Enhancing Key Confidentiality by Distributing Portions of the Key Across Multiple Hosts

Dennis Guster, Christopher Brown and Renat Sultanov
Business Computing Research Laboratory
St. Cloud State University
St. Cloud, Minnesota 56301
dcguster@stcloudstate.edu

Abstract

A concern with many of today highly robust algorithms is keeping keys secret. Sometimes this is difficult because the key is stored within a regular file system on a Unix host. If hackers obtain root access to that host then the key(s) are potentially compromised. To combat this problem the authors suggest storing portions of the key across several hosts so that in the event that one host is compromised the integrity of the keys will remain intact.

1 Introduction

The sophistication and robustness of the majority of encryption algorithms in use today is quite impressive [1]. However, many of these techniques when deployed in an operational setting make use of keys to ensure that each encryption session is unique (or at least very close to unique). A good example of this methodology would be secure shell (ssh) in which every client connecting to a given ssh daemon would use the same encryption algorithm, but be assigned different keys. In other words, more simply stated use a different "seed" number so each client's encrypted message would look different. Therefore, when using this methodology no matter how robust the algorithm if the key is compromised the data is in potential danger of being compromised. Further, while the widely used algorithms have proven to be quite robust there is a wealth of information on the internet that directly delineates the formulas they use as well as their potential vulnerabilities, see for example [17].

The weaknesses of the present methodology have in part been acknowledged and the need for improved encryption strategies recognized. A particular noteworthy account of this situation is depicted in the testimony to the US Senate by Barbara Simons of the ACM. Who pointed out given the capability of hackers to obtain their own clandestine computing clusters encrypted data is especially vulnerable to brute force attacks and particular the importance of sound key management [13]. The importance of sound key management is technically described in [4]. There has been much written since these early works including an interesting book chapter edited by Filipe and Obaidat, [6] and of course the topic has been expanded to deal with wireless communication in works such as [10].

One of the key principles of a sound key deployment strategy is effectively controlling how the key is generated and keeping it secret once it is generated. One of the common methods used to maintain those principles is the concept of key agreement protocols in which information from both entities (in a client/server model) is used to generate the shared key [6]. The logic being that the hacker has to compromise two different hosts to derive the key. This premise certainly makes it harder for the hacker and appears to be a sound strategy. Deriving a key from multiple sources can easily be expanded. In some cases a key translation center is used to provide a third entity that is needed to provide the appropriate evaluation of the encrypted method [3]. Sun Micro-Systems, [14] presents several strategies for efficient data encryption in the enterprise, but suggests the key and the management of that key are the primary foundation for that strategy.

Certainly, the importance of the key cannot be overstated if one evaluates two very important facts: if the key is compromised sensitive data can be read by the "bad guys" and if the key is lost the data cannot be read by the "good guys". So therefore protecting the key and not losing the key become paramount. Given a scenario in which a hacker breaks into a host and compromises the key and performs some mischief and then is effectively banned from the enterprise on the network firewall level. In addition to that sensitive data being compromised the hacker might modify or destroy the key as well. If a redundant key strategy is not in place then the data may be lost. Certainly some sort of

redundancy is needed, but the more keys there are the higher the probability that the encrypted data can be compromised. In other words, there is definitely a tradeoff between redundancy level and potential key vulnerability.

Moore, [11] also sees the key as the critical component in encryption strategy and feels that key length should be adequately assessed so that adequate robustness is achieved and suggests that encryption keys and passwords should be stored in escrow with a secure third party. While the third party concept outsources the responsibility and places the key on a host external to where the data is stored, the key might still be compromised if the third parties secure host is compromised and it might be a very tempting target to a hacker. In any case it is important to establish an effective key management plan. Key management is the key to successful use of encryption and research that addresses the vulnerabilities of key storage, generation and effectiveness is still needed especially in regard to key storage. This is especially critical in large security groups when a member leaves the group [15]. The issue of key generation and management also becomes critical in a wireless world. This is especially problematic in satellite applications in which the key generator may need to be on board the satellite and a reliable means is needed to ensure the keys are protected and synchronized with ground stations [5].

2 Review of Key Management

While some feel that moving to a highly efficient automated device independent system is needed its design and implementation will take time and the idea still needs further development [14]. Work such as [4] recognized the need to improve on key generation and management. It would appear that the goal would be to limit the number of sites that potentially have full access to the key and or the encryption strategy. The literature suggests the concept of proxies to help isolate the encryption process [2]. In other words, a third party takes responsibility for some of this process which means clients do not need to take total responsibility for the encryption process. If hundreds of clients are involved and serviced by a set of redundant proxy servers then the total number of devices that have total knowledge of the process is reduced. The effectiveness of this strategy is in part validated by the inclusion of proxy re-encryption and re-signature routines in the JHU-MIT Proxy Re-cryptography Library, [9].

