Impact Factor Value 4.046 e-ISSN: 2456-3463
National Conference on Quality Up-gradation in Engineering, Science and Technology (NC-QUEST018)
Organized by College of Engineering and Technology, Dhanamgaon Rly-444709
International Journal of Innovations in Engineering and Science, Vol. 3, No.6, 2018
www.ijies.net

Key Exposure for Cloud Computing Stored Data

Ms. Manisha A. Pradhan[1], Ms. Prajakta U. Bakhade[2]

[1] *Dr. Amit K. Gaikwad*
Dhamangaon Education Society's
College of Engineering and Technology,
Dhamangaon Rly

Abstract: *Recent news reveal a powerful attacker which breaks data confidentiality by acquiring cryptographic keys, by means of coercion or backdoors in cryptographic software. Once the encryption key is exposed, the only viable measure to preserve data confidentiality is to limit the attacker's access to the ciphertext. This may be achieved, for example, by spreading ciphertext blocks across servers in multiple administrative domains thus assuming that the adversary cannot compromise all of them. Nevertheless, if data is encrypted with existing schemes, an adversary equipped with the encryption key, can still compromise a single server and decrypt the ciphertext blocks stored therein. In this paper, we study data confidentiality against an adversary which knows the encryption key and has access to a large fraction of the ciphertext blocks. To this end, we propose Bastion, a novel and efficient scheme that guarantees data confidentiality even if the encryption key is leaked and the adversary has access to almost all ciphertext blocks. We analyze the security of Bastion, and we evaluate its performance by means of a prototype implementation.*

Keywords: Key exposure, data confidentiality, dispersed storage.

I. INTRODUCTION

The world recently witnessed a massive surveillance program aimed at breaking users' privacy. Perpetrators were not hindered by the various security measures deployed within the targeted services. For instance, although these services relied on encryption mechanisms to guarantee data confidentiality, the necessary keying material was acquired by means of backdoors, bribe, or coercion. If the encryption key is exposed, the only viable means to guarantee confidentiality is to limit the adversary's access to the ciphertext, e.g., by spreading it across multiple administrative domains, in the hope that the adversary cannot compromise all of them. However, even if the data is encrypted and dispersed across different administrative domains, an adversary equipped with the appropriate keying material can compromise a server in one domain and decrypt ciphertext blocks stored therein. In this paper, we study data confidentiality against an adversary which knows the encryption key and has access to a large fraction of the ciphertext blocks. The adversary can acquire the key either by exploiting flaws or backdoors in the key-generation software, or by compromising the devices that store the keys (e.g., at the user-side or in the cloud). As far as we are aware, this adversary invalidates the security of most.

II. COMPARISON TO EXISTING SCHEMES

In what follows, we briefly overview several encryption modes and argue about their security (according to Definitions 1 and 3) and performance when compared to Bastion.

CPA-encryption modes

Traditional CPA-encryption modes, such as the CTR mode, provide *ind* security but are only 1*CAKE* secure. That is, an adversary equipped with the encryption key must only fetch two ciphertext blocks to break data confidentiality.

CPA-encryption and secret-sharing

Another option is to rely on the combination of CPA secure encryption modes and secret-sharing. If the file f is encrypted and then shared with an n-out-of-n secret-sharing scheme (denoted as "encrypt then secret share" in the following), then the construction is clearly $(n-1)$*CAKE* secure and is also *ind*secure. However, secret-sharing the ciphertext comes at considerable storage costs; for example, each share would be as large as the file f using a perfect secret sharing scheme—which makes it impractical for storing large files. Secret-sharing the encryption key and dispersing its shares

across the storage servers alongside the ciphertext is not secure against an *ind*-adversary. Indeed, if the adversary can access all the storage servers and download all ciphertext blocks, the adversary may as well download all key shares and compute the encryption key. We assume that the CTR encryption routine starts with a random IV that is incremented at every block encryption.

