2016

# Securing Cloud Computing's Blind-spots using Strong and Risk-based MFA

MT Dlamini
*University of Pretoria*, mdlamini@cs.up.ac.za

MM Eloff
*University of South Africa*, eloffmm@unisa.ac.za

HS Venter JHP Eloff
*University of Pretoria*, eloff@cs.up.ac.za

JM Blackledge K Chetty
*University of KwaZulu Natal*, DVCResearch@ukzn.ac.za

# 58. Securing Cloud Computing's Blind-spots using Strong and Risk-based MFA

MT Dlamini
University of Pretoria,
mdlamini@cs.up.ac.za

MM Eloff
University of South Africa
eloffmm@unisa.ac.za

JHP Eloff, HS Venter
University of Pretoria
eloff@cs.up.ac.za

K Chetty, JM Blackledge
University of KwaZulu Natal
DVCResearch@ukzn.ac.za

## *Abstract*
Cloud computing presents an innovative technique to deliver computing as a set of services that can be consumed and utilized over the Internet. This technology breakthrough opens new doors for cybercriminals. The process of authenticating a new breed of cloud users who connect to cloud resources from anywhere, using any Internet-enabled devices and at any time complicates things more. This paper presents an innovative strong and risk-based authentication system. This is to deal with the rising issue related to unauthorized access of cloud hosted resources as a result of inadequate or weak authentication or even stolen user credentials. The proposed solution makes use of a risk engine which monitors user behavior in order to authenticate users based on specific risk indicators. Furthermore, this paper also makes use of an innovative encryption algorithm that takes chaotic random noise as input to generate encryption algorithms to help encrypt user authentication data at rest, in use and transit. The encrypted authentication data is then stored in multiple storage locations to improve its resiliency to cyber-attacks. This forms a key part of our contribution. The usage of multi-factor authentication systems can be argued to improve the overall system security by monitoring users interacting with a system and modifying defensive strategies to handle malicious behavior in a proactive manner. Our main goal for this architecture is to try and reduce cyber-criminal activities that are normally caused by weak authentication or stolen user credentials. The proposed solution has already been implemented as a proof-of-concept prototype to evaluate its applicability and suitability to achieve its goals. Despite a 7% of reported false positives or negatives, this solution is guaranteed to take strong risk-based multi-factor authentication on the cloud to the next level.

## *Keywords*
Cloud computing, Encryption, Multi-factor Authentication, User credentials, Steganography

## 1. Introduction
There is a rapidly increasing number of cyber-attacks that target cloud computing environments, more especially through static administrative user credentials (Kitten 2016). More cyber criminals are now relying on compromised user credentials linked to privileged accounts to attack systems. Given, the rising threats to user credential, it can be argued that traditional single-factor authentication measures like usernames and passwords (credentials) are no longer

sufficient to adequately protect cloud-hosted data (Kennedy et al. 2013; Webroot 2014; Dlamini, Venter & Eloff 2015; Raphiri, Dlamini & Venter 2015). However, many organizations are still using traditional user credentials to access cloud resources. Quite often these are stored and transmitted across vulnerable and untrusted networks in an unencrypted form. This opens up organizations for attacks such as man-in-the-middle which can be used to eavesdrop on user credentials as they are transmitted over vulnerable and untrusted networks in plaintext.

For example, a recent cyber-attack on Amazon Web Services Elastic Cloud Computing which started as a compromise on an employee's user credentials ended up putting a code-hosting company out of business (Butler 2014; Mimoso 2014; Rossi 2014; Venezia 2014 and RightScale 2015). Even though, it is not clear how the attackers landed on the administrative user credentials, the damage is far too much to ignore. Such attacks reflect cloud security's blind-spots and the increasing threat to compromise user credentials, more so administrative user credentials (Kitten 2016; Heller 2015). This is mainly because administrative user credentials grant attackers unlimited access privileges to management consoles or panels of cloud services like AWS EC2, Office 365, Google Apps and Rackspace among others. Hence, there is a need to improve the way we handle user credentials and authentication of users on the cloud. Furthermore, authentication techniques act as a front door for granting users authorized access to cloud services. Therefore, it important that authentication techniques be improved to best fit the cloud environment.

Research has shown that cloud computing demands new ways of handling user credentials and new authentication measures (Kennedy et al. 2013). Consequently, there is now a drive by most cloud service providers towards adopting and embracing multi-factor authentication measures. For example; Microsoft has roped in PhoneFactor to provide multi-factor authentication to their cloud based Microsoft Azure and Office 365 service offerings. Amazon makes use of Time based One-Time Passwords (TOTP) to provide multi-factor authentication to their AWS EC2 service offerings (Mathers 2016). This is to ensure that only authorized users can access cloud hosted data or applications. Multi-factor authentication measures require that user credentials must be validated using multiple techniques. However, hardware-based multi-factor authentication comes with a huge cost. For example the cost of hardware based multi-factor authentication in AWS service offerings ranges from $13 to $20 per device hardware (AWS 2016). AWS uses hardware based MFA to limit access to confidential data stored in AWS S3. Hence, they argue that in order to make it cost effective, the hardware based MFA should be only applied to administrators' accounts.

The rising cloud security challenge with regards to inappropriate handling of user credentials and inadequate or weak authentication techniques requires immediate action for a painless and seamless cloud adoption. Hence, the main research question that this paper answers is; how can we appropriately handle user credential and improve authentication techniques as the first line of defence to eventually increase the overall security of the cloud?

The rest of the paper is structured as follows. Section 2 discusses related work to identify existing research gaps. Section 3 presents and discusses our proposed strong and risk-based multi-factor authentication architecture. Section 4 evaluates the applicability and effectiveness of

our solution as implemented in our proof-of-concept prototype. Section 5 concludes the paper and provides future work.

## 2. Related Work

Inadequate or weak username and password authentication mechanisms leave cloud services like AWS, Microsoft Azure, Microsoft Office 365, Google Apps, Dropbox etc wide open to unauthorized access. This is prevalent in public cloud infrastructures whereby cloud services are shared by different users. The case where attackers stole millions of Dropbox user credentials is one classical example (Law 2014). Given the rising threat on user credentials, there is a need for a solution. Strong authentication presents a plausible solution and can play a key role as a first line of defense against cyber criminals on shared cloud services. Biometric scanning was initially offered as a better solution beyond the traditional username and password combination. However, the cost of biometric scanners is too high for most users. A number of researchers are now looking at cost-effective ways to provide strong authentication mechanisms for cloud services. For example, smartphone-based sensors are now being exploited to provide cost-effective voiceprints and fingerprints (Strom 2015). The work of CA Technologies (2014) adds user convenience on top of cost-effectiveness.

Raphiri et al. (2015) argues that strong authentication as a first line of defense can be used to secure the 'front door' to cloud computing. Hence, they argue that strong authentication hinges on using freely available multiple factors such as MAC addresses and geo-location coordinates to authenticate users based on their access devices and locations from which they request access. Approaching strong authentication from a banking perspective, Dlamini et al. (2015) extends the work of Raphiri et al. (2015) by making use of SIM card serial numbers, SMS texts, email, other factors and a concept called SurePhrase on top of MAC addresses and geo-location coordinates. Raphiri et al. (2015) sassume that contextual data and user credentials are already secure throughout the entire authentication process. Even though it is stated that user credentials are to be encrypted, the work of Dlamini et al. (2015) does not state explicitly how this is to be done. It also does not a make any comment on what other security mechanisms are to be used to secure contextual data and user credentials over the network and in use.