However, while the use of proxy servers may reduce the vulnerability on the client level the proxy server its self could be consider by hackers to be a prime target because of the wealth of information it might possess. So therefore strategies that limit the volume of information on this device might be considered attractive. Specifically, [2] suggests key server trust limiting mechanism so that the key server need not be trusted with every key within the system. One way of achieving this goal is to split the key among multiple key servers. Filipe and Obaidat, [6] report the development of a shared key schema which allows multiply passes with using multiple key elements. US patent 6026163 [16], reports the development of a distributed split-key cryptosystem in which each trustee is able to independently select their own key pairs.

While the idea of splitting the key and the encryption process has merit from a security perspective, there is still a need to make sure the added complexity does not cause a dramatic degradation of performance. Specifically, Hong-Wei, Shi-Hui and Ping-Ping, [8] point out the need to balance robustness with performance particularly in wireless applications. However, even if the key is split to be employed it must be assembled in memory and if it is all contained in the memory of one device then that key of course would still be susceptible to cold boot attacks [7].

3 A Basic Strategy in Key Distribution Across Multiple Hosts

The basic strategy involves limiting access to the entire key. To accomplish this in today's autonomous systems which feature multiple hosts in a distributed environment is easy to accomplish because pieces of the key can be stored across several host. This logic means that if any given host is compromised only a portion of the key will be compromised and without the entire key that portion has limited value. In its simplest form one could alternate key positions across the key servers. To illustrate this basic premise an example will be delineated involving four server hosts with a key size of twelve characters and a server key size of 3 characters. The basic truth table is depicted below as Table 1:

	n=4		key=12
svr			svrKey=3
a	1	5	9
b	2	6	10
c	3	7	11
d	4	8	12

Table 1: Basic Truth Table for 4 Servers and Key Size = 12.

This truth table indicates that the distributed server host architecture will consist of four servers, *a-d*, each server will host 3 characters (svrKey) of the total 12 character key. Server *a* will host relative key positions 1,5,9 and server *b* will host key positions 2,6,10 and so forth. In other words, each character alternates in sequence across all 4 server hosts. Of course this scenario is limited because the pattern is quite predictable and there is not fault tolerance. So if any of the 4 hosts is down then the encryption/de-encryption process comes to a halt. Certainly the number of server hosts used and the key size can be varied to obtain different levels of complexity. However, the complexity level needs to be carefully assessed because added complexity tends to result in less reliability and performance issues. However, depending on the configuration method there may be some exceptions. Within fault tolerance logic, more specifically raid5, whereby the added complexity of a parity bit actually improves system stability. Table 2 below provides a "what if" scenario in regard to expanding the number of server hosts and key sizes.

svr	n=6		key=36 svrKey=6			
a	1	7	13	19	25	31
b	2	8	14	20	26	32
c	3	9	15	21	27	33
d	4	10	16	22	28	34
e	5	11	17	23	29	35
f	6	12	18	24	30	36

Table 2: Basic Truth Table for 6 Servers and Key Size = 36.

In this example, the number of server hosts has been increased to 6 (svrs a-f) and the key has been increased to 36 characters. Of course one might argue that the rest of the key might be derived via social engineering although it is not likely since the key would be very random in nature. It might also be argued that the time to derive it using brute force may be reduced which seems more plausible than the social engineering vulnerability. However, even if one accepts the latter argument the brute force attacker need to realize that the key is a certain length and that the missing portion of the key will need to be interpolated rather than cracked. So therefore, brute force might still succeed, but would still probably be more difficult than if the whole key was compromised. The logic that would be used to implement this alternating character methodology (process A1) is depicted below in Figure 1. This flow chart begins with a number of basic functions such as setting the key length, the number of servers, the server ID, the key position and the server key position. The next step is retrieving the key value followed by incrementing the key position. If the key position <= to the key length then the next server key position is calculated. This looping process continues until the key is retrieved.

