CAN A DISTRIBUTED KEY SYSTEM BROKEN UP OVER MULTIPLE NODES PROVIDE GREATER SECURITY ROBUSTNESS WHILE MEETING SYSTEM PERFORMANCE REQUIREMENTS

James Redman and Erich Rice
Department of Information Assurance
Saint Cloud State University
St. Cloud, MN 56301
reja0601 and rier1201@stcloudstate.edu

Sen Han
Department of Computer Science
Saint Cloud State University
St. Cloud, MN 56301
hase0701@stcloudstate.edu

Rick Anderson, Ben Paulson, and Joel Schwarting
Department of Information Systems
Saint Cloud State University
St. Cloud, MN 56301
anri0902, pabe0401, scjo0909@stcloudstate.edu

Abstract

The need for increased network security has never been greater, with a new high profile security breach seemingly occurring on a weekly basis. Encryption can be a valuable tool in increasing network security, however, its weakness often resides in the vulnerability of the cryptographic key. If a given node in a network contains the whole key and it is compromised, then the network's security is breached. Also, if the key is destroyed then all the information it protects could be lost. In this paper we look at the option of breaking the key up over multiple nodes, in this case six, while only three nodes are necessary to replicate the key and unlock the lock. This hopefully creates greater security while also providing a greater level of fault tolerance by allowing retrieval of the entire key even if three of the nodes are compromised.

1 Introduction

In this era of constant threats from malicious actors on the Internet, the need for more secure and robust transmission of data has never been more paramount. The recent attacks on the likes of the Federal Reserve, New York Times, and even Apple has shown us that the need for increased security in our ever more connected world will only gain greater importance. While much of the data currently sent over the Internet is in the clear, the newer forms of encryption algorithms available today are very capable of hiding the data from prying eyes, and are therefore considered quite robust [1]. The problem then becomes the way the encryption is used, most often by a means of using a cryptographic key, to make the session as unique as possible. An example would be Secure Shell (SSH), which is a cryptographic network protocol used for secure data transmission, it creates a secure channel over an insecure network connection [2]. The problem then becomes, if the key is compromised, the underlying data is put in peril if the encryption can be broken.

Although the current algorithms that are being used are very robust, they are also widely known and therefore there is a great deal of information on how they are structured and perform. This knowledge could give a potential hacker the ability to crack the encryption utilizing the information that is often commonly available via the Internet, see [3]. The very fact that there is so much information readily available to the public has made many of the older algorithms too vulnerable to utilize anymore. Another factor adding to the increased vulnerability of encryption algorithms is the constantly increasing nature of computing power. 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 [4] also with the growing power and availability of computing clusters or extremely high powered computers [5]. All of these factors have made the necessity of keeping the cryptographic key secure even more imperative.

One way in which the key can be made more secure is by controlling how the key is derived, and keeping it as secure and secret as possible once it has been created. A method that is commonly used is through the concept of key agreement protocols, such as those discussed in [2], in which both entities of the client/server model are used to derive the shared key that they will use. This then increases the security of the key by making it more difficult for a hacker to corrupt it, as they would have to compromise both the client and the server to gain the entire key structure [6]. Sun Microsystems, IBM, HP, and others have all proposed various options for effectively managing keys at an enterprise level [7], however, splitting the key over a distributed nodal environment has not been a method openly discussed as a commercially viable alternative. Instead a third party entity is often used as an intermediary between the client and server, and it acts as a key translation center and evaluates the method of encryption [8].

Within all of the different methods that have been either proposed [7] or implemented [2, 7] is the common theme that there are two salient points: the first

being that if the key is compromised then the encrypted data can be utilized by a hacker for their own purposes, and second if the key is destroyed by a hacker then the encrypted data is lost and can no longer be used by the rightful owner. An enterprise might look upon the second scenario as the more frightening of the two, as a hacker breaking in and stealing say credit card numbers might lead to bad publicity and inconvenience to its customers, but if those records were permanently made unavailable either to the company or the hacker the outcome could be the total dissolution of the company itself. This creates the need for some sort of redundancy, but the tradeoff is that more keys create a higher probability that the encrypted data can be compromised. Thus, there is a tradeoff between redundancy level and potential key vulnerability.

