Article can be accessed online at http: //www.publishingindia.com

Design, Implementation and Performance Analysis of a Distributed Key Encryption System Deployed Within a Public Cloud

Erich Rice*, Dennis Guster, Laura Lebentritt*****

Abstract

The advent of cloud computing has decreased the cost of enterprise level system design and implementation, while at the same time increasing the need for a sound and secure strategy for security. While the use of encryption algorithms continues to be the main line of defense in performing secure data transmissions, the use of a Cloud Computing environment offers both advantages and disadvantages in the encryption process. Though the new series of encryption algorithms are quite robust, they require a "key" to make their use unique for an individual session, thus if the key is compromised then the underlying encryption algorithm can be broken. In a classically designed system, the entire cryptographic key is contained on one node within the network, if this node is compromised even though robustly protected, then the entire network would be at risk.

The flip side to the potential breaking in dilemma outlined above is perhaps an even scarier option, one in which the node on which the key is kept is corrupted either through malicious intent, unintended mishap, or simple system failure. This scenario opens up the possibility that the key is unrecoverable, in which case the data that has been encrypted with the cryptographic key may be rendered unrecoverable as well.

This paper analyzed how a distributed key system, broken up over varying numbers of multiple nodal instances, and distributed across the Amazon Web Services (AWS) Cloud reacted and performed their intended task of authenticating a web service. Different numbers of nodes were evaluated and timing was recorded to assure that latency did not exceed the specified level of three seconds, where e-commerce or other Web based activities would be negatively impacted. As additional numbers of nodes were added to the system the latency increased. Also, as nodes were taken offline the latency also increased, as there were fewer options of key nodes that could reply to the system to replicate the key. And finally, when more than the required nodes were taken offline the system failed to authenticate the Client.

Keyword : Distributed Systems, Cloud Computing, Key Management, Fault Tolerance

1. Introduction

1.1. Identifying the Problem and Background

Currently we reside in an era of constant threats from advanced persistent threat (APT) perpetrators on the Internet, the need for more secure and robust transmission of data has become ever more paramount (Raiu, 2012). The recent attacks on the likes of the Federal Reserve, New York Times, Target and even Apple, as well as the purported snooping by the NSA from the Snowden leaks, has shown us that the need for increased security in our ever more connected world will only gain greater importance (Hentoff, 2013). While much of the data currently sent over the Internet is clear and unencrypted through http (hypertext transfer protocol), the newer forms of encryption algorithms available today are very capable of hiding the data from prying eyes, and

* Saint Cloud State University, United States, Email: rier1201@stcloudstate.edu
** Saint Cloud State University, United States. Email: dcguster@stcloudstate.edu
*** Saint Cloud State University, United States. Email: lauraleigh4097@att.net

are therefore considered quite robust such as https (hypertext transfer protocol secure) (Abdalla, Bellare, & Neven, 2010). Thus the problem then becomes how the encryption is utilized, and whether it is done in as secure a manner as possible. This can most often be by a means of using a cryptographic keyto make the session as unique as possible and thereby as secure as possible. An example of a secure transmission protocol would be Secure Shell (SSH), which is a cryptographic network protocol used for secure data transmission. SSH creates a secure channel over an insecure network connection using public-key cryptography (Ylonen & Lonvick, 2006). In such a case as this where a secure transmission protocol is used then the problem becomes if the key is compromised, the underlying data is put in peril if the encryption used to encrypt the data can be broken and the data can be unlocked.

The threat of data loss has only increased as the use of cloud computing has gained traction, along with more disparate systems and networks. New threats like data-loss or leakage, non secure application programming interfaces (API), account hijacking through phishing or other social engineering means, and other insider threats (Cloud Security Alliance, 2013). Current algorithms being utilized are very robust, however, they are also widely known and thus there is a great deal of information on how they are structured and perform, thereby making them potentially vulnerable to attack through a simple web search of their vulnerabilities. Although this may be bad news for current algorithms being utilized, it is much worse for older algorithms. This creates the impetus for one to keep their encryption strategies up to date.The knowledge garnered from such a search could give a potential hacker the ability to crack the encryption utilizing the information that is often commonly available via the Internet (Herong, 2013). The fact that there is so much information readily available to the public on how the encryption algorithms work and how exploits have been found and how they operate has made many of the older algorithms too vulnerable to utilize as they are no longer a secure form of transmission.

Another factor adding to the increased vulnerability of encryption algorithms is the constantly increasing nature of computing power through faster CPU speed, and larger and larger distributed clusters of computers (Moore's Law, 2014). This can be seen both on a CPU level, as with Moore's Law which states that transistors or integrated circuits double roughly every two years, (Moore's Law, 2014) and also with the growing power and availability of computing clusters or extremely high powered computers (Goodin, 2012, Moore's Law, 2014). Both of these factors have made the necessity of keeping the cryptographic key used by the encryption algorithm extremely secure and not tampered with. Thus, one needs to be prepared to combat numerous attack vectors against the encryption methodology being utilized; including brute force attacks, social engineering, side channel attacks and the compromising the key attack. Of course, compromising the key offers the quickest and easiest solution to compromising the encrypted data as once the key is gained the protection provided by the encryption algorithm is lost. And therefore it is imperative to defend the key at all costs.

A potential methodology that can be used to protect the key is to control the key derivation as through the use of a random number generator as seen in (Barker & Roginsky, 2012) and further keeping it as secure and secret as possible once it is created. A method that is commonly used is the concept of key agreement protocols which is discussed in (Barker & Roginsky, 2012) and is similar to a public-key infrastructure and uses both the client and server to derive the shared key that they will use. The strategy increases the security of the key by making it much more difficult to corrupt it through a key guessing strategy as both the client and the server would need to be compromised to attain the entire key structure (Chalkias, Baldimitsi, Hristu-Varsakelis, & Stephanides, 2008). Although various software companies have proposed options for effectively managing the cryptographic keys at the enterprise level (Mearian, 2009) and recommendations from government agencies such as NIST exist about generating them (Barker & Roginsky, 2012) thus far splitting the key within a cloud-computing environment has not been widely researched. Further it appears that this strategy has not been implemented commercially at this time, at least as what can be discerned from published sources. Therefore, from the literature it appears that the most popular commercially available option is using a third party entity as an intermediary between the client and server, which in turn acts as a key translation center and evaluates the method of encryption (Boyd & Mathuria, 1998). This is somewhat surprising because as mentioned earlier a non-secure interface or API is one of the greatest threats to cloud infrastructures (Cloud Security Alliance, 2013).

An evaluation of the different methods that have either been proposed (Mearian, 2009) or implemented (Ylonen & Lonvick, 2006; Mearian, 2009) reveals a common theme. Described within that theme there are two salient points: the first is that if the key is compromised then the encrypted data can be utilized by a malicious intruder for their own purposes; and second if the key is destroyed by the malicious intruder then the encrypted data is lost and can no longer be used by the rightful owner (Redman, et al., 2013). A system administrator of an enterprise's architecture could look upon the second scenario as the more frightening of the two possibilities, because for instance a hacker breaking in and stealing credit card numbers could lead to adverse publicity and inconvenience to its customers.