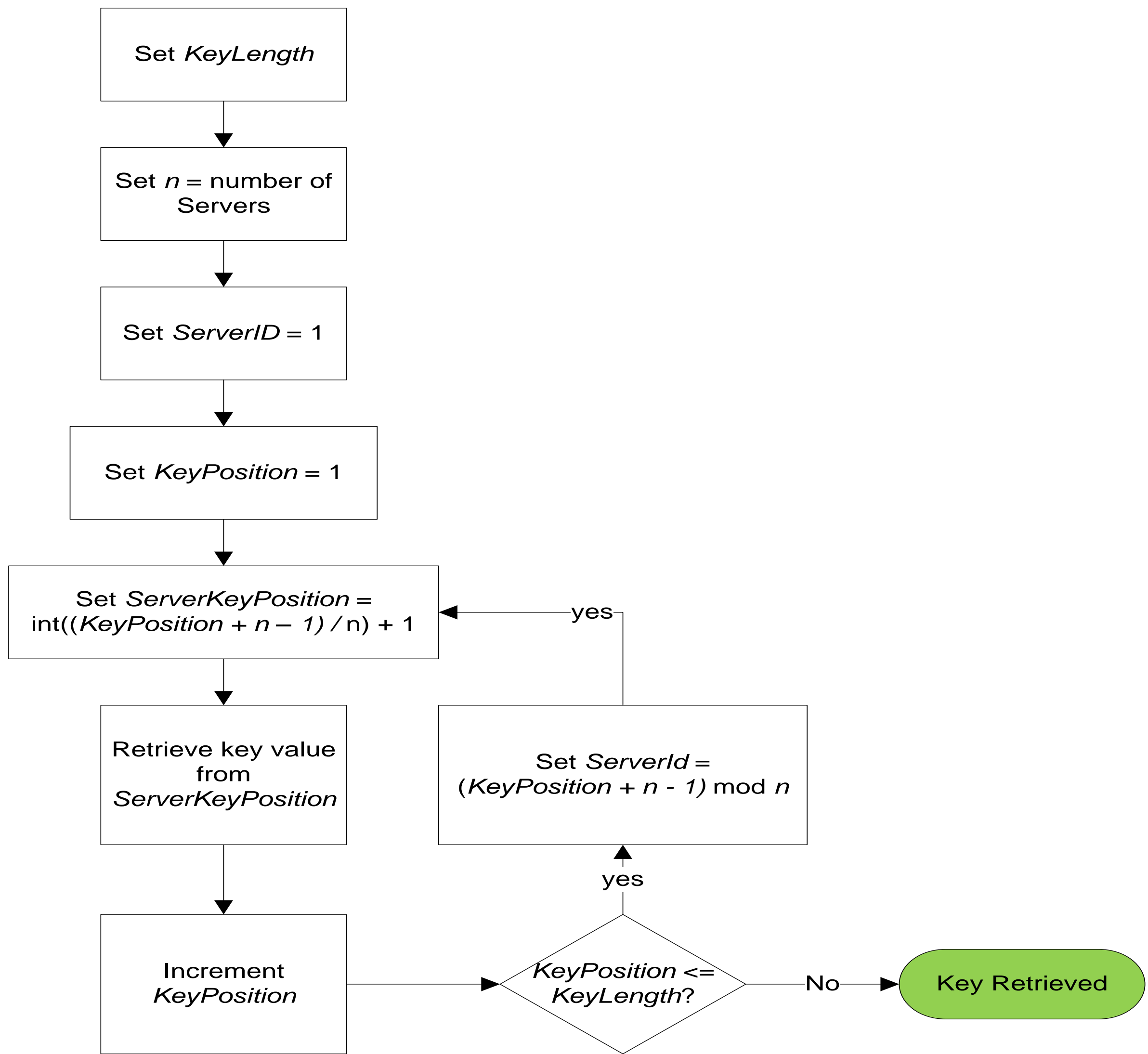

Figure 1: Flow Chart Process A1.

Of course one of the primary problems with process A1 was its lack of fault tolerance. Using the same basic logic of distributing a key across multiple hosts process A2 was developed and it structure is illustrated in the example that follows. In this example the key will be distributed across four different hosts (N) and the key length will be sixteen bytes. Of course the goal is to make sure that the key never entirely resides on any one host. Further, one can expect reliability problems with any one host so therefore there is a need to provide fault tolerance for this very important security process. A simple solution

would be to invoke a N-2 reliability logic in that the key can still be extracted if any two of the four hosts fails.

In this scenario the key structure would look as follows (where the numbers represent relative byte position in the key).