The key is the critical component in encryption strategy and the length of the key should be adequately assessed so that adequate robustness is achieved, and some have suggested encryption keys and passwords should be stored in escrow with a secure third party [9]. While the third party concept outsources the responsibility and places the key on a host external to where the data is stored, that host might be a very tempting target to a hacker. Regardless, it is paramount to have an effective key management plan in place, and it should be consistent within an enterprise. Successful key management is the linchpin to successful use of encryption and further research is needed especially in regard to key storage. This becomes critical in large security groups, especially when a member leaves [10].

The goal then is to balance the needs for greater security robustness and the need for fault tolerance and backup redundancy, against the increased system overhead and attack points created by breaking up the key across multiple distributed nodes.

2 Review Of Literature

The distribution of encryption methods across multiple nodes is not a new or novel idea, however, its use in enterprise level applications to this point has not caught on. As far back as the mid-1990s the concept of breaking up the encrypted key across multiple nodes has been bandied about, and even implemented, in a test scenario [11]. Here the authors implemented a distributed encryption and decryption system such that there was (n – t) nodal support, where n was the number of nodes in the system and t was the number of nodes that could be faulty due to hardware or software malfunctions. In their system the encryption is started at a certain "source node" and then proceeds along a path which contains the various broken up pieces of the key, until a *completed path* is performed with all the necessary pieces of the key [11, pp. 12-13] Ultimately they were able to implement their design to show that the algorithm which they had created worked with a t = 2 logic, while traveling five completed paths, within a Unix environment on a pair of Sun workstations [11, p. 19]

A more recent example of a work looking at a distributed key methodology across multiple hosts looked at basic key management, and a simplistic strategy of splitting the key up across a number of hosts (N) with a key size of 16 bytes [12]. In the authors example the key was split across four nodes, with a (N – 2) logic whereby up to two of the nodes could be compromised and the system would still function. In this example the keys used were relatively small, only 16 bytes total, with each of the nodes containing 12 out of the 16 bytes, in a sliding window pattern so that any two of the nodes would have the requisite bytes of the entire 16 byte key [12, pp. 5-6]. Guster, et al., also discussed one of the main advantages to the distributed key system, that a hacker even if they took over one of the key nodes would more than likely not know the total length of the key, and would thereby increase the difficulty in successfully implementing a brute force cracking of the key. They also looked at the possibility of generating the key, or portions of the key, through the use of a quantum number generator, which could increase the robustness of the key by making it more random and less vulnerable to pattern recognition and subsequent cracking [13, 14].

Another example of a distributed cryptographic key system being implemented can be seen in [15] where a Unix script was devised to split an RSA [16] algorithm across several hosts. A pre-generated key would then be read in a file and the output that was created by the Unix script would then be treated as fractions of the initial file in the working directory. Upon reforming the fragments, there would be a check against an MD5sum hash that had been generated to assure that file integrity had been maintained. While this was a good initial step at implementing a working solution to the problem, it was acknowledged that it lacked the complexity needed for a production environment [15, p. 10].

3 Methodology

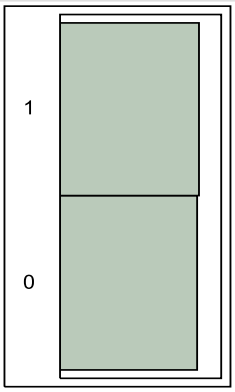
3.1 Number Randomness

As discussed in the preceding section, the randomness of the seed number for the key generation is an important part in determining the robustness of the encryption. The more random the numbers are the more unlikely it is a hacker could reengineer the algorithm in use, and replicate the key. Though it is difficult operation, given enough instances, patterns can begin to present themselves and that is when the algorithm can become vulnerable [14].