An interesting point of the work of of Dlamini et al. (2015) is the SurePhrase concept. This concept is meant to provide an extra level of authentication on top of one-time password (OTP) tokens. This concept originates from the concept of security questions that are normally used as an extra level of authentication beyond user credentials. However, instead of querying a user for an almost trivial answer to a security question, our system prompts the user to randomly choose a phrase. An example made in Dlamini et al. (2015) is a phrase "*hash tag David goes home at 6:00 sharp*" which results in the following SurePhrase "*#Dg$H@6^*". This work adopts the SurePhrase concept.

A number of global companies such as Google, Apple, Twitter, Facebook and LinkedIn are now using multiple factors which include OTPs to try and strengthen their authentication systems (Strom 2015). The research focus has moved beyond strong authentication mechanisms based on multiple factors towards a risk-based approach. A risk-based approach refers to authentication system that makes access decisions based on the risk posed to the cloud resources being requested by users. At the heart of a risk-based authentication system is a self-learning risk

engine. The risk engine makes use of multiple factors to scale and adapt access decisions based on risk indicators. Kennedy et al. (2013) defines a risk engine as an authentication system that continuously mines, analyzes and processes behavioral and context data. This data is then compared against the sensitivity of the requested resources to either scale up or down authentication factors depending on the risk posed. Strom (2015) defines it as risk-appropriate authentication that must consider numerous use cases and be able to evaluate minimum levels of accountability that is in line with the level of risk.

A risk-based authentication system must authenticate users based on a combination of what a user know (i.e. user credential or OTP), with something the user have (not necessarily a biometric feature but an access device which could be a mobile device, tablet or desktop.) and something they do (context or behavioral data) (Goode, 2015; EMC Corporation, 2013 and Saif, Siebenaler & Mapgaonkar, 2013). Goode (2015), EMC Corporation (2013) and Saif et al. (2013) argue that a risk-based authentication solution can improve security without burdening users with extra level of detail. The implication is that authentication must be done with ease of use and hide extra details in order to enhance the user experience. This is supported by CA Technologies (2014) which argues that authentication must scale up and down using a combination of factors to authenticate users in a cost-effective and convenient way. However, Goode (2015), EMC Corporation (2013) and Saif et al. (2013) does not discuss or comment even in passing about how the contextual data is to be kept secure and confidential throughout the process of authentication.

Some researchers refer to a risk-based authentication as adaptive authentication (RSA 2015). Some refer to it as context-aware authentication (Strom 2015). RSA's adaptive authentication uses device forensics and user behavioral analysis to balance strong authentication and usability. In support, Dlamini et al. (2015) argues that risk-based and strong authentication must be done in a seamless manner that does not get between users and their core duties. Strom (2015) extend the above work to include user roles and activities. The user activity proposed in Strom (2015) is similar to the behavioral or context data proposed in RSA (2015). The work of Webroot (2014) further extends strong and risk-based authentication discussions towards speed and efficiency. Therein, it is argued that authentication systems must be strong and scaled up or down depending on the risk posed. However, this must be balanced with speed and efficiency which raises ease and convenience of access.

The covered related work reflects that authentication is indeed one of the hottest security topic for cloud computing. Its state-of-the-art is transforming in an unprecedented way. However, the current-state-of-the-art seems to place more emphasis on adding more factors to provide seamless and fast risk-based authentication. It appears that related work is concerned about seamless (authentication done without burdening the user with extra level of detail) and on-the-fly type (high speed) of authentication. Most of the work seems to be in agreement on the use of multi-factors in conjunction with a risk-based approach. Hence, this paper makes use of multi-factor and risk-based authentication as a baseline and adds the SurePhrase concept. Most of the covered literature does not even mention how to secure user credentials or context data to be used for authentication. According to the authors of this paper, there is a research gap concerning how to handle contextual and user credential data throughout the process of authentication (at rest, in use and over the network). Therefore, this paper attempts to move beyond the current

state-of-the-art to close the research gap on how to secure contextual data used for authentication purposes.

The main contribution of this paper comes in presenting a secure way of authenticating users to access cloud resources. Moreover, it also focuses on how to handle behavioral or contextual data secure manner throughout the entire process of authentication and storage. The main goal is to ensure that users are properly authenticated and behavioral or contextual data used for authentication is to be appropriately secured at rest, in use and in transit. The next section begins by outlining system requirements and then discusses the details of the proposed solution.

# 3. Proposed Solution

This section considers the research gaps and strengths identified in the related work section above and formulate them into a number of system requirements. The main goal is to ensure that we ground our work on existing related work. Furthermore, we reflect on how this work advances the current state-of-the-art in handling user credentials or behavioral or context data used in authentication users to access cloud resources at rest, in use and in transit. Hence, this work adopts the strong and risk-based approach with the idea of a SurePhrase. Furthermore, it adds end-to-end security of contextual data used in authentication.

The system requirements for the proposed solution are as follows:

- Provide a suitable multi-factor authentication system that scales up and down as it authenticates users using multiple factors based on different risk indicators.
- Provide a novel and strong encryption algorithm and keys that are created, issued, controlled, managed and revoked by clients to secure user credentials and behavioral data at rest.
- Provide a secure technique that makes use of steganography to appropriately handle the delivery of user credentials in transit.
- Improve the security of encryption keys, SurePhrase and user credentials at rest.
- Provide a seamless and easy to use encryption key management system.

Based on the above system requirements, we present a three tier architecture which comprises of a graphical user interface, system processing and split database layer as depicted in figure 1 below.

The graphical user interface level provides the user interface management module which has an interface of the entire system. The system processing level is designed with components such as strong encryption, steganography and risk-based authentication. All these play important roles within the proposed multi factor and risk-based authentication architecture. The lower level is the database level which stores all the user credential and contextual data. This lower level is concerned with securing user credential and contextual data at rest. The following subsections provide a detailed analysis of the proposed architecture. Each component highlights how it achieves its goals and its contribution to the proposed solution.

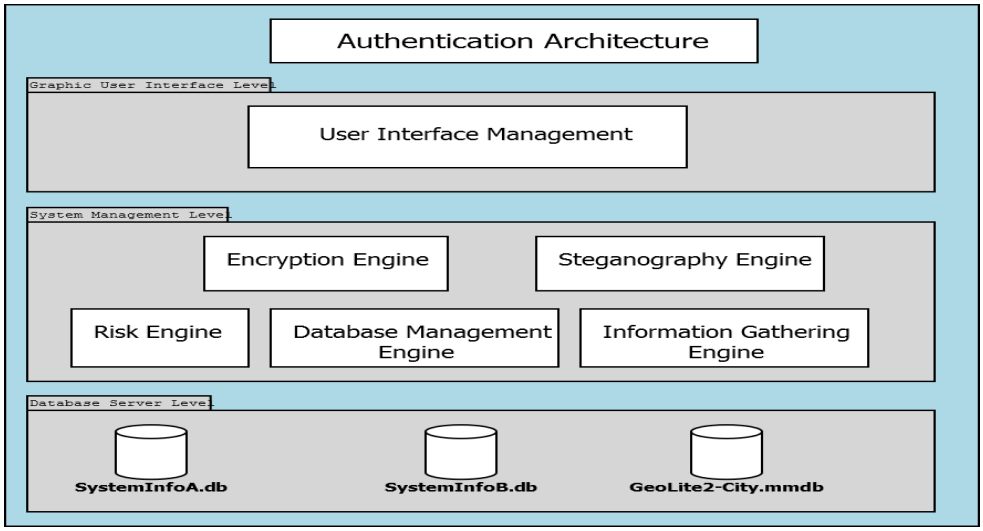


**Figure 1**: Strong and Risk-based Multi factor and Risk-based Authentication System Architecture

Note that the discussions start from the middle tier and goes to the bottom tier and ends with the interface tier. The idea is to explain all the processes that happen in the background before we actually show how the graphical user interface finally presents the data to the user.