Host1: 1 2 3 4 5 6 7 8 9 10 11 12
Host2: 5 6 7 8 9 10 11 12 13 14 15 16
Host3: 9 10 11 12 13 14 15 16 1 2 3 4
Host4: 13 14 15 16 1 2 3 4 5 6 7 8

In other words, each host has only ¾ of the key (12 out of the 16 bytes), but the key can be built from any two of the hosts. For example suppose host three was down, the key can still be obtained by getting positions 1-12 from host 1 and 13-16 from host2. Note in this example that host 4 could also fail. What if host 1 and 2 fail. Then the key could be derived from host 3 and 4 with host 3 providing key positions: 1-4; 9-12 and host 4 providing positions 5-8. Obviously this example could be vastly improved from a security perspective, but it does illustrate the basic premise and that premise is that if a host is compromised the entire key is not. Another way of looking at this design might be to use the term augmented layering distribution, as that is essentially what is occurring between the server key segments, whereby each segment is slightly overlapping the other. Perhaps a future study concept might be looking into utilizing a hot standby if one system were to fail, or a dynamic randomization model where the key placements are randomly reorganized at random intervals.

4 Relationship Between/Among Number of Key Servers and Key Length

To make this relationship work there is a minimum number of bytes for any key and that number is a function of the number of key servers (hosts) selected. These relationships are depicted in Table 3 below. For example if two servers were selected the key size required would be six bytes and each server key would then be required to contain four bytes. So therefore, host one would contain positions 1234 and host 2 positions 5612(note this example does not provide fault tolerance). However, fault tolerance begins with three servers in that host1 contains positions 1 2 3 4 5 6 7 8 9, host2 positions 4 5 6 7 8 9 10 11 12 and host3 positions 7 8 9 10 11 12 1 2 3. In this case a N-1 fault tolerance scheme results in which any one of the hosts can fail and the remaining hosts can still recover the key.

Key Size		Interval (i)								
		2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Key Servers (n)	**2**	6	8	10	12	14	16	18	20	22
	3	9	12	15	18	21	24	27	30	33
	4	12	16	20	24	28	32	36	40	44
	5	15	20	25	30	35	40	45	50	55
	6	18	24	30	36	42	48	54	60	66
	7	21	28	35	42	49	56	63	70	77
	8	24	32	40	48	56	64	72	80	88
	9	27	36	45	54	63	72	81	90	99
	10	30	40	50	60	70	80	90	100	110

Server Key Size		Interval (i)								
		2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Key Servers (n)	**2**	4	6	8	10	12	14	16	18	20
	3	6	9	12	15	18	21	24	27	30
	4	8	12	16	20	24	28	32	36	40
	5	10	15	20	25	30	35	40	45	50
	6	12	18	24	30	36	42	48	54	60
	7	14	21	28	35	42	49	56	63	70
	8	16	24	32	40	48	56	64	72	80
	9	18	27	36	45	54	63	72	81	90
	10	20	30	40	50	60	70	80	90	100

Figure 2: Truth Tables for Process A2.

The logic to implement the A2 algorithm is depicted in a flow chart form in Figure 2. This flow chart begins with a number of basic functions such as setting the key server offset ((i) of course i is the interval or offset related to the key length – the overlap or augmentation), the key length ($n \cdot i$), and the condition $n < 2$ is tested. If the condition is true an error occurs because that is the minimum allowed size. If the condition is false then the key length is set according to $n \cdot (i/1)$. Next the server ID is set at 1 which is followed by setting the server key position $F(x)$. Next a decision box for checking if a key value position is missing for the server key. If yes then a retrieve process is implemented and then the server key position is incremented. Otherwise control is passed directly to the incrementing of the server key position process. The next condition that is checked is to determine if the server key length is less than or equal to the key length. If so, control loops back to the key value position missing for the server key position decision box. Otherwise the logic moves on to the increment server ID according to the appropriate algorithm (see appendix A). Following this process another condition is tested to ascertain whether the server ID is less than or equal to n. This is to ensure fault tolerance and to do so effectively the number of servers must also be adequate to retrieve

the key. If this condition is true the logic loops back eventually to the set server key position process to support a rebuild of the key. Otherwise, control is passed to another decision box which checks to determine if the received key length is equal to $n \bullet (I + 1)$. If this is true the key has been successfully retrieved, otherwise an error has occurred.

