computer Science, Department of Computer Science, University of Aarhus.
[32] Yi, X., Paulet, R. and Bertino, E. (2014). *Homomorphic encryption and applications*. Cham [etc.]: Springer. https://doi.org/10.1007/978-3-319-12229-8
[33] Choinyambuu, S. (2009). *Homomorphic Tallying with Paillier Cryptosystem*. Rapperswil-Jona: University of Applied Sciences Rapperswil (HSR), pp.1-10.
[34] Galindo, D., Martin, S., Morillo, P. and Villar, J. (2002). *An efficient semantically secure elliptic curve cryptosystem based on KMOV*. [online] Cryptology ePrint Archieve. Available at: https://eprint.iacr.org/2002/037.pdf.

7 Authors

Basma Hathout, Samy Ghoniemy, and **Osman Ibrahim** are with The British University in Egypt, El Shorouk City, Cairo, Egypt (basma.hathout@bue.edu.eg).

This article is a revised version of a paper presented at the BUE International Conference on Sustainable Vital Technologies in Engineering and Informatics, held Nov 07, 2016 - Nov 09, 2016 , in Cairo, Egypt. Article submitted 19 December 2016. Published as resubmitted by the authors 03 February 2017.