## 3.1 Information Gathering Engine

Contextual data such as GPS coordinates, IP addresses and computer MAC addresses and other data are gathered as a user interacts with the system. The contextual data that is collected depends on the device or devices (tablet, laptop, desktop, mobile device) being used to access the system. For example, using a SIM enabled mobile device would help collect GPS coordinates which might not be possible on desktops and laptops. This data is collected starting from the point when a user first registers. This data is stored as historical data. A risk engine then draws upon the stored historical data when required and evaluates a users' threat level. Below is a snippet of the results of a risk engine capture from a desktop. This is then extracted and stored for evaluating a user's threat level later.

```
Current User Location
[userNumber1, South Africa, KwaZulu-Natal, Durban, 4001, -29.8561, 31.0352]
              User MAC Details
              [userNumber1, 5C-F9-DD-DE-55-3B, 146.230.97.72, WVL-CS-HONS-05]
```

## 3.2 Encryption Engine

The encryption engine is a vital part of our architecture. The following subsection discusses this process from the point when the encryption function is generated, to the generation of normalized floating point values until the actual encryption keys are generated. This is followed by a discussion on the decryption process in section 3.2.

### *3.2.1 Encryption Function Generation*

This paper adopts the work of Blackledge et al. (2013) which make use of a cloud-based Eureqa System (Eureqa 2015) to generate an encryption function. One could also make use of neural networks in place of the Eureqa system. However, the Eureqa system is a fast growing cloud-based solution which is gaining more widespread adoption from the research community. Our

choice was informed by its ease of use and short turn-around time to get the resultant fitness function. It also does not require a lot in terms of converting input data. The format is straight forward. The fitness function is derived from random and chaotic noise which is sourced from RANDOM.ORG (RANDOM.ORG 2015). However, noise generated from other sources could also be used; more especially natural noise like thunder, breaking waves, wind etc which is quite random and chaotic enough to strengthen the resultant encryption key to be generated. The author decided to get already existing noise. In future, we might need to make use of other natural noise from other sources. Figure 2 below depicts the process of a fitness function generation.

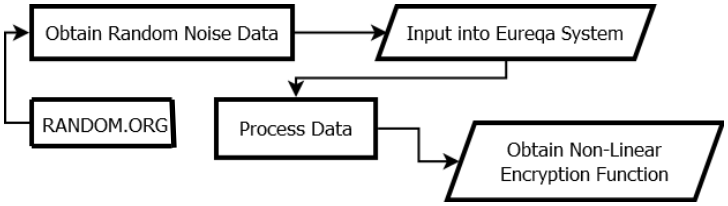


**Figure 2**: Process of Encryption Function Generation

Figure 3 below depicts a sample of the random noise sourced from RANDOM.ORG (RANDOM.ORG 2015) which was used as input into the Eureqa system (Eureqa 2015).

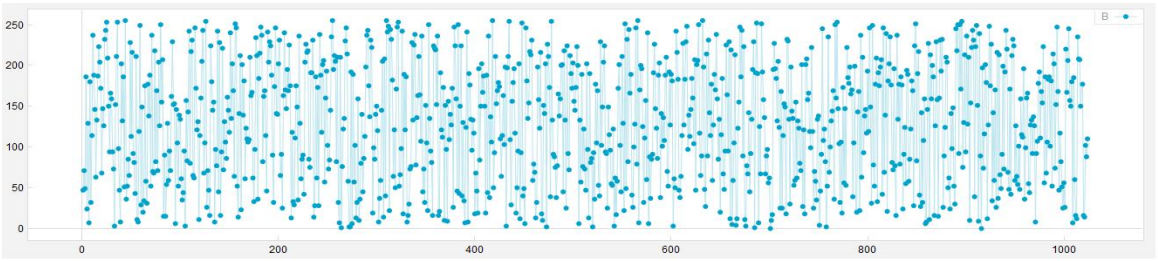


**Figure 3**: Distribution of Random Noise Sample

Essentially, the noise input requires must be left to run for sufficient time in order to produce the best non-linear fitness function that correctly maps to the random and chaotic noise input. The resultant non-linear fitness function is then implemented within the encryption engine as an encryption function. As it stands this process is done manually. However, we hope to automate it in future. This is used to generate encryption keys. The best resultant encryption function we could get after running the Eureqa system for 205 hours on a basic desktop in one of our labs is as follows:

$$f(x) = 16.4784884382193 * \cos(17.0431814003143 * x) + 22.0049234937009 * \cos(58.0995461530586 * x^2)) - 29.5769709843887 * \sin(5.14006029121902 + 0.00104086659951608 * x^2 + \sin(128.692647466496 * x) - 0.225891742277157 * x) \quad (1)$$

The weight coefficient (e.g. 16.4785884382193; 17.0431814003143; 22.0049234937009 etc) comes as 13 decimal floating point numbers from the Eureqa system and we decided to keep them as that since, rounding them off might result in a not so random and chaotic key generation

process. This was to ensure that we get a result that matches the input as close as possible. The Eureqa system has now been linked to Amazon Web Services, where users can source more processing power to speed up the process of getting the required results in a much shorter timeframe. The details pertaining on how the inputs are converted to the final function can be found in the website of Eureqa system (Eureqa 2015).

### *3.2.2 Encryption Key Generation*

The encryption function is then converted to an encryption algorithm to encrypt the data. The encryption process within the encryption engine starts by converting the input data stream into binary data stream. For example, an input 'Private Information encryption' results in the following binary stream:

```
01010000011100100110100101110110011000010111010
01100101001000000100100101101110011001100110111
01110010011011010110000101110100011010010110111
01101110001000000110010101101110011000110111001
01111001011100000111010001101001011011101101110
```

The encryption algorithm is then used to insert a large number of randomly generated values. These values are floating point integers normalized between 0 and 1. This process generates an array consisting of 64 bit stream encryption keys. One key is randomly selected and stored. The process has to be repeated four times in order to produce the required 240 bit stream key. Each key bit stream is generated from a totally new array to further increase the randomness of the generated keys. Figure 4 and 5 below depicts the floating point values with their corresponding encryption keys. A key is selected from each run to further confuse the process of brute-forcing keys and strengthen the encryption of user credential and behavioral data used to authenticate users. The four keys which are randomly selected from each generation run are then converted into binary streams. They are as follows:

Key 1 = 1000000001101101001101111101101100101001101010011011111100100101
Key 2 = 1111111110011001011100111101000011001000111010001011001110000
Key 3 = 1000000001001101100101111100011111011001101100001000110010101111
Key 4 = 1000000000100110001011010011000010110111010001110011011010011100