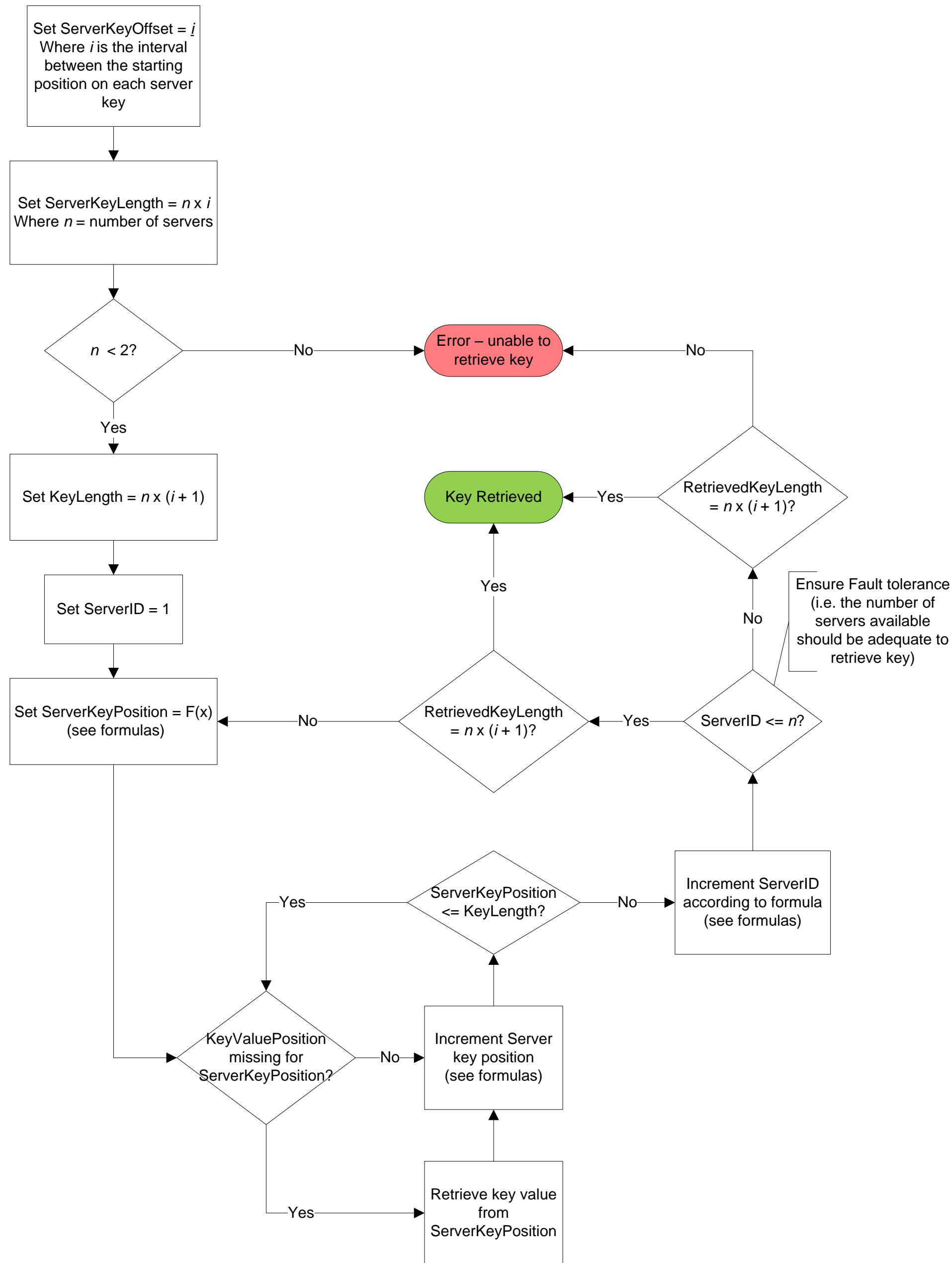

Figure 1: Flow Chart Process A1.

5 Discussion/Conclusions

While the examples presented herein lack the complexity needed for a production system it is hoped that the basic logic for such a system has been demonstrated. Obviously the key complexity is not there with the 12 digit key used to illustrate the fault tolerant example. However, the logic is such that the size could be easily expanded. Deriving the key is also a critical portion of the algorithm, one method would be to simply generate it on a "master" host and then split and store appropriately thereafter. This method would mean that the whole key would reside on a single host for a short period of time. A second method would fit well with the four server model presented in the Basic Strategy in Key Distribution Across Multiple Hosts section of this paper. In this model sever 1 could generate positions 1-6, sever 2 position 7-12 and then the resulting values could be distributed accordingly. In this case at least the whole key is never entirely on one host. There are certainly other and more complex variants of this basic premise.