For the purposes of this project we looked at two options for generating the number to create the key: first we used a Javascript random number generator for the pseudo-random number generator, and second a quantum random number generator for the other. The Law of Large Numbers states that, as the number of observations increases, the mean x-bar will eventually approach the population mean (as closely as you'd like to estimate the population mean) [17, 18]. Now, we know that the population mean is 0.5 ((1+0)/2), so eventually our estimate will hit 0.5 as well. We used 10,000 observations for this first

iteration, and you can see in Table 1 below, that the pseudo-RNG performed well. The closer to 0.5 it is, the more "unpredictable" it is, because the value will not lean probabilistically toward one way or the other.

Pseudo-Random Number Generator

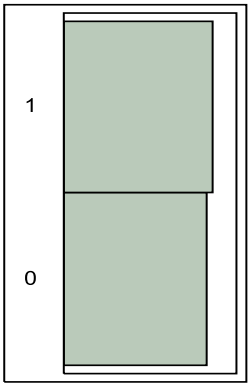


Frequencies

Level	Count	Prob
0	4997	0.49970
1	5003	0.50030
Total	10000	1.00000
N Missin	0	

2 Level

Quantum Random Number Generator



Frequencies

Level	Count	Prob
0	4930	0.49300
1	5070	0.50700
Total	10000	1.00000
N Missin	340000	

2 Level

Table 1: Comparison of Pseudo-Random vs. Quantum

Going off of the reasoning that the closer to the mean, the more random our RNG is, it seems our Quantum RNG performed worse on this try than the Pseudo-RNG. However, we didn't just perform this test once and give up. For thoroughness' sake, we did this test 35 times, each with 10,000 observations, for each type of RNG. The averaged results are as follows: Pseudo RNG was Prob(0) = 1 – Prob(1) = *0.515* while the Quantum RNG was Prob(0) = 1 – Prob(1) = *0.506.* Thus with more iterations the Quantum RNG did perform slightly more random than the Pseudo RNG, although many more iterations would need to be run to get a more statistically significant picture, for our purposes we choose to use the Quantum RNG to produce the key values.

3.2 Splitting the Key

For our system implementation we choose an (N – 3) logic, where N represented six nodes within a distributed encryption system, thus building upon the authors work in [12] we needed to determine how to split up the key across the nodes to allow any three to replicate the key. To take full advantage of the added security the distributed encryption system provides we also needed to make sure that any two nodes did not recreate the key. And finally, to prove the fault tolerance and added recoverability of the distributed key system we needed to split the key in such a way so as up to three nodes could be lost, including their portions of the key, and the remaining three nodes could recreate it.

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

Table 2: Splitting the Key Across the 6 Nodes (X=missing bytes)

As can be seen in Table 2 the key is broken up into 24 bytes, with each byte being equal to 8 bits, and represented by two Hex characters. We chose to use the 24 byte / 192 bit AES (Advanced Encryption Standard) encryption because it is a very robust encryption method approved by the Federal Government up to "TOP SECRET" classification [19], with AES 192 bit taking roughly $2^{189.7}$ operations to recover the key [20].

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		

Table 3: Key (Top Boxes) with Initialization Vector (IV) and Salt

Table 3 shows the key represented by the two Hex characters under each of the 24 bytes across the top of the Table, the IV and SALT are then listed under the key, and finally the six rows beneath them represent the six nodes of the distributed key system and show the key broken up according to the method described in Table 2. The blank spaces in Table 3 are the equivalent to the X's that marked the missing bytes in Table 2 they were left blank so as not to be confused with an X Hex character.

3.3 Creating the Distributed Encryption Environment

To create the environment that the distributed encryption system would reside on, it was determined to create the various nodes on virtual machines on a Xen Server, all the servers were running Node.js, which is built on JavaScript and can build "fast, scalable network applications. Node.js uses an event-driven non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices" [21]. All of the nodes were running Ubuntu Linux 12.04 64 bit Edition.