As the importance of data has become ever more critical to commercial enterprises, such situations need to be planned for and potential eventualities dealt with. A situation of this type emphasizes the need for redundancy in the cryptographic key, and how a distributed method of dealing with the cryptographic key dilution could be dealt with. However, there is a trade-off in that the more keys or parts of keys create a higher probability that the encrypted data can be compromised as there are more potential targets to attack. This situation then leads to a trade-off between redundancy level through multiple key stores and key vulnerability as to the increased number of attack points, and thereby to the overall security of the system.

So therefore, as it has been pointed out the cryptographic key is the crucial part of the encryption strategy of any organization. And also, the key length needs to be evaluated to ensure an adequate level of security robustness is attained to meet the requirements that have been set forth in the security policy. A widely implemented solution is to store encryption keys and passwords in escrow with a secure third party who is delegated both the responsibility and liability for their storage (Moore, 2005). The utilization of a third-party storage concept does redirect the security responsibility by placing the key on an external host. However, it may not absolve the parent organization from all harm if a breach occurs since that host still stores the entire key and therefore would simply become a new and very tempting target to a malicious intruder to the network. Given the vulnerabilities of storing an entire cryptographic key on a single host, even a well protected one; it is thus a paramount concern to have an effective key management plan in place. The plan would need to be consistent across the entire enterprise and take into account all aspects of the wide area network (WAN) and local area network (LAN) connections that it would encounter. For these reasons, effective key management is the lynchpin to successful use of encryption in an enterprise level cloud computing environment and further research is needed in this area especially related to key storage strategies.

For the purposes of this current test situation the decision was made to build upon the system that was developed in Redman, et al. (2013) but to expand its scope out to a full cloud computing implementation. Although there were options within the authors' private cloud test-bed, it was decided to utilize the Amazon Web Service (AWS) cloud as it might be more replicable for other organizations or individuals that might wish to replicate the system or expand upon it. Testing was to be accomplished by setting up the various portions of the cryptographic key of the system on EC2 Amazon micro nodes; the tests were developed to ascertain the system latency as additional nodes were added to the system. The tests were set up with varying degrees of minimum and maximum nodal support in the base line test a minimum of three nodes were needed to authenticate out of a maximum pool of six potential nodes. In the last test there was an eight node minimum requirement of a maximum eight node system, thus all the nodes within the system were required to send back their portion of the cryptographic key prior to the authentication occurring. The various times for authentication were recorded and then plotted so as to determine how much latency was created by the addition of more nodes to the system. Also, testing was done to determine whether the expected redundancy of the system was acting as expected, and that if too many nodes were down then the system would fail to perform the authentication.

1.2. Research Questions

To provide structure to the paper a series of four research questions were devised. Specifically, there were designed to explore the effectiveness of a distributed encryption system that has been set up on varying numbers of nodes on the AWS computing cloud. To provide a realistic assessment of the design, the testing took place at a variety of times during the day and night to see if that

had any effect on the latency of the system to perform its authentication process and unlock the data store. By conducting a number of experiments utilizing various numbers of nodes required to unlock the data store, it also provided pertinent timing information on the system. More specifically, the following questions were evaluated:

1. How does the different (n-t) nodal logic increase or decrease the overall authentication time of the distributed encryption system?
2. How does the time of testing during the day or night effect the system, are there any heavy use times that negatively effect system performance?
3. How is the nodal redundancy affected when nodes are purposely taken offline? And does it effect the overall authentication time?
4. And finally, if enough nodes are taken down that the nodes down are greater than (t) given the (n-t) logic, will the system fail to authenticate?

In the course of answering these questions a review of the testing data will be provided and thoroughly analyzed. It is hoped that this research will have transferability to other distributed encryption system testing and potential uses beyond the ones used in the experiments contained in this paper, and perhaps relate to mobile device authentication and security as well. As this project progressed it became evident that the distributed encryption system could also be used to look at general cloud computing performance as well. Thus, testing has also been performed to ascertain the latencies of the system as it was run on both the AWS cloud and also on a public cloud provided by Digital Ocean.

1.3 Overview of the Design, Algorithm and Testing Environment

Server Operating System Selection. The server operating system selected for this project is a Linux based operating system, Ubuntu Linux 12.04 64 bit Edition was chosen to run on the various nodal instances. It was selected because it is open source software and offers a high degree of flexibility. It also offers high performance capabilities due to its low overhead and optimized code, as well as increased security.

Nodal Structure. The choice of cloud based nodes to perform the testing of the distributed encryption system were determined by two main factors, first the geographic distance between them and second the cost to setup and implement them. In both cases the optimal design seemed to point to the AWS elastic compute cloud or (EC2) micro nodes, given their high performance and reliability as well as the reasonable pricing structure (AWS Amazon, 2013). It was hoped that all testing on the system could be accomplished for around $300 to $400 dollars, which would be much cheaper than setting up a dedicated WAN or wide area network for the testing purposes. The AWS also allows flexibility in choosing the geographic sites for the nodes that are utilized. For the test purposes nodes were selected from the US East data center located in northern Virginia and the US West data center located in northern California (AWS Amazon, 2013). Thus there was a wide geographic disparity between the various nodes in the system that would authenticate to unlock the data store, replicating a typical real world scenario of an enterprise level cloud infrastructure.

Determining the Number of Nodes. Although it would be interesting to test the performance of the distributed encryption system on some fairly large clusters of nodes, for the purpose of this project it was determined to keep it fairly simple and limit the maximum number of nodes with portions of the key to eight. It was thought that this would provide a large enough set of possibilities to test the basic system and see if there was much difference in the latencies as the number of nodes needed to replicate the entire key was increased. Obviously, if the distributed encryption system were to be put into a production system on an enterprise level of operations the number of nodes being used would undoubtedly increase. However, as the scope of this paper is geared more towards determining the efficacy of the system in general on a public cloud, this relatively small number of nodal instances should be sufficient.

2. Methodology

To add rigor to the paper the four research questions stated earlier have been modified to provide four null hypotheses, which can be tested through experimentation on the system that was implemented on the AWS.

Hypothesis 1. The number of nodes defined by the system parameters has no effect on the overall latency of the transmissions and does not affect the authentication process.

Hypothesis 2. The time of day of a test on the distributed encryption system will have no impact on how long it takes for the system to authenticate, as the amount of data being sent is small.

Hypothesis 3. The time taken for the distributed encryption system to authenticate will not be affected by taking nodes offline.

Hypothesis 4. If the number of nodes down exceeds the value of (t) for any given test the system will not be affected and will still authenticate and unlock the data store.

In order to collect data to test these hypotheses the distributed encryption system test bed described earlier was devised so that the various nodal logics could be tested on the AWS cloud, the tests were roughly the same as in Redman et al. (2013) except that these were conducted in a live public cloud computing environment. A basic web based front end GUI (Graphical User Interface) was developed to perform the testing. Ten thousand instances of each of the eight tests were performed for a grand total of 80,000 test instances. The tests were performed at varying times of the day and night to provide a relatively good cross section of timing data for the latency of the system to perform the authentication process. As each attempt was run, it was necessary to unlock the data in the data store, and in the test situation the word "hello!" would appear on the screen if the test were successfully run and the data store was unlocked. If a problem occurred or the requisite number of nodes for a given test failed to communicate back to the data store then the word "Error" would appear on the screen as the data had failed to be unlocked. The following figure shows the web based GUI with the eight different tests that were performed, clicking on the button next to each of the tests would run the test and timing output would be displayed to the screen.