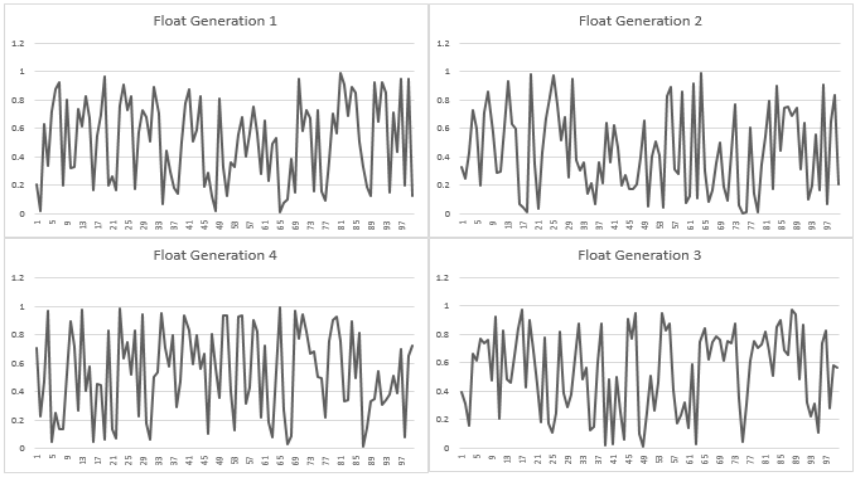

**Figure 4**: Distribution of Floating Point Values

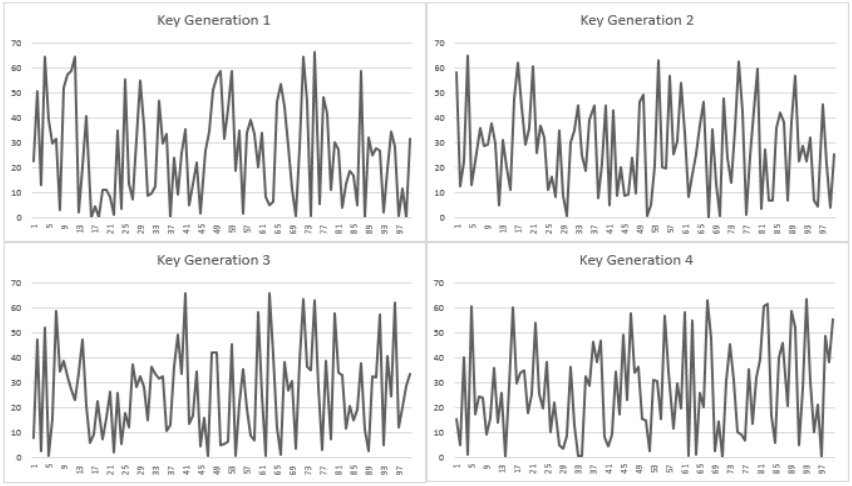


**Figure 5**: Distribution of Resultant Encryption Keys

### *3.2.3 Data Encryption*

The encryption process involves encrypting private user information such as passwords, profile numbers, geo-location coordinates, user activity or behavioral activity and other data that is collected for authentication purposes. The process of encryption is initiated by first transferring the user authentication data to the encryption engine. The above keys are then combined to form one 240 bit stream key that matches the binary stream of the plaintext. The key is then compared to the plaintext and trimmed until both bit streams are equal. Once they are equal, they are then XORed.

The authors acknowledge that an XOR operation is associated with its own vulnerabilities which make decryption very easy. In order to make up for the weakness of the XOR operation, this paper adds noise to the cipher-text in terms of blending it with the actual key. This is to make sure that reversing an XOR operation would not result in a one-to-one direct translation between plaintext and cipher-text. An ideal approach would be to introduce blocks and rounds more like 3DES and AES crypto systems. However, rounds and blocks are not considered in this paper. The result of our example is as follows:

000100000100010011110010100110111111010110100000101110100000010101110110100010
000011101010011011010000000101011101001101100101000010100101001001101001011100
001110001001101101100010010100100101001110010110001101100010111100010011010011
001101

Once this process is complete, key management is performed to store the key and increase the strength of the encryption. This is performed by splitting the binary key and cipher text binary stream into sets of 4 bits. Thereafter these sets of 4 bits are alternately selected and concatenated i.e. 4 cipher-text bits append 4 key bits append 4 cipher-text bits and so on until all sets of bits are used up. The final cipher-text containing the encryption key is the following bit stream.

000101000000000001000011010001101111100100101011100111101011101111110010101010010101101000001001011110110101111000000100101010101110011011011111000111010000110001101011010110010011111101101000100001100000010010100110111101001000010111001001111100100000001001001001000001000010100101101010110001011011110011100011100011101001110010111101011010000010010001010110001001010101011100110100100000110000100110011011000010010011011111100100011000001101010100101111001010011010011

## 3.3 Decryption Engine

In order to decrypt the stored information, the cipher-text is transferred into the decryption engine. Once sent to the decryption engine, it then reverses the encryption process in order to obtain the plain text. The decryption engine first retrieves the embedded encryption key. The cipher-text is split into sets of four bits. The first set of four bits represents a portion of the initial cipher-text and the next four represents the key. Each respective alternating set of bits is concatenated to form the full initial cipher-text and key. An XOR operation is then performed on the cipher-text and key and the result is the plaintext binary stream. This stream is then converted back into text according to its assigned ASCII values.

The next subsection discusses how the encrypted data is then stored by the proposed architecture.

## 3.4 Database Management Engine

Proper management of private user data is of utmost importance to any authentication system. The storage of user data in multiple locations helps increase its resiliency. Multiple storage locations are meant to mitigate the increasing theft of user credentials. A compromise on one storage location would not reveal much to criminals as this contains just part of the user data. Cyber criminals would have to compromise datasets in all storage locations to be able to at least combine the data into a full set before making an attempt to decrypt the data. A dataset from one storage location without the other sets is useless and cannot be decrypted without the rest of the datasets. Data retrieval and storage undergoes two processes as highlighted below. Data needs to be retrieved from at least two disparate databases in different locations. An instruction set is sent to the Database Management Engine. These instructions specify which data is to be retrieved and includes the profile number of the user.

The profile number is divided into equal parts based on the number of databases required to retrieve all data. The engine then retrieves each encrypted data set where the data header matches one of the parts of the profile number. One will be retrieved from each database. Once all datasets are obtained, they are concatenated such that the order of the dataset headers matches the user profile number. This dataset is then sent to the encryption engine where it is decrypted and the plaintext is returned. This process is essentially a reverse of the process of splitting data. Below is an example of how this is achieved.

Assuming a user, with profile number "***dogd7bttca***", requests a write operation for data containing the text "Private Information encryption". This information undergoes encryption and forms the cipher-text as shown in section 3.2. The resultant bit stream is then divided into partitions equal to the number of databases; in this case we use two. The user profile number is also divided into partitions equivalent to the number of databases. Each partition of data is assigned a unique header, consisting of a portion of the user profile, and is then written into the separate databases. Table 1 below depicts the datasets after going through the database management engine.

**Table 1**: Datasets after Database Engine processing

| Data Set | Data header | User Data |
|---|---|---|
| **1** | dogd7 | 000101000000000001000011010001101111100100101011100111101011110 111111001010101001010110100000100101111011010111100000010010101 010111001101101111100011101000011000110101101011001001111110110 100010000110000001001010011011110100100001011011100 |
| **2** | Bttca | 100111100100000000100100100100000100001010010110101011000101101 111001110001100111000111010011100101111010110100000100100010101 100010010101010111001101001001000001100001001100110110000100100 110111110010001100000110101010010111100101011010011. |

Table 2 and 3 below depicts the partial data as is written to the two different databases along with their associated sub-headers.