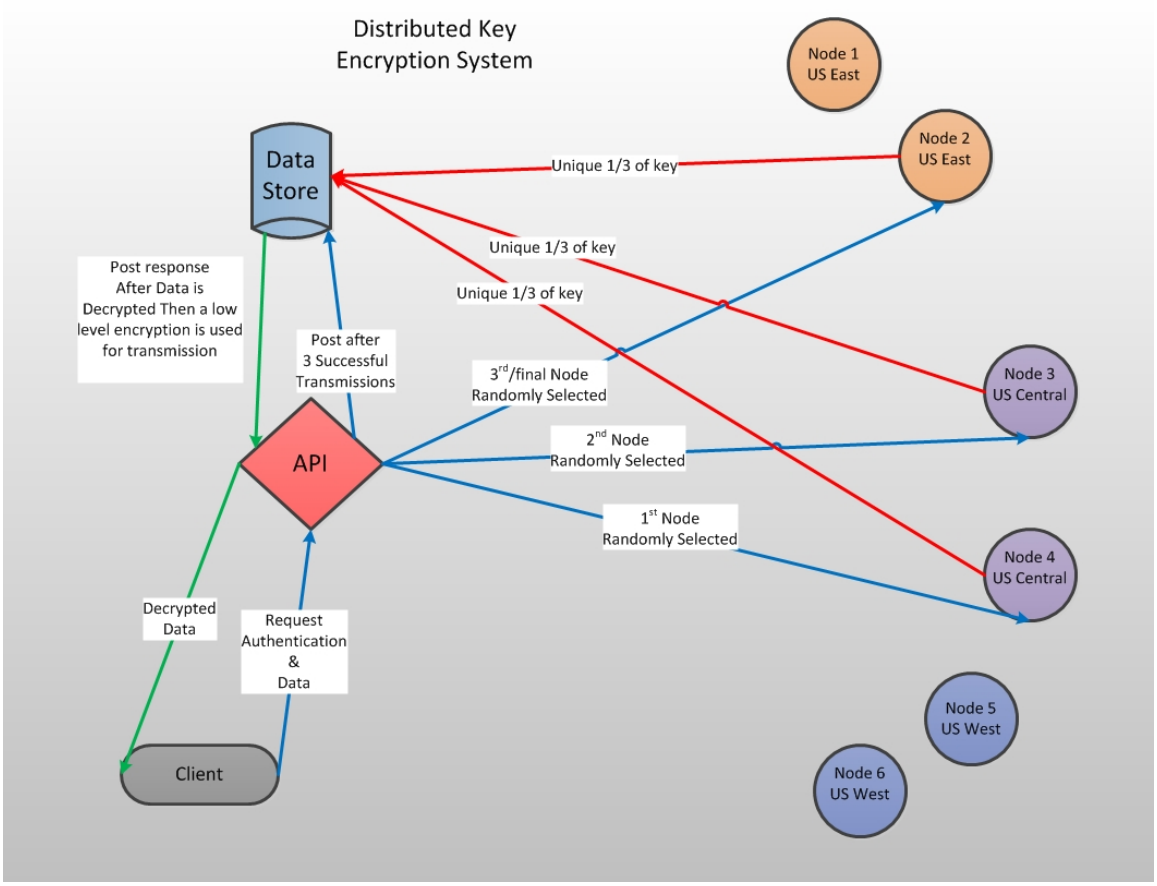


Figure 1: Diagram of the Distributed Key Encryption System

3.3 Distributed Key Encryption System Process Explained

1. User loads a client interface – This is where a user would put in a name and password to authenticate with the Node.js API. The user will request the data they need access to from here. A post is sent to the API from the client side interface using HTTPS and JSON.
2. API – When the API receives the Post a number of actions take place.
    a. The user is authenticated with the API.