AON encryption

Recall that an AONT is not an encryption scheme and does not require the decrypt or to have any secret key. That is, an AONT is not secure against an *ind*-adversary which can access all the ciphertext blocks. One alternative is to combine the use of AONT with standard encryption. Rivest suggests to pre-process a message with an AONT and then encrypt its output with an encryption mode. This paradigm is referred to in the literature as AON encryption and provides (n−1)*CAKE* security. Existing AON encryption schemes require at least two rounds of block cipher encryption with two different keys. At least one round is required for the actual AONT that embeds the first encryption key in the pseudo-ciphertext. An additional round uses another encryption key that is kept secret to guarantee CPA-security. However, two encryption rounds constitute a considerable overhead when encrypting and decrypting large files. In Appendix A, we describe possible ways of modifying the AONTs to achieve *ind* security and (n−1)*CAKE* security without adding another round of block cipher encryption, and we discuss their shortcomings. Clearly, these solutions are either not satisfactory in terms of security or incur a large overhead when compared to Bastion and may not be suitable to store large files in a multi-cloud storage system.

III. Literature Review

M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. E. Spafford [1] proposed Secure outsourcing of scientific computations. A customer who needs computations done but lacks the computational resources (computing power, appropriate software, or programming expertise) to do these locally, would like to use an external agent to perform these computations. This currently arises in many practical situations, including the financial services and petroleum services industries. The outsourcing is secure if it is done without revealing to the external agent either the actual data or the actual answer to the computations. The general idea is for the customer to do some carefully designed local preprocessing (disguising) of the problem and/or data before sending it to the agent, and also some local post processing of the answer returned to extract the true answer.

C. Wang, K. Ren, and J. Wang [2] proposed Secure and practical outsourcing of linear programming in cloud computing. Cloud computing enables customers with limited computational resources to outsource large-scale computational tasks to the cloud, where massive computational power can be easily utilized in a pay-per-use manner. However, security is the major concern that prevents the wide adoption of computation outsourcing in the cloud, especially when end-user's confidential data are processed and produced during the computation.

IV. System Design

We consider a multi-cloud storage system which can leverage a number of commodity cloud providers (e.g., Amazon, Google) with the goal of distributing trust across different administrative domains. This "cloud of clouds" model is receiving increasing attention nowadays with cloud storage providers such as EMC, IBM, and Microsoft, offering products for multicloud systems. In particular, we consider a system of s storage servers S1, . . . , Ss, and a collection of users. We assume that each server appropriately authenticates users. For simplicity and without loss of generality, we focus on the read/write storage abstraction of which exports two operations: write(v)This routine splits v into s pieces {v1, . . . , vs} and sends hv ji to server Sj , for j ∈ [1 . . . s]. read(·) The read routine fetches the stored value v from the servers. For each j ∈ [1 . . . s], piece vj is downloaded from server Sj and all pieces are combined into *v*. We assume that the initial value of the storage is a special value ⊥, which is not a valid input value for a write operation.

Algorithm

Algorithm 1: Encryption in Bastion.

1: **procedure** Enc(K, x = x[1] . . . x[m])
2: n = m + 1
3: y′[n] {0, 1}l ◁ y’[n] is the IV for CTR
4: **for** i = 1 . . . n − 1 **do**
5: y′[i] = x[i] _ FK(y′[n] + i)
6: **end for**
7: t = 0l
8: **for** i = 1 . . . n **do**
9: t = t _ y′[i]
10: **end for**
11: **for** i = 1 . . . n **do**
12: y[i] = y′[i] _ t
13: **end for**
14: **return** y ◁ y = y[1] . . . y[n]

15: **end procedure**

Algorithm 2 : Decryption in Bastion.

1: **procedure** Dec(K, y = y[1] . . . y[n])
2: t = 0
3: **for** i = 1 . . . n **do**
4: t = t _ y[i]
5: **end for**
6: **for** i = 1 . . . n **do**
7: y′[i] = y[i] _ t
8: **end for**
9: **for** i = 1 . . . n − 1 **do**
10: x[i] = y′[i] _ F−1
κ (y′[n] + i)
11: **end for**
12: **return** x ◅ x = x[1] . . . x[n − 1]
13: **end procedure**