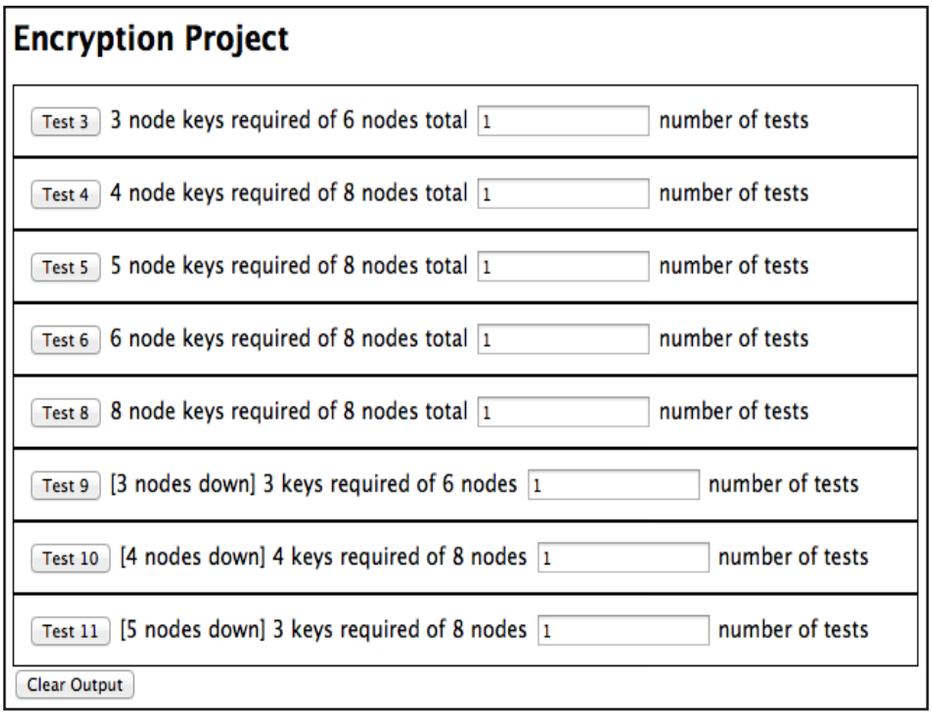

Fig. 1. Web Based Testing GUI (Graphical User Interface)

2.1 Randomness of the Seed Number

The randomness of the seed number for generating a cryptographic key is an important consideration to take into account when looking at the overall robustness of the encryption itself (Eastlake, 1994). As Redman et al. (2013) pointed out, perhaps the best method to utilize when creating a truly random seed number would be through the use of a quantum number generator. As was shown in their testing the results showed that there was a slightly more random set of numbers that were generated over the course of ten thousand examples (Redman et

al., 2013), however, if given a great enough entropy in the operating system the current software based number generators should provide a random enough set for utilization (Eastlake, 1994).

Thus, for the purposes of this testing it was decided to use the Javascript pseudo random number generator that was used in the Redman et al. (2013) testing as it was far easier to implement in the AWS cloud environment. Also, the main purpose of this paper is to look at the latency of the system once it is operationalized in a public cloud computing environment. The quantum number generator is a physical device that needs to be installed on a physical server, and as such it would be impossible to utilize on the AWS cloud, because physical access to the servers was not available, only remote console access. As this testing is more akin to a beta type system, the pseudo random number generator is sufficient, however if the system were ever deployed in an enterprise production environment the use of a quantum number generator would be the preferred method to use as it would likely enhance the randomness of the seed number and make the overall security of the system that much greater.

2.2 Splitting the Cryptographic Key

The main advantage of a distributed encryption system such as the one that has been proposed in this paper is twofold: first, that the cryptographic key as it is split up across multiple nodes is much more difficult for a hacker to gain access to and second, that if the key server were corrupted and it held the entire key on it then the data it protected might be rendered unrecoverable. If the purported benefits are accepted to be true, then the next question that needs to be answered is how the portions of the key should be split up across the various nodes of the system?

In Sanders and Beals (2008) various methods were covered that could help to provide a more secure method of encryption than currently provided by the public key encryption (PKE) architectures commonly in use, such as the RSA algorithm (n.d.). Since PKE methods use two types of keys, a public key known to all and a private key known only to the recipient of the transmission, the security provided by the PKE is based on the difficulty in guessing the key itself. However, as Sanders and Beals (2008) point out there have been vulnerabilities that have been proven to exist in even the most robust encryption algorithms, such as RSA, especially given the potential use of quantum computers and the use of cryptanalysis tools such as Peter Shor's quantum algorithm for integer factorization (Shor's algorithm. (n.d.)).

In Redman et al. (2013) the cryptographic key was split up in such a way that any 3 of the 6 nodes of the system could be used to replicate the key. The cryptographic key in question was a 24-byte key or 192 bits, and utilized AES or Advanced Encryption Standard encryption, which is a very robust encryption method, developed by the United States federal government. Within the U.S. federal government structure AES encryption is a valid form for use up to "Top Secret" security classifications, and when AES encryption is used with a 192 bit key it would take roughly $2^{189.7}$ operations to try and brute force recover the key (Hathaway, 2003). In Table 1 below you can see how the data for the cryptographic key was striped across the various nodes with an X marking the missing data, in this case there were 6 nodes for the system, any three of the nodes could be chosen to replicate the entire 24 byte key.

Table 1: Splitting the Key Across 6 Nodes (X = Missing Bytes)

Node	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	x	x	x	x	x											x						x		x
2	x					x	x	x	x							x	x				x			
3		x				x				x	x	x					x	x					x	
4			x				x			x			x	x				x	x			x		
5				x				x			x		x		x				x	x	x			
6					x				x			x		x	x					x			x	x

In the example shown in Table 1 above a great deal of time and effort was needed to properly stripe the cryptographic key across the various nodes in such a way that any three of them could allow for the reproduction of the total key (Redman et al., 2013). If care was not taken at this step it might invalidate the ability to replicate the key from the requisite number of nodes, in the example above it was a 6-3 logic, thus any three of the six nodes on the system could be lost or compromised and the total cryptographic key could still be recovered. In Table 2, which follows on the next page, one can see how the actual data from the key has been striped across the six nodes and also the initiation vector and salt which was used when creating the key.