b. The API creates a unique session id (GUID).
c. The API selects a random sequence of key nodes (refer to Figure 1).
d. The API begins sending HTTPS posts to each node one at a time, but after three “Key Nodes” responds with a successful transmission to the “Data Store” the API stops sending out “Key Node” requests.
    i. If a key Node fails to respond with a successful transmission to the Data Store, the API tries a new randomly selected Key Node, which has not been selected previously.
    ii. The API sends a JSON object in the post, which identifies:
        1. Unique session Id
        2. Requested Key Node/ Key section Id
        3. Requested Data Store
        4. Requested Data
e. After all three successes, the API sends an HTTPS post to the “Data Store” requesting decrypted Data.
f. API receives the Data store response, and then decrypts a network transmission level encryption.
g. The API then responds to the user’s original HTTPS Post request, with the Decrypted Data.

3. Key Nodes – Each “Key Node” stored a unique section of the key (1 of 6 unique parts, any 3 of which for the full key, refer to Table 2 and 3).
    a. When the Key Node receives the API Post request, the node looks at the JSON in the request and identifies which Data Store is required, and which session the request is from.
    b. The Key Node pulls the key section from the database and appends it to the JSON object, which was sent from the API.
    c. The Key Node then sends the JSON object through an HTTPS post request to the correct Data Store.
    d. Once the Key Node receives a successful response from the Data Store, the Key Node responds to the API with a Successful response.
4. Data Store – The Data Store would be any server that is holding encrypted Data
    a. When the Data Store receives the first “Key Node” HTTPS Post request, the Data Store reads the JSON object, which was sent.
        i. It first authenticates the Key Node.
        ii. It then identifies which Key Node / Key Section the request came from.
        iii. It then identifies what Data is requested.
        iv. It then records which session the request is from.
    b. The Data Store then creates a unique session, and all 3 nodes will manage a specific Data request.
    c. The Data Store then sends a successful response signal to each Key Node.

d. For each remaining Key Node Post Request, the Data Store will identify which session it is apart of and process the Key Sections accordingly.
e. The Data Store then waits to receive a Post Request from the API, which will be identified by the GUID.
f. Then the Data Store will use the 3 sections to re-create the full key.
g. The full key is then used to decrypt the encrypted data.
h. Once the Data is successfully decrypted the Data Store will apply a basic level encryption for network travel safety.
i. Finally the Data Store will respond back to the API with the data.

4 Results

The first test of the system was to see how it would respond in a more typical third party fashion as described in [9], where the entire key resides on one node (0(ms) column below) and it responds with it to the Data Store, then we tested increasing numbers of nodal interactions to see how the transmission latency was effected.

# of Nodes / Experiments	0(ms)	1(ms)	2(ms)	3(ms)	4(ms)	5(ms)	6(ms)
Experiment 1	67	72	63	94	90	123	86
Experiment 2	35	44	44	32	41	48	58
Experiment 3	21	34	39	44	35	51	48
Experiment 4	23	43	39	69	46	54	56
Experiment 5	22	34	40	32	44	41	46
Experiment 6	20	36	61	42	57	54	44
Experiment 7	18	33	42	34	39	45	54
Experiment 8	21	41	40	29	32	39	42
Experiment 9	20	37	38	31	44	47	54
Experiment 10	28	37	45	40	46	45	59
Experiment 11	19	38	44	33	34	68	45
Experiment 12	20	34	38	40	44	41	51
Experiment 13	23	37	33	33	47	61	43
Experiment 14	20	33	27	64	38	47	44
Experiment 15	22	33	63	35	53	41	48
Experiment 16	22	28	34	40	53	36	41
Experiment 17	33	42	33	42	38	46	45
Experiment 18	22	39	43	37	43	50	64
Experiment 19	18	39	38	36	35	47	47
Experiment 20	22	34	32	37	32	42	53
Average	24.8	38.4	41.8	42.2	44.55	51.3	51.4