**Table 2:** Partial data in database one

| | | | | | |
|---|---|---|---|---|---|
| bttca | 0110010000000... | 0110010000000... | 0111010000010... | 0000010011110... | 0110010000000... |
| b8l10 | 0110010000000... | 0110010000000... | 0111010000010... | 0010010010100... | 0110010000000... |
| h5vdb | 0110010000000... | 0001001111111... | 01110100 | 0011010001000... | 0110010000000... |
| np4ff | 0110010000000... | 0110010000000... | 00000011 | 0010010010110... | 0110010000000... |
| 1qfgd | 0110010000000... | 0110010000000... | 01110100 | 0001010010010... | 0110010000000... |
| 2e7no | 0110010000000... | 0110010000000... | 0111010001010... | 0001010001000... | 0110010000000... |
| prnc1 | 0111010000010... | 0111010001110... | 0111010000010... | 0000010001000... | 0111010000010... |
| 8n915 | 0111010000010... | 0111010001110... | 0000001111101... | 0111010001100... | 0111010000010... |
| jhqh | 0111010000010... | 0111010001110... | 0111010000100... | 0011010001000... | 0111010000010... |
| va4qk | 0111010000010... | 0000001111101... | 0111010000010... | 0011010001110... | 0100001110001... |

**Table 3:** Partial data in database two

| | | | | | |
|---|---|---|---|---|---|
| dogd7 | 1001111101001... | 0111100010101... | 0111010000010... | 0110010111100... | 1010011101100... |
| 9kgos | 1001111110010... | 1010010111111... | 0000001111101... | 1111101100011... | 0110101101110... |
| r38iu | 0100001000011... | 0011110000110... | 01110100 | 1011110100000... | 1001010010101... |
| 8klgs | 0110000010100... | 1001011001110... | 00000011 | 1100100100011... | 0111101000010... |
| ku8kp | 1100101010100... | 0110100100100... | 00010000 | 0111001101011... | 1100000100010... |
| u4i75 | 1110100001111... | 1100001110101... | 0000001101010... | 1001110011001... | 1001010011001... |
| kldrc | 0011000001010... | 1010100111111... | 0111010001000... | 1010111100010... | 0101001100000... |
| nm590 | 1000101110011... | 0101011001100... | 1100111110101... | 0100001011101... | 0110000000000... |
| 8m41o | 1000101101100... | 0101011000110... | 0001111101100... | 0001010110100... | 0110000010111... |
| v7te5 | 1000101101010... | 0010000101110... | 0001001000010... | 0000011111011... | 1000111000000... |

The next section discusses the Steganography Engine.

## 3.5 Steganography Engine

The proposed architecture uses features such as emailing or SMS texting of OTP tokens over networks which are sometimes not even secured. This is where steganography comes into play with the usage of images embedded with hidden OTPs. The system is implemented in such a

manner that the OTPs expire after a short while and this requires timestamp information. Embedding the data within the image requires least significant bit (LSB) watermarking. This process involves first converting OTPs into a binary format and then hiding the binary stream within the first two bits of each pixel in the image. The following subsection discusses the process of embedding OTPs into and retrieval from the steganography image.

### *3.5.1Embed data onto the image*

To ensure uniqueness of the images sent to each user, a random image of size 640 by 320 is generated. This is done by randomly selecting colors for each pixel in the image to ensure that each user obtains a unique image. A call is then made to the OTP Generation Engine and a unique OTP is sent to the Steganography Engine. The time of issue is also recorded and is concatenated to the OTP string. The string containing the OTP and time of issue is then sent to the Encryption Engine where it is encrypted. The resulting cipher-text binary stream is then split into two sets of bits. The engine then iterates over each pixel within the generated image. The first two bits within each pixel of the image are replaced by a set of two bits from the split binary cipher-text. Once the end of the cipher-text is reached, the remaining first two bits in each pixel are set to empty binary values, we use 00. This image is then transferred to the user over a network. An illustration of the whole process is as follows:

### *3.5.2Embed and extract data to and from the image scenario*

Assuming a user just completed registration and submitted their data to the cloud. The cloud then responds by issuing a one-time authentication image containing the following information.

**3TlUbcUi8f 1444390509345**

This data undergoes encryption before the digital watermarking. The first portion of the information contains a ten character OTP and its timestamp. The timestamp ensures that the OTP is only valid for a short duration; in this case only 2 minutes. Each authentication image is randomly generated upon issue to ensure each user obtains a unique image.

**Figure 7**: A resultant steganography image

Once received by the user, this image is sent to the authentication system in combination with the user profile number as a means of validation. This part prevents an attacker using a man-in-the-middle type of an attack to try and eavesdrop for user credentials or the OTP over unsecured networks. This is to ensure that even if someone were to be able to intercept the communications they would not just land on plaintext user credentials. This ensures the confidentiality and integrity of the OTPs being exchanged between the system and the user.

After image has gone through watermarking, it is then sent over the network and eventually into the authentication system. The image is then transferred to the steganography engine. Each pixel from the image is extracted to determine its binary value. The first two bits of each pixel are extracted and concatenated. This forms the encrypted cipher-text containing the user's OTP and timestamp. This is sent to the Encryption Engine and for decryption. The plaintext is then trimmed to remove empty spaces. The plaintext is then returned to the user for authentication purposes. On correct entry of the plaintext OTP a user can then be authenticated. However, if the

user fails to enter the right OTP, the system captures such failed attempts and stores them on the user's historical data. Each login attempt is logged and stored. In the interest of space, the next section is shortened to provide a snip preview to some of the resultant GUIs that are presented to the user when different scenarios like the one of failed OTP login attempts arise.

## 3.6 Graphical User Interface (GUI)

User interaction is classified as risky behavior if the system detects a user behaving in a certain bad manner. This is flagged on a scale of 1 to 5 where 1 is safe and 5 is a high risk level. The calculation of the user's risk score is done by finding the average of their current risk score and the level of the offence being committed. Risky behavior cannot only be as a result of bad behavior, but it might also result from a user that is trying to log onto the system from multiple location or multiple devices which are new and have not yet been registered in the user's profile. Below we discuss some of these behaviors that may be classified as risky.

It must be noted though, that the diagrams below are the actual screenshots of our prototype implementation. They show the exact messages from our system that a user is presented with when medium and high risks are detected by the risk engine. In this scenario (real test of the system), during login, the risk engine analyses the user, Mr Xyz and detects that he and his context pose medium risk. The system prompts for a SurePhrase input as an extra layer of authentication. Notice that the user only has to fill in the missing characters of the SurePhrase i.e. those in blue. The system has been designed and built in such a way that if the risk is high, it presents the user with even fewer characters that are already filled in. In that case, a user would have to enter more characters of the SurePhrase. Hence, the level of detail of the SurePhrase also scales up or down depending on the risk posed. The user then makes several mistakes in entering the SurePhrase. This is now re-evaluated and detected as high risk by the risk engine. When the risk level is escalated to high risk, the user is sent an OTP to use as another extra layer of authentication on top of the SurePhrase. After failing to enter the right OTP, the system eventually kicks the user out.

The upper right corner of the GUI presents a gauge which highlights the current risk level in real-time. The gauge points towards the users current risk level, which is medium and high. The gauge shows the actual threat or risk posed by the user as they make an attempt to login to our system in real-time as the risk engine evaluates and classify each user's actions. In the next section we briefly discuss how this feature was evaluated to determine its usefulness. Despite some false positives or negatives, it appears to correctly classify a user's risk level. The next section provides a detailed section on how the proposed solution was evaluated for its effectiveness.