Table 2: Cryptographic Key (Red Font) with Initialization Vector (IV) and Salt

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*
D8	77	9B	57	4A	BB	43	34	5F	77	96	C3	42	19	42	79	47	2A	D6	E5	A1	B3	50	4F
IV= 99BA8C95FBB459DBE5E92C769FFC55BB																							
SALT=2B336181CE15142C																							
					BB	43	34	5F	77	96	C3	42	19	42		47	2A	D6	E5	A1		50	
	77	9B	57	4A					77	96	C3	42	19	42			2A	D6	E5		B3	50	4F
D8		9B	57	4A		43	34	5F				42	19	42	79			D6	E5	A1	B3		4F
D8	77		57	4A	BB		34	5F		96	C3			42	79	47			E5	A1		50	4F
D8	77	9B		4A	BB	43		5F	77		C3		19		79	47	2A				B3	50	4F
D8	77	9B	57		BB	43	34		77	96		42			79	47	2A	D6		A1	B3		

In Table 2 one can see the full representation of the striping of the key data across the six nodes of the test system. Each of the 24 bytes of data are broken up into two 4-bit hexadecimal characters, as there are 8 bits to one byte of data, thus one can see that in the first byte of the key data there is a D and an 8. The rows below the SALT, or the random data that is used as an additional input to a one-way function that hashes a password, are the six rows that correspond to the six nodes that are listed in Table 1 on the previous page. Thus, in the first and second node we can see that none of the "true key" data is listed and is just filled with random data. Whereas, in nodes three through six we can see that the "true key" data of D and 8 is actually listed, thus in those nodes the first byte of key data is actually housed. From this striping of the cryptographic key data it becomes clear that were a hacker able to compromise one of the nodes it would become far more difficult to discern which if any of the data related to the "true key" data or if it was just filler data that was meaningless.

This effect is what gives a distributed encryption system its greater advantage when it comes to the key security, even if one or more of the nodes within the system are corrupted or breached, there will still be a portion of the "true key" that is outside of the view of a hacker or other malicious intruder. Now, in this very basic example there is not a very large portion of the "true key" that is not available to an intruder. If one node were compromised they would have two thirds of the "true key" though they would not know what of the data was good and what was bad. If however two nodes were compromised then the amount of effort to replicate the key would be greatly reduced as evidenced by looking at nodes one and two in Table 2, as only the first and sixteenth bytes of data are missing. The effort to replicate the "true key" would still be greater than brute forcing against just those two bytes of data however, as only the data in bytes 10 through 15, 18 through 20, and 23 actually overlap between node one and two. Thus even with two nodes compromised a hacker would only have 10 bytes total of the "true key" completely figured out, it would still take time and effort to determine which of the remaining data housed in the two nodes was a part of the "true key" and which was just filler placed there to confuse and obfuscate.

2.3 Creating the Distributed Encryption System

The next step in the process was to actually develop the encryption system. Once the cryptographic key splitting

issues were solved it was much easier to determine the way in which the distributed key encryption system could be set up on a public based cloud computing site. The Amazon Web Service cloud was chosen for its ease of use and the ability to create a geographically diverse set of nodal instances to test the system under real world working conditions. The following example in Figure 2 shows a graphical example of how the distributed key encryption system works.

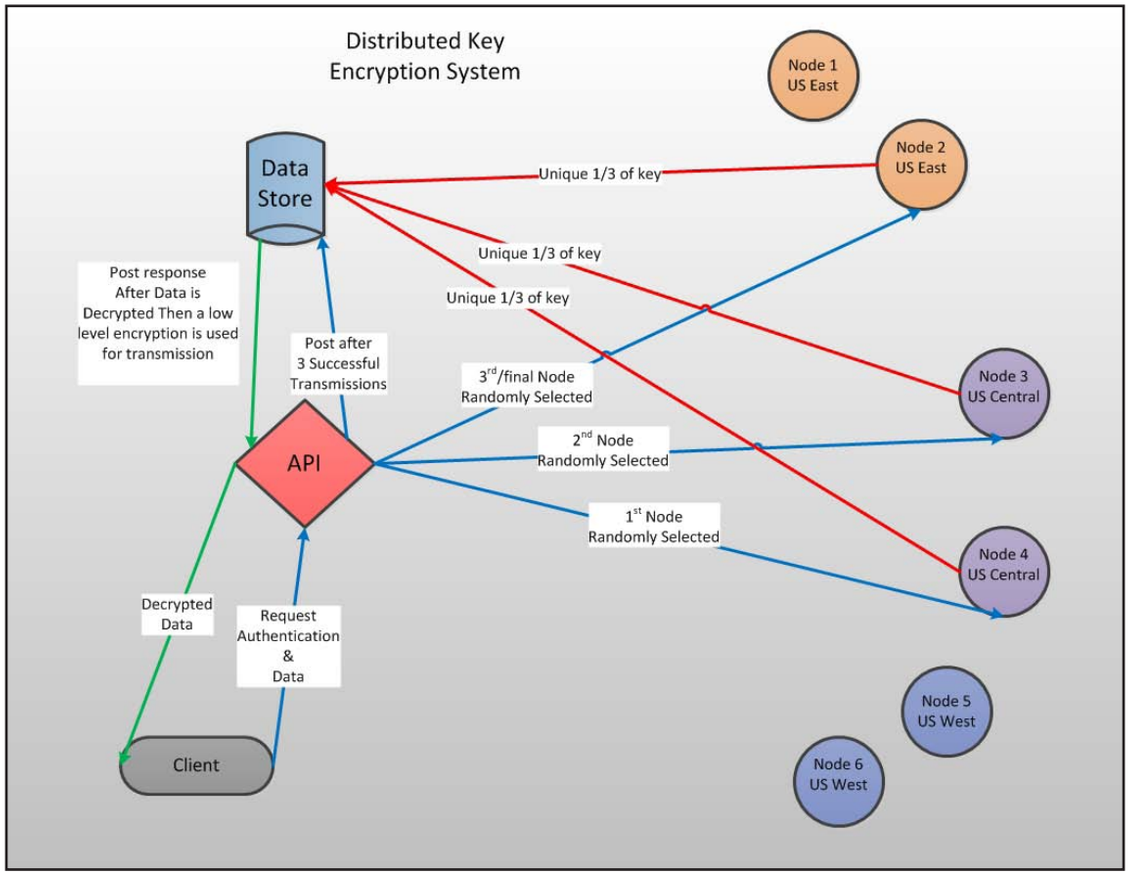

Fig. 2: Distributed Key Encryption System Example

As can be seen in Figure 2 a Client initiates the process by sending a request for authentication and transfer of the data held in the Data Store. The request is sent to the API or Application Programming Interface, which transmits the request on to the various nodes in the distributed encryption system. In the case of this example there are six nodes spread out across the U.S., however in the actual setup of the system on the AWS there were three nodes each housed in Northern California and Northern Virginia. In Figure 3 below one can see how the AWS data centers are spread out across the globe, the boxes on the east and west coast of the United States indicate the rough location of the physical locations of the servers that housed the EC2 virtualized nodal instances of the distributed encryption system.

Once the system receives the requisite number of responses, in a 6-3 system it would need to receive any three of the nodal responses, this would then unlock the Data Store and the client would be authenticated and the requested data would be transferred. Because the distributed encryption system was set up over a public wide area network (i.e., the Internet), there was also a need to encrypt the data transmission as it was sent out from the testing location in St. Cloud, MN to the various nodal instances which as mentioned earlier were located in Northern California and Northern Virginia. To accomplish this the authentication requests were sent using the HTTPS or Hypertext Transport Protocol Secure (or HTTP over SSL or TLS), which utilize the SSL (Secure Socket Layer) or TLS (Transport Layer Security) to encrypt the packets of data as they are sent out from the Client and the reply from the EC2 Server instances (McKinley, 2003). The use of HTTPS allows for a relatively secure transmission of data over an insecure network, which the Internet very much is.