Table 4: Response Time of Additional Nodes

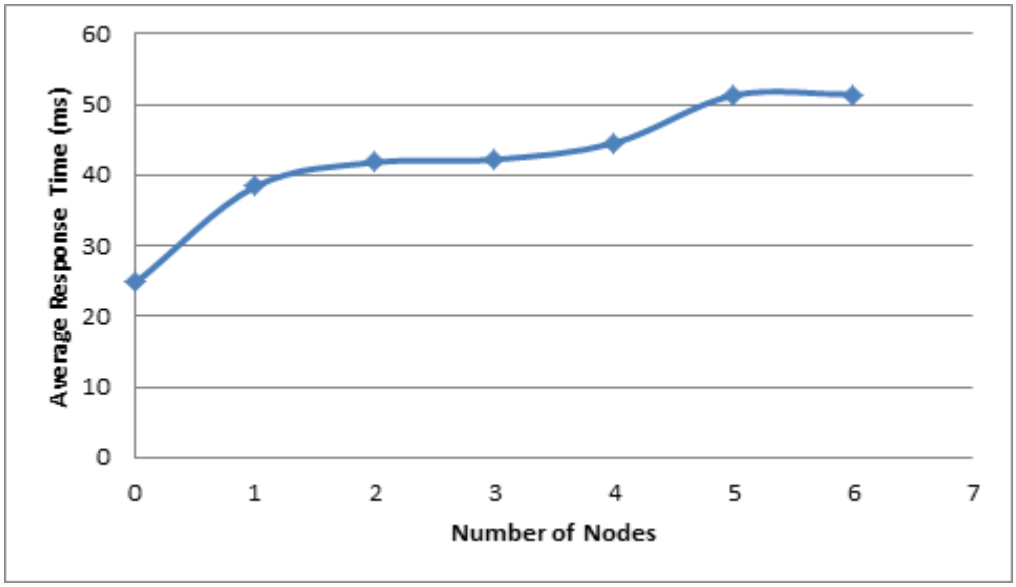


Figure 2: Graphical Depiction of Additional Nodal Latency

As can be seen from the experimental data, there is a sharp increase in average response time going from the typical third party single key holder framework, but after that, adding additional nodes to transmit their respective pieces of the key adds only a small amount of additional time to the process. Now, it is important to point out that in this case the nodes are all in close proximity, whereas in an optimal production environment they would be widely distributed, so as to take advantage of maximum disaster recovery and data redundancy advantages. These would then have to be balanced against the need to meet the 3-second rule of e-commerce [22].

The next test that was run was to determine how the system would respond when one or more of the nodes were down. This could be caused by hardware or software failures, or potentially if a hacker had compromised it and its portion of the key was destroyed. Again the fault tolerance logic that was used was (N – 3) where N represented the six nodes in the test bed system, so we tested it with zero through three nodes down, to make sure that the fault tolerance aspect of the system was working properly. At this time we have not tried to run the system in a positive fail mode, whereby more than three of the nodes have failed, as there was concern that it may throw multiple exceptions within the code as it currently is written. As we continue to develop the system and tune it for more optimal performance, this will be another part of the process towards attaining a working level of quality assurance.

	0	1	2	3
1	36	53	84	84
2	73	46	52	52
3	58	38	40	52
4	44	39	55	52
5	64	43	42	43
6	49	41	41	41
7	32	59	32	44
8	33	39	36	41
9	38	38	34	49
10	40	31	39	65
11	53	92	54	49
12	35	42	62	66
13	41	40	53	37
14	56	33	36	36
15	39	32	49	45
16	37	84	34	44
17	58	33	43	40
18	28	42	42	33
19	34	40	37	61
20	33	37	37	32
AVERAGE	44.05	45.1	45.1	48.3