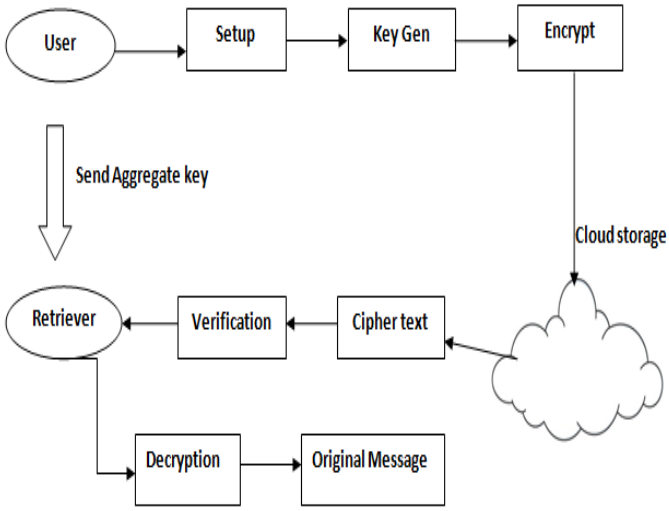

Fig. 1: Phases of Key Generation

Therefore, we are only left to show that the linear transformation computed in lines 7-14 of Algorithm 1 is correctly reverted in lines 2-8 of Algorithm 2. In other words, we need to show that t =Li=1..n y[i] (as computed in the decryption algorithm) matches t =Li=1..n y′[i] (as computed in the encryption algorithm).Recall that t can be computed as follows:

t =Mi=1..ny[i]=Mi=1..n(y′[i] ⊕t)=Mi=1..ny′[i]

i=1..ny′[i]!!=Mi=1..n□ □ Mj=1..n§=iy′[j]□ □ =Mi=1..ny′[i]

Notice that the last step holds because n is even and therefore each y′[j] is XORed for an odd number of times.

We point out that Bastion is not restricted to the CTR encryption mode and can be instantiated with other ind-secure block cipher (and stream ciphers) modes of encryption (e.g., CBC, OFB). To interface with our cloud storage model described in Section 3.1, we assume that each user encrypts the data using Bastion before invoking the write() routine. More specifically, let Enc(K, ・),Dec(K, ・) denote the encryption and decryption routines of Bastion, respectively. Given encryption key K and a file f, the user computes v ← Enc(K, f) and invokes write(v) in order to upload the encrypted file to the cloud. In this setting, key K remains stored at the user's machine. Similarly, to download the file from the cloud, the user invokes read(・) to fetch v and runs f ← Dec(K, v) to recover f.

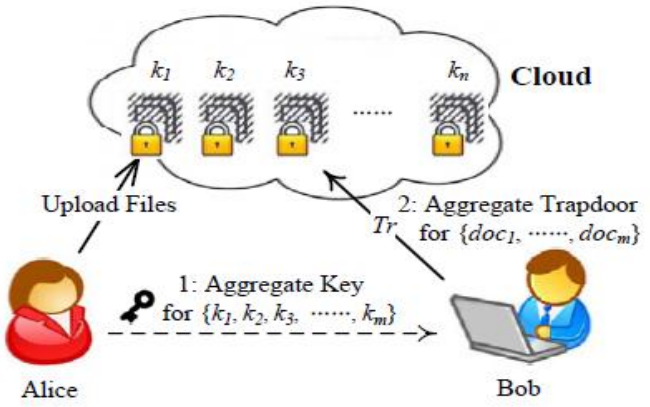

Fig 2: Example

V. CONCLUSION

We then proposed Bastion, a scheme which ensures the confidentiality of encrypted data even when the adversary has the encryption key, and all but *two* ciphertext blocks. Bastion is most suitable for settings where the ciphertext blocks are stored in multi-cloud storage systems. In these settings, the adversary would need to acquire the encryption key, and to compromise all

servers, in order to recover any single block of plaintext. We analyzed the security of Bastion and evaluated its performance in realistic settings. Bastion considerably improves (by more than 50%) the performance of existing primitives which offer comparable security under key exposure, and only incurs a negligible overhead (less than 5%) when compared to existing semantically secure encryption modes (e.g., the CTR encryption mode). Finally, we showed how Bastion can be practically integrated within existing dispersed storage systems.

REFERENCES

[1]. *NEC Corp., "HYDRAstor Grid Storage," http://www.hydrastor.com.*
[2]. *A. Shamir, "How to Share a Secret?" in Communications of the ACM, 1979, pp. 612–613.*
[3]. *D. R. Stinson, "Something About All or Nothing (Transforms)," in Designs, Codes and Cryptography, 2001, pp. 133–138.*
[4]. *StorSimple, "Cloud Storage," http://www. storsimple.com/.*