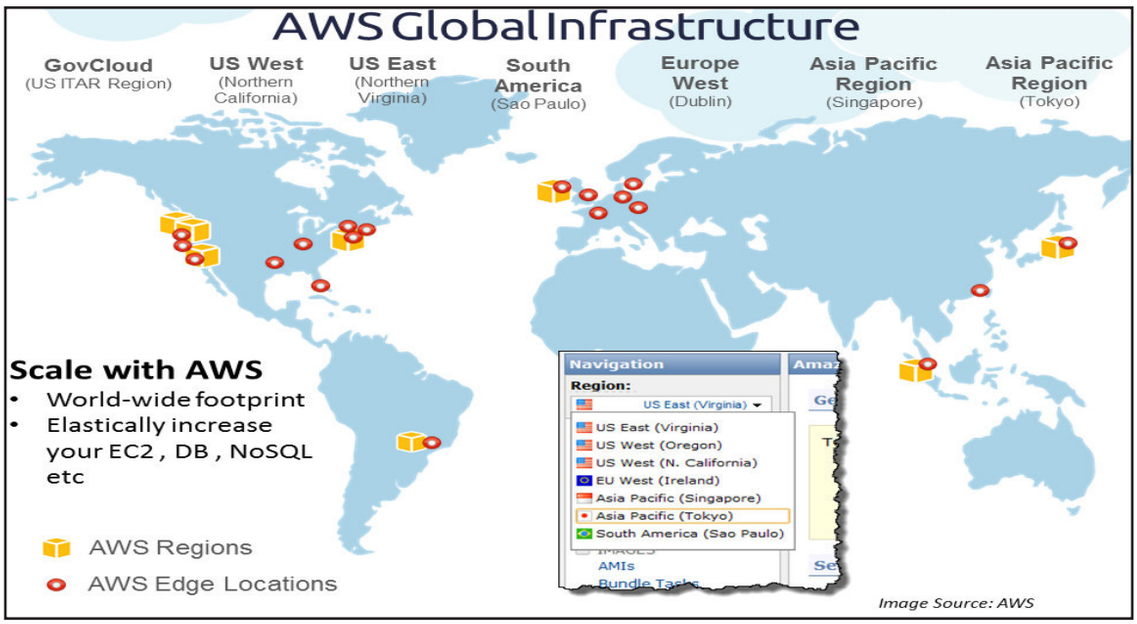

Fig. 3: Global Map of the Amazon Web Service Cloud

2.4 Distributed Key Encryption Process

Once the system was setup on the AWS and the nodal instances were brought online, then the system was ready to begin to be tested. The testing results for the series of tests that were performed will follow in the next section of this paper, however, to give the reader a more in depth understanding of how the distributed encryption system actually works in practice a thorough explanation of the process will be given here. As mentioned above in relation to Figure 2, a User of the system would initiate the process by loading a Client interface, which would allow the User to input a name and password to authenticate with the Node.js API. The User would then request the data that they wished to extract from the Data Store, a post is then sent to the API from the Client side interface using HTTPS and JSON (or Javascript Object Notation), which allows data to be transmitted between a server and a web-based application (JSON, 2015).

The second part of the process begins when the API receives the post message that was sent by the Client. The User is then authenticated with the API and a unique session id or GUID (Globally Unique Identifier) is created for the session. Then the API selects a random sequence of the distributed key nodes to broadcast the signal to, see Figure 2, the API sends out HTTPS posts to each of the key nodes and when the requisite number of nodes responds then the API stops sending out the requests. If for some reason the key nodes fail to respond to the Data Store within a prescribed amount of time then the API will begin to send out the requests again. Each time the API sends out an HTTPS post to one of the key nodes it includes a JSON object, which includes a unique session id (GUID), a requested key node/key section id, the requested Data Store to be queried, and the data that is sought from it. After the requisite number of replies from the key nodes has been received then the API sends an HTTPS post to the Data Store requesting that the requested data be decrypted and sent to the Client. The API then receives the Data Store response and decrypts a network transmission level decryption, and finally passes on the decrypted data to the Client who requested it.

Each of the key nodes stores a unique section of the cryptographic key, in the example from Figure 2 each of the six nodes in the system would have a portion of the "true key" housed on them, and it would take any three of the six nodes to replicate the complete "true key". When a key node receives the API request, the node looks at the JSON in the request and identifies which Data Store is required, and which Client session that the request is from. The key node then pulls the key section from the database and appends it to the JSON object, which was sent from the API in the original post. The key node then sends the JSON object through an HTTPS post request to the correct Data Store, and once the key node receives a successful response from the Data Store, the key node then sends a successful response to the API.

The Data Store in this situation is any server that is holding encrypted data that needs to be accessed for some purpose. When the Data Store receives the first key node HTTPS post request to unlock the data, the Data Store reads the JSON object that was sent with the request and it first authenticates with the key node, second it identifies which key node/key section that the request came from, third it identifies what data has been requested, and fourth it then records which session the request was sent from. The Data Store then creates a new unique session with all of the required nodal sections, in this case three nodes, will manage a specific data request. The Data Store then sends a successful response to each of the key nodes. For each of the remaining key node post requests, the Data Store will identify the session it belongs to and process each of the cryptographic key sections accordingly.

The Data Store then waits to receive a post request from the API, which is identified by the unique session id or GUID. Once it receives this the Data Store uses the, in this case, three cryptographic key sections to recreate the "true key", and then it is used to decrypt the data that is sought. Once the data is successfully decrypted the Data Store applies a basic level encryption to the data for network travel safety, and it travels by the HTTPS protocol. Finally, the Data Store responds back to the API with the requested data.

3. Results

The testing took place across several days. Day 1 accounted for 1500 instances of the system authenticating with a (n-t) nodal logic of 6-3. As the web-testing interface required a manual pressing of the "button" which corresponded to each test, it took roughly one hour to complete each of the 1,500 instance tests, and anywhere between 30 to 45 minutes to complete a 1,000 instance test. For each of the tests, there were six trials of the 1,500 instance tests and one of the 1,000 instance test to accrue a total of 10,000 instances for each of the eight configurations of the distributed encryption system.

Tests were performed in a rotating pattern, thus after the first 1,500 instances of the 6-3 nodal logic test was performed then the 8-4 test, then the 8-5, and then the 8-6 until all eight of the various types of tests were completed. As the results in Table 3 below show, although it was proposed in Hypothesis 1 that the overall number of nodes required to authenticate would not have an impact on overall authentication time, it was not born out in the data.

Table 3: Timing Results (No Nodes Down)

Test Type	*3 of 6 Nodes*	*4 of 8 Nodes*	*5 of 8 Nodes*	*6 of 8 Nodes*	*8 of 8 Nodes*
Min	290 ms	294 ms	297 ms	301 ms	305 ms
Max	2629 ms	1364 ms	9231 ms	9773 ms	11256 ms
Average	401.96 ms	401.90 ms	428.88 ms	449.86 ms	553.46 ms
Standard Deviation	122.54 ms	110.83 ms	376.15 ms	412.97 ms	873.92 ms

As can be seen in Table 3 the minimum times for each of the individual tests only varied by 15 ms (1 ms is equal to 1/1000 of a second) not a significant amount. If only looking at this result one could conclude that the first hypothesis was correct, however when looking at all the data it becomes clear that is not the case. If one thinks of the minimum time for each test as the "best case" scenario and the maximum as the "worst case" scenario we can see that as the number of required nodes increases so does the amount of time. In a perfect world without any interruptions in transmission of data the increase of time of the minimum time required to authenticate appears to be almost linear as more nodes are required. The time seems to increase by about 3 to 4 ms as each additional node is required to authenticate the system, although the jump from 6 required nodes to 8 was only an increase of 2 ms per node.