Also, the manner in which the key or "key portions" are generated is also a concern. The first method one would consider is to use some type of classical random number generator and if sophisticated methodology is employed this method may in fact be adequate. However, given the work of Shannon, [12] it is clear that there are limitations even with the best classical system. Therefore, one might consider generating the random numbers using some type of quantum system. Devices are now available for between $1,500 to $3,000 and marketed from companies such as http://www.idquantique.com/index.php/component/content/article/6.html.

In summary, this paper attempted to present some basic algorithms that could be used to split the encryption key among multiple hosts so that the whole key would never totally reside on one host thereby protecting its identity if any given host within the host group was compromised at the root level. Further, the importance of providing fault tolerance within the host group was recognized and a fault tolerance related solution was proposed. While this paper lays some ground work for designing this solution much work and refinement is needed before this solution would be appropriate for deployment in a production environment.

References

[1] Abdalla, M., Bellare, M. and Neven, G. (2008). A Provable-Security Treatment of Robust Encryption. Cryptology ePrint Archive, Report 2008/440.

[2] Ateniese, G., Fu, K., Green, M. and Hohenberger. (2006). Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. ACM Transactions on Information and Systems Security. 9:1, 1-30.

[3] Boyd, C. and Mathuria, A. Protocols for Authentication and Key Establishment. Springer, Berlin.

[4] Cavaiani, C. and Alves-Foss, J. (1996). A Mutual Authentication Protocol with Key Distribution in a Client/Server Environment. http://www.acm.org/crossroads/xrds2-4/authen.html.

[5] European Space Agency. (2006). Industrial Policy Committee: List of Intended Invitations to Tender. ESA/IPC(2006)11.Rev.11.

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

[7] Halderman , J. et al. (2009). Lest we Remember: Cold-boot Attacks on Encryption Keys. Communications of the ACM. 52:5.

[8] Hong-Wei, Z., Shi-Hui, P. and Ping-Ping, L. (2007). Distributed Key Establishment Scheme in Wireless Sensor Networks. Eight ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Qingdao, China, 2: 268-273.

[9] JHU-MIT Proxy Re-cryptography Library. (2007). http://spar.isi.jhu.edu/prl/.

[10] Li, J. and Garuba, M. (2008). Encryption as an Effective Tool in Reducing Wireless LAN Vulnerabilities. Proceedings of the Fifth International Conference on Information Technology: New Generations, 557-562.

[11] Moore, F. (2005). Preparing for Encryption: New threats, Legal Requirements Boost Need for Encrypted Data. Computer Technology Review, August-September.

[12] Shannon, Claude (1949). "Communication Theory of Secrecy Systems". *Bell System Technical Journal* **28** (4): 656–715.

[13] Simons, B. (1996).Testimony to the US Senate Regarding S.1726, the “Promotion of Commerce On-line in the Digital Era.

[14] Sun MicroSystems. (2007). Encryption Strategies: The Key to Controlling Data, Sun Services White Paper, August, http://www.sun.com/storage/docs/Encryption_Strategies_wp.pdf.

[15] Tseng, Y. (2003). A Scalable Key Management Scheme with Minimizing Key Storage for Secure Group Communications. International Journal of Network Management. 13:6, 419-425.

[16] US patent 6026163, http://www.freepatentsonline.com/6026163.html.

[17] Yang, H. (2007). Cryptography Tutorials: Herong's Tutorial Notes, http://www.herongyang.com/crypto/.

F(x) ServerID Iteration

$$ {}_{x}^{1}\{n_x, n_{(x+(n-i)+1)}\} $$

Where:

- x = Server from 1 to n
- n = Number of key servers
- i = interval
- $i <= n$

$$ {}_{x}^{1}\{n_x, n_{(x+1)}\} $$

Where:

- x = Server from 1 to n
- n = Number of key servers
- i = interval
- $i > n$

F(x) Server Key Length

$$ n \times i $$

Where:

- n = Number of key servers
- i = interval

F(x) Key Length

$$ n \times (i + 1) $$

Where:

- n = Number of key servers
- i = interval

F(x) Server Key Address Position

$$ {}_{y}^{x}\{(n-1) \bullet i + x, (n-1) \bullet i + (x+1), \ldots, (n-1) \bullet i + x + y\} $$

Where:

- x = Key position from 1 to y
- n = Number of key servers
- i = interval
- $y = (n \bullet i) - 1$

Appendix A: Process A2 Formulas.