Table 5: Showing Added Latency With Increasing Nodal Failure

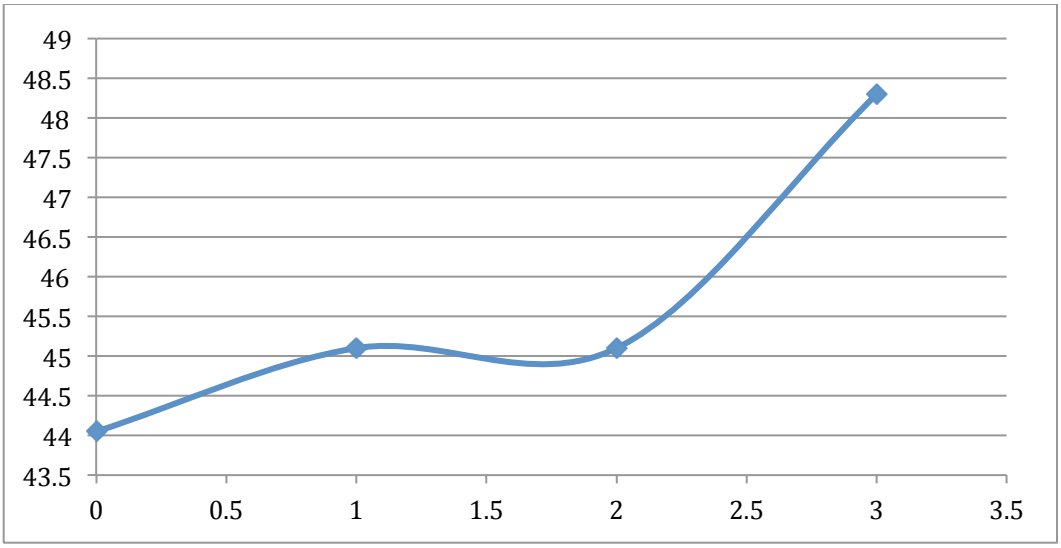


Figure 3: Shows Graphical Increase in Latency

As can be seen in Table 5 and Figure 3, the latency within the distributed encryption system increases slightly with one nodal failure, with two nodal failures there is almost no change in performance, and finally when the system is fully stressed with three nodal failures there is a sharp increase in latency, albeit only 3.2 ms. Again, it is important to remember that in our test bed system the nodes are in close

proximity to one another, and in a production environment they would more than likely be widely distributed geographically, increasing overall latency.

In determining the added security robustness of splitting the key across multiple hosts we utilized the following logic:

A basic 48 hex key is 1.59309E-58 assuming each piece of the key is independent and it takes three sub-keys to determine the whole key utilizing the overlapping key method in Tables 2 and 3, which gives us: 16^32 * 16^12 * 16^4 = 1.59309E-58

This is the same robustness, but the value is in this case related to the probability of compromising a host and then the security therein of the host, using localized values in the Business Computer Research Lab (BCRL) at St. Cloud State University we find that there are 3/86400 (3 compromises on any given day, based on data from a one year period) = 3.47222E-05

So probability of compromising one host = 3.47222E-05

Probability of compromising 3 hosts = 4.65136E-15

Thus by multiplying key probability * host probability we get for one host 5.53157E-63 and for 3 hosts 7.41005E-73, a significant overall increase in robustness.

5 Conclusions

From our development of this distributed encryption system we have been able to determine that it does in fact work, albeit in a virtualized setting, and does produce added fault tolerance as well as increased security robustness. The challenge will now be in fine-tuning the system to better handle adversity, whether through malicious or non-malicious circumstances, as well as to create a more realistic test bed environment. This could be done through a number of methods, including placing some of the nodes in physical machines at great distances from one another, similar to the setup that was envisioned in the diagram of Figure 1. Also, the way the code was written it will allow us to increase the number of nodes to make the system even more fault tolerant and secure. The way the cryptographic key is split among the nodes is another area to be investigated, as the more random it becomes the less likely a hacker could discern a pattern. We also will be looking at other encryption methods to take advantage of the most robust and secure algorithms currently available, but again with an eye towards maintaining the 3-second rule for e-commerce purposes, because in a production environment a security lapse can mean the loss of millions of dollars, but a poorly tuned and slow responsive system can mean loss of customers and perhaps ceasing to exist.