On the maximum or "worst case" scenario side of the scale we can see that as the number of nodes increases the potential for increasing wait times for the authentication process to occur can go up significantly. As evidenced by the jump in times for the 5, 6, and 8 required node tests, even though this may be explained away by network disturbances it is easy to see how as the required number of nodes increases the likelihood of system disruption can increase. Also, as evidenced by the rapid increase in the standard deviation of the tests, there was variation as the number of nodes required to transmit their portion of the key increased so that the authentication process to continue. The 6-3 and 8-4 test both require 50% of the nodes to transmit their data and they both have much lower standard deviations, whereas the 8-8 test which

requires all nodes on the system to respond and has a standard deviation eight times larger.

There were variations within the individual tests as well, as can be seen in Figure 4 below, depending on when the tests occurred the timing data could vary quite a bit. Two of the 1,500 instance test periods resulted in average test times of only 26 and 38 ms above the minimum time recorded for the 6-3 test. However, as was seen across the various tests the possibility of network disruption could cause a sharp increase in the time that the distributed encryption system takes to authenticate. For the 1,500 instances that were performed on day 2 at 10 pm the minimum time was 356 ms and the maximum time was 2,629 ms or roughly 2.6 seconds, which is a significant increase in the authentication time. The overall latency of the system was impacted significantly during this time as the times across all of the tests were significantly impacted during that time-frame

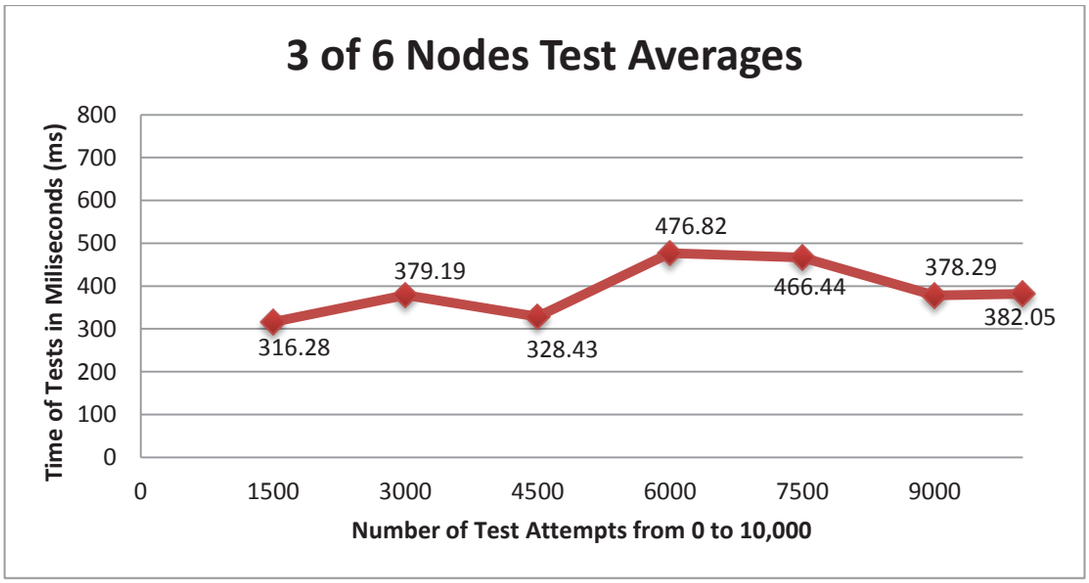

Fig. 4: 6-3 Node Test Results

In Table 4 one can see the variation of the timing results, the 4th test period (4,501 through 6,000 instances) really spiked, but also the 5th and 7th test periods as well. The maximum time spikes to 2,629 ms in the 4th test frame, and the average is 476.82 ms.

Table 4: Timing Results for 6-3 Node Test (No Node Down)

Tests	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Min	292 ms	353 ms	290 ms	356 ms	358 ms	354 ms	354 ms
Max	587 ms	879 ms	666 ms	2629 ms	1249 ms	734 ms	1164 ms
Average	316.28 ms	379.19 ms	328.43 ms	476.82 ms	466.44 ms	378.05 ms	382.05 ms

The results for the tests where nodes are taken off line is also eye opening, and fits with the narrative of the previous testing in that one can see a gradual increase in the time authentication takes place as more nodes are required, but also there is a sharp increase in time overall.

As can be seen in Table 5, the time to authenticate the system increases by a significant margin when nodes are purposely taken offline, although the system is still trying to communicate with them and it slows the overall process down

Table 5: Timing Results (Nodes Down)

Tests	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Min	292 ms	353 ms	290 ms	356 ms	358 ms	354 ms	354 ms
Max	587 ms	879 ms	666 ms	2629 ms	1249 ms	734 ms	1164 ms
Average	316.28 ms	379.19 ms	328.43 ms	476.82 ms	466.44 ms	378.05 ms	382.05 ms

Here one can see that the minimum times for authentication have increased by almost sixty percent across the board for the various types of tests, obviously not all of the previous tests were replicated with an equivalent node down test but the results for each is fairly similar. The 6-3 test for instance took 211 ms longer when three of the nodes were purposely taken offline, thus instead of the quickest three of the six nodes that replied, the API had to wait until the three nodes that were still active replied before unlocking the data store. Interestingly, the overall maximum times for the tests were lower overall, thus the impact that a wide area network connection (i.e., the Internet) can have on a distributed encryption system. In Table 6 below, it can be seen how the test data for the 6-3 test with 3 nodes down varies from the data in Table 4 when all nodes are active.

Table 6: Timing Results for 6-3 Node Test (3 Nodes Down)

Tests	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Min	292 ms	353 ms	290 ms	356 ms	358 ms	354 ms	354 ms
Max	587 ms	879 ms	666 ms	2629 ms	1249 ms	734 ms	1164 ms
Average	316.28 ms	379.19 ms	328.43 ms	476.82 ms	466.44 ms	378.05 ms	382.05 ms

As can be seen from the data in Tables 4 and 6 the minimum times for the distributed encryption system to authenticate have gone from the 292-358 ms range to the 501-534 ms range. However, the maximum time that the system took to authenticate was actually less in the nodes down test, undoubtedly this was caused by a more stable wide area network connection. Thus, it can be seen that the reliability of the distributed encryption system is a function of both the number of nodes available and also the stability of the network connections between them. If an enterprise was willing to pay for a dedicated private wide area network connection between the nodal instances then the importance of that portion of the equation would be much less, but given a public wide area network like the Internet it is undoubtedly one of the major factors in the system performance.

In Figure 5 below one can see the spike in the times of the five different tests without a nodal instance down, the largest spike in time occurring in the 8-8 node test where the maximum time was 11,256 ms.