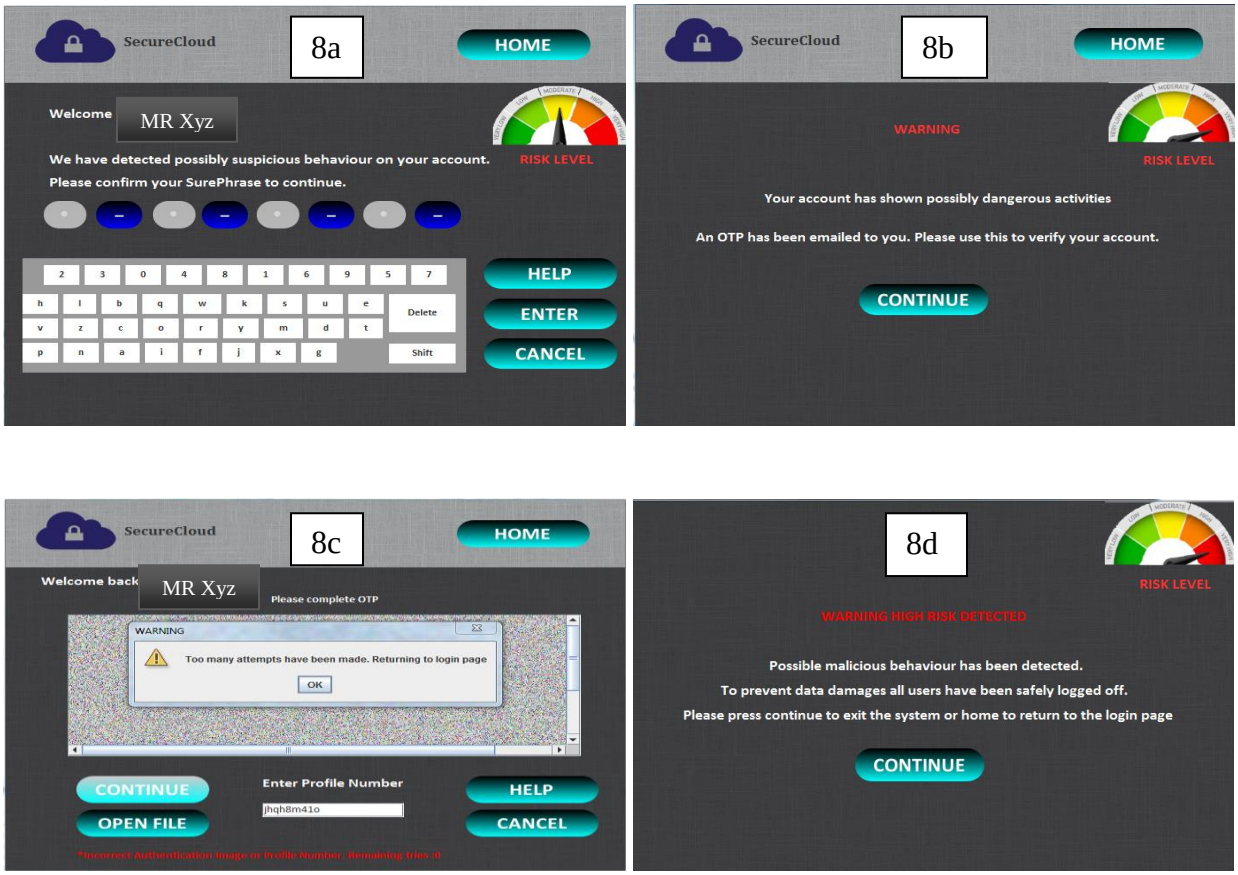


**Figure 8a, b, c, d**: Medium risk (SurePhrase) to high risk (OTP) to multiple failed OTP attempts to malicious behavior.

# 4. System Evaluation Results

The solution was put through a number of evaluation tests to determine its validity and appropriateness to securely authenticate users and handle their user credentials. It must be mentioned though, that the evaluation test were conducted on the prototype not necessarily on an operational cloud infrastructure. The solution was evaluated for usability, encryption and decryption response time, effectiveness of the steganography engine, strength of encryptors, and effectiveness of the risk engine.

## 4.1 Usability Evaluation

The solution was evaluated for ease of use in terms of ease of installation, user friendliness, level of response, features or functionality and mistakes or errors. The tests were carried out by a group of ten users, who after interacting with the system were asked to complete a questionnaire. The results on a scale of 0 – 5 (i.e. 1 – Exceptional; 2 – Good; 3 – Moderate; 4 – Minimal and 5 – Not at All) are as shown in the next figure below.

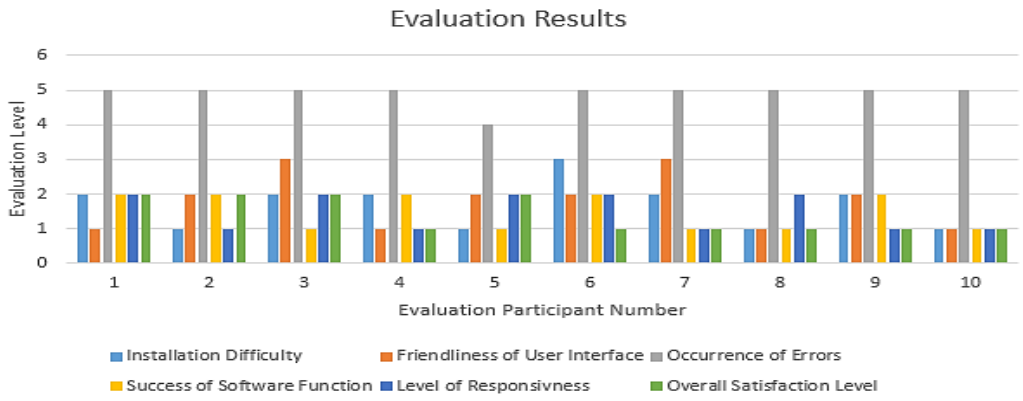


**Figure 9**: Graph of User Responses to Usability Evaluation

The results reflect that solution performed between exceptional to moderate (with just a few cases reported to be moderate). For example, user 3, 6 and 7 report '*friendliness of user interfaces'* and *'installation difficulty*' to be the only usability attributes that are ranked moderate level. Otherwise, all usability properties ranked between '*exceptional'* and '*good*'. For all users, there were no cases of errors or mistakes. It was also good to note that the respondents were now aware of the risk engine performing rapid threat evaluations on the background. Moreover, users did not have any challenges with handling the SurePhrase and OTP concepts. The overall user satisfaction on usability demonstrates the security and improved ease of use of our system.

### 4.2 Steganography

The solution was then evaluated on how it embeds and extracts data from watermarked images. A series of ten runs were conducted on ten different images, noting the response time on each case i.e. ten runs on embedding data from an image and ten runs on extracting data from an image. The figure below shows the results.

Furthermore, the solution was tested using a word processor (Microsoft NotePad) to determine if it could be compromised to extract the hidden data. It has been noted that attackers now use word processors to extract hidden data from images. The figure below shows the results of such an act. The hidden text could not be separated from the image. Hence, we deduced that the proposed steganography part of the solution is resistant to such attacks.