References

[1] M. Abdalla, *et al*., "Robust Encryption," Cryptology ePrint Archive, Report 2008/440; http://eprint.iacr.org/2008/440.pdf.

[2] T. Ylonen and C. Lonvick, *The Secure Shell (SSH) Authentication Protocol*, IETF RFC 4252, January 2006; http://tools.ietf.org/html/rfc4252.

[3] Y. Herong. (2013). *Cryptography Tutorials – Herong's Tutorial Examples*. [Online]. Available: http://www.herongyang.com/Cryptography/.

[4] Wikipedia (2013, March 10) *Moore'sLaw*. [Online]. Available: http://en.wikipedia.org/wiki/Moore's_law.

[5] D. Goodin. (2012, December 9) *25-GPU Cluster Cracks Every Standard Windows in <6 Hours: All your passwords belong to us*. [Online]. Available: http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/.

[6] J. Filipe and M. Obaidat, "Two Types of Key-Compromise Impersonation Attacks against One-Pass Key Establishment Protocols," Communications in Computer and Information Science, 2008/23, p. 227-238.

[7] L. Mearian. (2009, February 17) *Sun offers open source encryption key management protocol.* [Online]. Available: http://www.computerworld.com/s/article/9128101/Sun_offers_open_source_encryption_key_management_protocol

[8] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment.* Berlin, Germany. Springer, 1998.

[9] F. Moore, "Preparing for Encryption: New threats, Legal Requirements Boost Need for Encrypted Data," *Computer Technology Review*, (2005, August 1).

[10] Y. Tseng, "A Scalable Key Management Scheme with Minimizing Key Storage for Secure Group Communications," *International Journal of Network Management*. Vol. 13, issue 6, p. 419-425.

[11] A. Postma, *et al*., (1996). *Distributed Encryption and Decryption Algorithms* [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.3186.

[12] D. Guster, *et al*., *Enhancing Key Confidentiality by Distributing Portions of the Key Across Multiple Hosts* [Online]. Available: http://www.cs.uwec.edu/MICS/papers/paper_11.pdf.

[13] T. Ristenpart and S. Yilek, *When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography* [Online]. Available: www.isoc.org/isoc/conferences/ndss/10/pdf/15.pdf.

[14] D. Eastlake, *et al.*, *Randomness Recommendations for Security,* IETF RFC 1750, December 1994; http://www.ietf.org/rfc/rfc1750.txt.

[15] I. Rego, *et al.*, *Implementing a Distributed Key Algorithm to Enhance Confidentiality and Provide Fault Tolerance* [Online]. Available: http://www2.css.edu/mics/Submissions/submissions/Implementing%20a%20Distributed%20Key%20Algorithm%20to%20Enhance%20Confidentiality%20and%20Provide%20Fault%20Tolerance.pdf.

[16] Wikipedia (2013, March 12) *RSA (algorithm)*. [Online]. Available: http://en.wikipedia.org/wiki/RSA_(algorithm).

[17] S. Ghahramini, *Fundamentals of Probability with Stochastic Processes, 3rd edition.* Western New England College, MA. Pearson, 2005.

[18] S. Moore and G. McCabe, *Introduction to the Process of Statistics, 5th edition.* Purdue University, IN. W.H. Freeman & Company, 2005.

[19] L. Hathaway, (2003, June) *National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information* [Online]. Available: *http://csrc.nist.gov/groups/ST/toolkit/documents/aes/CNSS15FS.pdf*.

[20] Wikipedia (2013, March 15) *Advanced Encryption Standard*. [Online]. Available: http://en.wikipedia.org/wiki/Advanced_Encryption_Standard#cite_note-24.

[21] Nodejs.org (2013, March15) [Online] Available: http://nodejs.org.

[22] J. Overstreet (2012) *E-Commerce & The Importance of Site Speed: Why Site Speed Matters Big Time For Retailers* [Online]. Available: http://blog.edgecast.com/post/31472458187/e-commerce-the-importance-of-site-speed.