In comparison, the graph of the nodes down tests are less drastic in their deviation from the average, however there is still a slight dip in the 4,500 through 7,500 instance tests and a slight rise in the 7,501 through 10,000 tests. In Figure 6 below it can be seen that the trend line of time to authenticate decreases from the initial 1,500 test instances and then spikes for the 8-4 and 8-5 with node down tests, while the 6-3 with nodes down test stayed roughly the same during that test period.

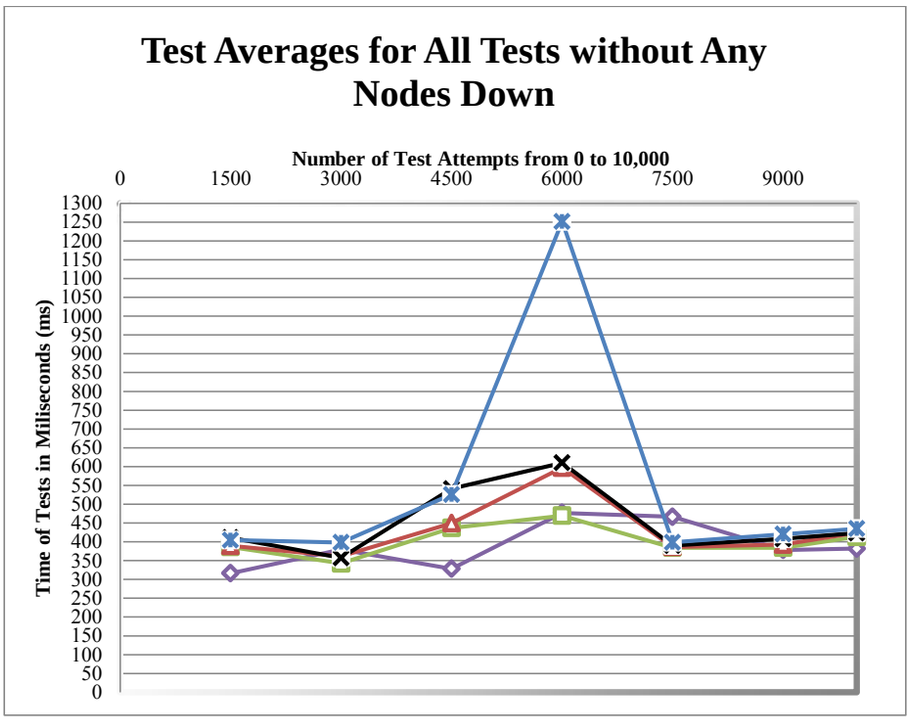

Fig. 5: All Tests without Nodes Down

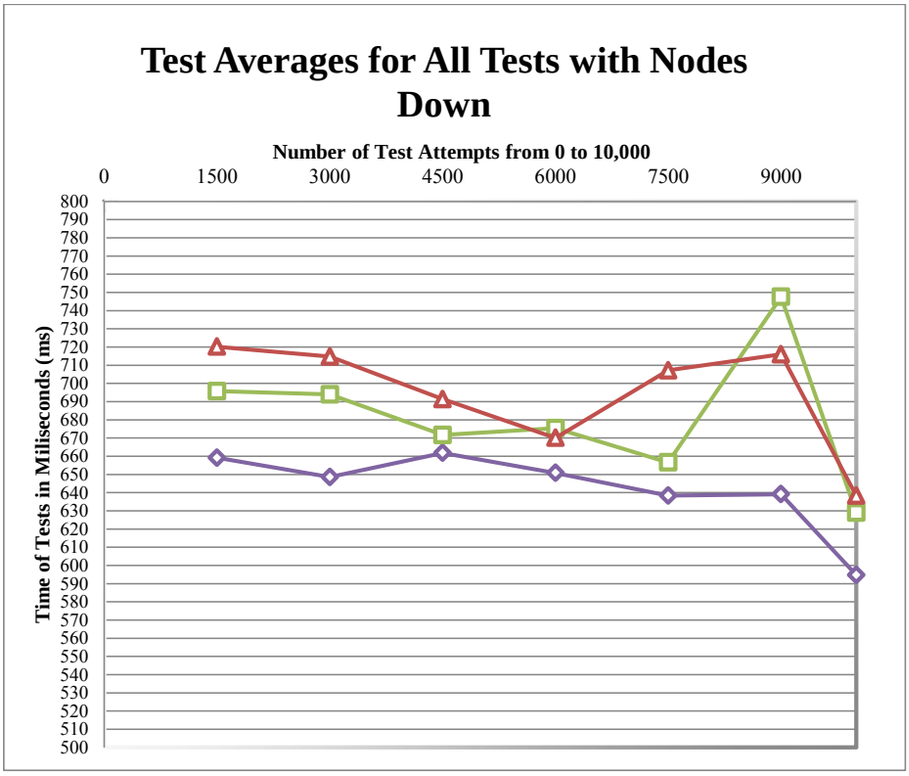

Fig. 6: All Tests with Nodes Down

4. Conclusions

The discussion will follow the results of the four null hypotheses.

Hypothesis 1. The number of nodes defined by the system parameters has no effect on the overall latency of the transmissions and does not affect the authentication process.

As can be seen from the data provided from the test situations the number of nodes did have an impact on the overall system authentication time, though it did not seem to be a truly linear progression as nodes were taken offline the time it took for the system to authenticate increased. The amount of time did not increase significantly when the system was working optimally, but when wide area network delays occurred the time to authenticate increased significantly.

Hypothesis 2. The time of day of a test on the distributed encryption system will have no impact on how long it takes for the system to authenticate as the amount of data being sent is small.

The time of day in and of itself did not seem to overly influence the results of the tests, however the testing was definitely thoroughly impacted by the result of wide area network delay which occurred during distinct time periods of the testing window. Additional testing would need to be used to determine if the time of day in and of itself had a statistically significant bearing on the distributed encryption system. For instance the 6-3 node test that had the highest average time was performed at 10 pm on Day 2 with an average time of 476.82 ms, but the very next day at 10:30 pm the average time for the 6-3 node test was nearly 100 ms less when it averaged 378.29 ms.

Hypothesis 3. The time taken for the distributed encryption system to authenticate will not be affected by taking nodes offline.

This null hypothesis was also disproved as shown in the results of the 6-3 node test, when no nodes were offline the overall average time of all 10,000 instances was 401.96 ms. When the three nodes were taken offline and only the minimum three nodes were left for the system to authenticate with then the time rose significantly, as the average time for the 6-3 test with three nodes down was 638.67 ms or 236.71 ms greater than the time when all six of the nodes were available for the system to utilize for the authentication process. Overall, the time for all three of the tests where nodes were taken offline increased significantly over the similar tests when the maximum number of nodes became available.

Hypothesis 4. If the number of nodes down exceeds the value of (t) for any given test the system will not be affected and will still authenticate and unlock the data store.

In this null hypothesis it is assumed that the system will not perform as intended and will actually unlock the data store even if the requisite number of nodes for the distributed encryption system are not available. Thus, for the 6-3 node test in which a minimum of three nodes are required to replicate the “true key” if only two of the six nodes were still available the system would still allow access to the data. This was easy to test as four nodes were taken offline and a 6-3 node test was performed, the system failed to authenticate and unlock the data store, another 100 tests were performed and in none of them was the system allowed to authenticate the Client and unlock the data store thus the null hypothesis was disproved.