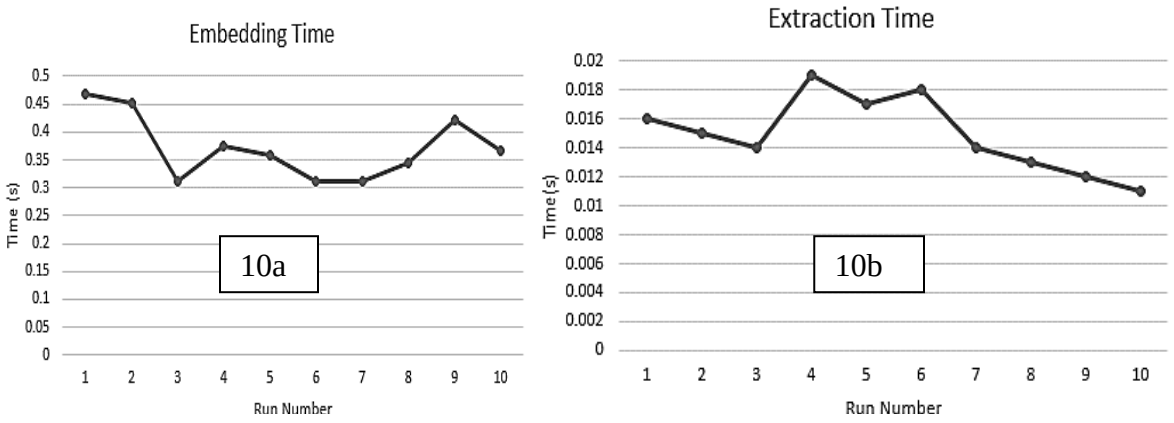


**Figure 10a, b**: Response Time to Embed and Extract Data from an Image over 10 runs

**Figure 11:** Result of The Image after it was opened using NotePad

### 4.3 Encryption

Most of the time was spent on the encryption feature as the entire solution hinges on it. The part was evaluated for its usefulness and applicability to preserve confidentiality of encrypted data. Drawing from the results previous work, the encryption was evaluated on a number of aspects; i.e. strength of encryption keys – uniform distribution test, response time of both encryption and decryption process, XOR cipher-text for frequency analysis and brute-force attacks, acceptable cycle length.

The evaluation tests started by proving that the resultant fitness function generated from the Eureqa system appropriately matches the input chaotic random noise. For the fitness function to be truly random, it requires the keys to follow a uniform distribution. A uniform distribution of the keys ensures that selecting each key from the generated set of keys is equal to one value. A set of 400 encryption keys were generated to test the fitness function. The results are shown in figure 12a below and reflect a uniform distribution. This is due to the fitness function producing rational numbers rounded off to 8 decimal places. This is a more efficient technique compared to using integers as demonstrated in the figure 12b below. This indicates that normalization of the points between zero and one does not result in a loss of generalization. Hence, we can deduce that the encryptors are also evenly distributed as the chaotic random noise data points.

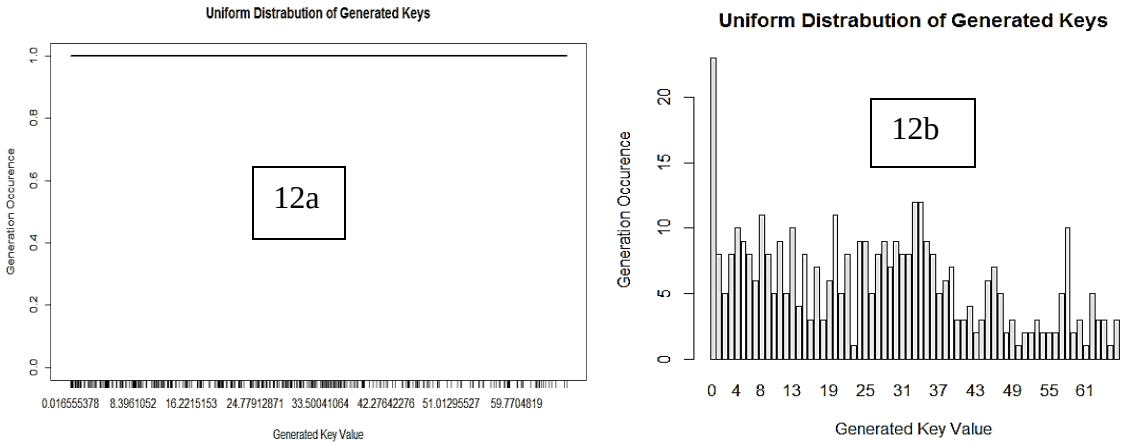


**Figure 12:** Graphs of Distribution of Float vs Integer Key Values

The solution was also evaluated on its encryption and decryption response time. However, we did not evaluate it against other crypto systems. We performed 11 runs to evaluate the time taken to encrypt a segment of random sets of data of 100 words in length for each run. The results (figure 13a) show that it takes less than a second to encrypt which indicates that our solution is highly suited to encrypt large datasets. We also performed 11 test runs on encrypted binary stream of 10496 bits for the decryption process. Again the decryption process took less than one second. Hence, our solution can be argued to perform well in encrypting and decrypting large datasets under a second.

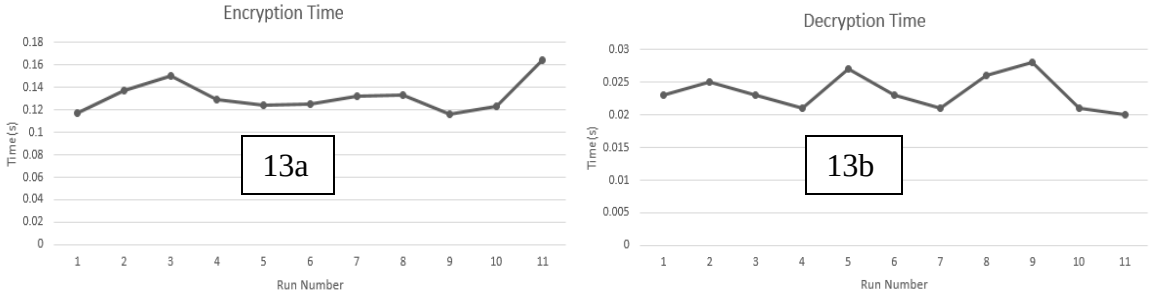


**Figure 13a, b**: Encryption and decryption response time for 11 runs

The strength of the cipher-text was evaluated using a frequency analysis and brute-force attacks. A binary key is split into segments of 8 bits and converted to its ASCII values. This covers the entire ASCII table values (128 possible values) not just 26 alphabets. To try and brute-force a possible cipher key on a text of length $n$ becomes $128^n$. Assuming we have a cipher of length 100; this means we have $128^{100}$ possible keys. This can only be computationally feasible if brute-forced on massive super computers for some significant time. Frequency attacks also become difficult as ASCII values do not map to recognizable alphabet characters. By incorporating a blend of the key and the cipher-text we can prevent frequency analysis attacks. This is mainly because the key adds noise to the cipher-text which further strengthens our decryption. Hence, brute-force attacks can be mitigated by the system and frequency analysis attacks are ineffective as the cipher-text does not directly represent the encrypted plaintext.

In order to continue strengthening the encryption key using more random uniform distribution, each key is randomly chosen from a new cycle of key generation. Each cycle generates a set of 100 keys from which only one is chosen. This is repeated until there final key is equal to the plaintext to be encrypted.

### 4.4 Risk-based

The risk engine was tested for its response time and for false positives and negatives. These were evaluated as the risk engine tries to correctly classify user's risky behavior and the risk they pose to the system. The idea here is to keep the response time low. At the same time the classification of users' behavior must be done on a fly but be always correct and avoid false positives and negatives. The accuracy of 10 evaluation tests is as follows. On average, the response time is under half a second. Only 7% cases of false positives or negatives were detected. The accuracy of the risk engine can further be improved by adding more classification rules and through the use of genetic programming. And as the system gets to be used more often then it will eventually self-learn to further reduce false positives or negatives.

Hence, we can conclude that the proposed solution is effective and efficient to secure the blind-spots of cloud computing in terms of strong multi-factor risk-based user authentication and secure handling of user credentials in use, transmission and storage. The next section concludes the paper and provides future work.

## 5. Conclusions

The proposed solution provides strong and risk-based multi-factor authentication that scales up and down based on the threat levels or risk posed on the system. It provides an end-to-end security of user credentials and other authentication data at rest, in use and in transit. This forms a key part of our contribution. As part of this contribution, this paper employs an innovative encryption algorithm that takes chaotic random noise as input to generate encryption iterators to help encrypt and secure user data which is then stored in multiple storage locations to increase resiliency. It also makes use of steganography to transport encrypted OTPs to the user. The proposed architecture makes a plausible attempt to reduce cyber-criminal activities that are normally caused by weak authentication or compromise to user credentials. Future work will delve into performance analysis of the proposed solution against others to determine its effectiveness and response time. For example, our encryption has not yet been compared to how it fares against other encryption algorithms like Triple-DES, AES 256 etc.

## References


AWS (2016) *Amazon EC2 Instance Types,* Amazon Web Services, Available online: https://aws.amazon.com/ec2/instance-types/, accessed: 20 January 2016.

Blackledge, J.M., S. Bezobrazov, P. Tobin (2013) *Cryptography using Evolutionary Computing,* ISSC 2013, Letterkenny Institute of Technology, Ireland.

Butler, B. (2014) *A Wakeup Call for the Cloud,* NetworkWorld, Available at: http://www.networkworld.com/article/2366862/iaas/a-wakeup-call-for-the-cloud.html, accessed: 01 March 2016.

CA Technologies (2014) *Authentication Strategy: Balancing Security and Convenience,* Available online: http://www.ca.com/content/dam/ca/us/files/ebook/intelligent-authentication-balancing-security-and-convenience.pdf, accessed: 08 March 2016.

Dlamini, M.T., H.S. Venter, J.H.P. Eloff (2015) *An Innovative Risk-based Authentication Mechanism for Closing the New Banking Vault*, Proceedings of the 3rd International Conference on Innovation and Entrepreneurship ICIE-2015. Durban, 19 – 20 March 2015.

EMC Corporation (2013) *RSA Risk-based Authentication: For RSA Authentication Manager 8.0*, Available online: https://www.emc.com/collateral/data-sheet/h11506-rsa-rba-ds.pdf, accessed: 08 March 2016.

Eureqa (2015) *A software tool for detecting equations and hidden mathematical relationships in your data,* Available online: http://creativemachines.cornell.edu/eureqa, accessed: 18 December 2015.

Goode, A. (2015) *How Risk-based Authentication can Improve The Authentication Experience*, Available online: https://blogs.rsa.com/risk-based-authentication-can-improve-authentication-experience/, accessed: 08 March 2016.

Heller, M. (2015) *Stolen credential are key to avoid breach detection,* Available online: http://searchsecurity.techtarget.com/news/4500253604/Stolen-credentials-are-key-to-avoiding-breach-detection, accessed: 08 March 2016.

Kennedy, D., S. Kanjee, D. Schraader, E. Ward, J.A. Nyangaya, C. Gibson, N. Sing, M. Alberts, J. Kock, T. Simon (2013) *Enemy at the gates: What executives need to know about the keys to the*

*organization's front door,* 2013 Deloitte & Touche, available online: http://deloitteblog.co.za/wp-content/uploads/2013/08/Enemy-at-the-gates.pdf, accessed: 10 October 2014.

Kitten, T. (2016) *More Hackers Relying on Compromised Credentials,* Available online: http://www.bankinfosecurity.com/interviews/more-hackers-relying-on-compromised-credentials-i-3107?rf=2016-03-08-eb&mkt_tok=3RkMMJWWfF9wsRonvqzMd%2B%2FhmjTEU5z14%2BgqWKaxlMI%2F0ER3fOvrPUfGjI4AT8VhNa%2BTFAwTG5toziV8R7DALc16wtwQWRLl, accessed: 08 March 2016.

Law, T. (2014) *Dropbox User Credentials Stolen: A Reminder to Increase Awareness in House,* Symantec.connect, Symantec Thought Leadership, Available online: http://www.symantec.com/connect/blogs/dropbox-user-credentials-stolen-reminder-increase-awareness-house, accessed: 08 March 2016.

Mathers, B. (2016) *Getting Started with the Azure Multi-Factor Authentication Server,* Microsoft Azure, Microsoft, Available online: https://azure.microsoft.com/en-us/documentation/articles/multi-factor-authentication-get-started-server/?cdn=disable, accessed: 20 January 2016

Mimoso, M. (2014) *Hacker Put Hosting Service Code Spaces out of Business,* ThreatPost, available at: https://threatpost.com/hacker-puts-hosting-service-code-spaces-out-of-business/106761/, accessed: 01 March 2016.

Random.org (2015). *Random.org: True Random Number Service,* Available online: https://www.random.org/, accessed: 18 December 2015.

Raphiri, T.V. M.T. Dlamini, H.S. Venter (2015) *Strong Authentication: Closing the Front Door to Prevent Unauthorized Access to Cloud Resources*, Proceedings of the 10th International Conference on Cyber Warfare and Security ICCWS 2015, Kruger National Park, South Africa, 24 – 25 March 2015.

RightScale (2015) *State of the Cloud Report: Central IT is Taking the Lead to Broker Cloud Services to the Enterprise*", RightScale, Available online:

http://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf, accessed: 18 December 2015.

Rossi, B (2014) *Catastrophe in the cloud: What the AWS hacks mean for cloud providers: Recent attacks on AWS EC2 should act as a catalyst for change.* Information Age: Insight and analysis for IT leaders. Available at: http://www.information-age.com/technology/cloud-and-virtualisation/123458406/catastrophe-cloud-what-aws-hacks-mean-cloud-providers, accessed: 01 March 2016.

RSA (2015) *RSA Adaptive Authentication: Balanced Security and Usability for Strong Authentication,* RSA, Available online: http://www.concordantfinancial.com/security/rsa-fraud-prevention/rsa-adaptive-authentication.htm, accessed 20 January 2016.

Saif, I., Siebenaler, R. and Mapgaonkar (2013) *Risk-based Authentication: A Primer,* Deloitte, Available online: http://deloitte.wsj.com/cio/2013/10/30/risk-based-authentication-a-primer/, accessed: 08 March 2016.

Strom, D. (2015) *Taking a Risk-Based Approach to Fraud Prevention,* Vasco Whitepaper, Vasco, Available online: https://www.vasco.com/Images/Taking%20a%20Risk-based%20approach%20to%20Fraud%20VASCO%20WP%20v4.pdf, accessed: 20 January 2016.

Venezia, P. (2014) *Murder in the Amazon cloud,* InforWorld, available at: http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html, accessed: 01 March 2016.

Webroot (2014). Secure Mobile Banking: Protecting Your Customers and Your Bottom Line, Available online: http://f6ce14d4647f05e937f4-4d6abce208e5e17c2085b466b98c2083.r3.cf1.rackcdn.com/secure-mobile-banking-protecting-your- customers-your-bottom-line-pdf-4-w-1141.pdf, accessed: 10 October 2014.