All four of the null hypotheses were disproved by the data that was accumulated during the testing of the distributed encryption system. Obviously this was just a beta testing system and would not be something that could be implemented as a production system without significant improvements. That being said, the system did perform as it was designed to and allowed for multiple configurations of distributed cryptographic keys, and it failed to authenticate and unlock the data that was held in the data store when the requisite number of nodes failed to communicate with the API.

For this reason it is easy to see one of the potential values in implementing a system such as this for storage of cryptographic keys, it expands the number of points, which a malicious hacker would need to corrupt to retrieve or potentially destroy. One might argue that this creates a wider attack area that would need to be protected, and while this might be the case, these types of nodes would undoubtedly reside on dedicated back channel lines in a production system and would allow for easier monitoring of network traffic. This should give security personnel and system administration staff more time to observe potential malicious probing, also the fact that multiple sources would need to be corrupted before systems would

be truly in danger would allow staff the opportunity to shut access down or reorder the keys to make the loss of one or even two of the nodes irrelevant.

Another point to keep in mind is the potential benefit such a system could provide in the case of a disaster recovery situation. If your main key server goes down due to human error, natural disaster, or is destroyed by malicious attack the data that is encrypted in databases might be inaccessible for an extended period of time or potentially forever depending on the level of encryption and the amount of data that would need to be decrypted. For a modern enterprise that requires access to huge amounts of data, in some cases petabytes worth, to make business decisions such an event could be even more damaging than a data breach such as has occurred with greater and greater regularity. While a large-scale data breach in which credit card or personally identifiable information is exfiltrated from an enterprise might cause a public relations nightmare and potential legal and financial costs, an event in which an enterprise loses all or most of its data could potentially shut it down completely.

Potential problems with developing an enterprise grade solution for a distributed encryption system could include many things. Obviously as mentioned earlier, the greater number of nodes added to the system will increase the complexity and increase the system administration overhead as well. Also, the method used in this beta system whereby the "true key" was manually broken up across the nodes would quickly become overly complex and problematic to implement as the key size increased, which it would absolutely need to do to come in line with modern cryptographic requirements and the number of nodes increased.

As mentioned in the patent application for the distributed encryption system other options might be available to make the implementation less awkward and cumbersome to actually create (Sanders & Beals, 2008). For high value situations the use of such a system could provide potentially far greater security and the added benefit of increased disaster recovery potential.

The one major drawback that the timing data showed was that the system does have the potential to cause greater system latency, and could have noticeable impact on overall system performance especially for time sensitive systems such as an e-commerce website. In such an instance as an online commercial application the delay of even a few seconds could mean the loss of sales and of potential repeat customers and business. When the system was performing optimally the time it took to authenticate and unlock the data store was not overly burdensome, however in those test instances in which thousands or tens of thousands of milliseconds passed by the risk became much greater. Also, the time the distributed encryption would take to authenticate the Client would be on top of all the other backend system interactions that need to take place for a typical online purchase. When looked at in that context the increase of even a few hundred milliseconds could potentially propagate through the system and create a cascade effect on overall system performance.

All in all the distributed encryption system offers intriguing possibilities to increase overall system security and recoverability. It is an area in which more research should be done and new and upcoming technologies should be adapted to make the system more robust.

References

1. Abdalla, M., Bellare, M., & Neven, G. (2010). Robust Encryption. In D. Micciancio (Ed.) *Theory of Cryptography* (pp. 480-497). Zurich, Switzerland: Springer.
2. AWS Amazon. (2013). *Amazon Elastic Compute Cloud (EC2) Pricing*. Retrieved from http://aws.amazon.com/pricing/ec/2/
3. Barker, E., & Roginsky, A. (2012). *Recommendation for cryptographic key generation*. NIST Special Publication. Retrieved from http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-133.pdf
4. Boyd, C., & Mathuria, A. (1998). *Protocols for authentication and key establishment*. Berlin, Germany: Springer.
5. Chalkias, K., Baldimitsi, F., Hristu-Varsakelis, D., & Stephanides, G. (2008). Two types of key-compromise impersonation attacks against one-pass key establishment protocols. In J. Filipe & M. Obaidat (Eds.) *Communications in Computer and Information Science: E-business and Telecommunications* (pp. 227-238). Berlin, Germany: Springer.
6. Cloud Security Alliance (2013). *Cloud Security Alliance Warns Providers of 'The Notorious Nine' Cloud Computing Top Threats in 2013*. Retrieved from https://cloudsecurityalliance.org/media/news/ca-warns-providers-of-the-notorious-nine-cloud-computing-top-threats-in-2013/

7. Eastlake, D., Crocker, S., & Schiller, J. (1994). Randomness Recommendations for Security. *IETF RFC 1750*. Retrieved from http://www.ietf.org/rfc/rfc1750.txt
8. Goodin, D. (2012). *25-GPU Cluster Cracks Every Standard Windows in <6 Hours: All your passwords belong to us*. Retrieved from http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/
9. Hathaway, L. (2003). *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*. Retrieved from *http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf*
12. Hentoff, N. (2013). How many Americans will remember Edward Snowden? *Free Inquiry*, 33(6), 12-24.
13. Herong, Y. (2013). *Cryptography tutorials - Herong's tutorial examples*. Retrieved from http://www.herongyang.com/Cryptography/
14. JSON or Javascript Object Notation. (n.d.). Retrieved from http://en.wikipedia.org/wiki/JSON
15. McKinley, H. (2003). *SSL and TLS: A beginners guide.* Retrieved from http://www.sans.org/reading-room/whitepapers/protocols/ssl-tls-beginners-guide-1029
16. Mearian, L. (2009). Sun offers open source encryption key management protocol. *Computer World.* Retrieved from http://www.computerworld.com/s/article /9128101/Sun_offers_open_source_encryption_key_management_protocol
17. Moore, F. (2005). Preparing for encryption: New threats, legal requirements boost need for encrypted data. *Computer Technology Review*.
18. Moore, S. & McGabe, G. (2005). *Introduction to the Process of Statistics,* (5th ed.). Purdue University, In W.H. Freeman & Company.
19. Moore's Law. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Moore's_law
20. Raiu, C. (2012). Cyber-threat evolution: The past year. *Computer Fraud & Security.* 2012(3), 5-8.
21. Redman, J., Rice, E., Han, S., Anderson, R, Schwarting, J., & Paulson, B. (2013). Can a distributed key system broken up over multiple nodes provide greater security robustness while meeting system performance requirements? Retrieved from http://micsymposium.org/mics_2013_Proceedings/submissions /mics20130_submission_34.pdf
22. RSA Algorithm. (n.d.). Retrieved from http://en.wikipedia.org/wiki/ RSA_(algorithm)
23. Sanders, B. &Beals, T. (2008). U.S. Patent No. 8,050,410. Washington, DC: U.S. Patent and Trademark Office.
24. Shor's algorithm. (n.d.). Retrieved from the Shor's algorithm Wiki: http://en.wikipedia.org/wiki/Shor's_algorithm
25. Ylonen, T. & Lonvick, C. (2006). The Secure Shell (SSH) Authentication Protocol. *IETF RFC 4252*. Retrieved from http://tools.ietf.org/html/rfc4252