Securing Cloud Data under Key Exposure Using OTP

K.Gobinathan[1], S.Kiran Kumar[2] ,A.Mohammed Yaseen[3], M.Abuthathir[4]

1.Assistant Professor,Dept Of CSE,Gnanamani college of Technology,Namakkal-637018.

2.Final Year UG Student, ,Dept Of CSE,Gnanamani college of Technology,Namakkal-637018.
3.Final Year UG Student, ,Dept Of CSE,Gnanamani college of Technology,Namakkal-637018.
4.Final Year UG Student, ,Dept Of CSE,Gnanamani college of Technology,Namakkal-637018.

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 there in.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. We also discuss practical insights with respect to the integration of Bastion
in commercial dispersed storage systems.

1. INTRODUCTION

HE world recently witnessed a massive

T surveillance program aimed at breaking users' privacy. Perpetrators were not hindered by the various security measures deployed within the targeted services [31]. 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. 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. We also discuss practical insights with respect to the integration of Bastion in commercial dispersed storage systems.

cryptographic solutions, including those that protect encryption keys by means of secretsharing (since these keys can be leaked as soon as they are generated).

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. We also discuss practical insights with respect to the integration of Bastion in commercial dispersed storage systems. Existing AON encryption schemes, however, require *at least* two rounds of block cipher encryptions on the data: one preprocessing round to create the AONT, followed by another round for the actual encryption. Notice that these rounds are sequential, and cannot be parallelized. This results in considerable—often unacceptable— overhead to encrypt and decrypt large

files. On the other hand, Bastion requires only one round of encryption—which makes it well-suited to be integrated in existing dispersed storage systems.

We evaluate the performance of Bastion in comparison with a number of existing encryption schemes. Our results show that Bastion only incurs a negligible performance deterioration (less than 5%) when compared to symmetric encryption schemes, and considerably improves the performance of existing AON encryption schemes [12], [26]. We also discuss practical insights with respect to the possible integration of Bastion in commercial dispersed storage systems. Our contributions in this paper can be summarized as follows:

2. Encryption and decryption modes

An encryption mode based on a block] is the conversion of data into a form, called a ciphertext, that cannot be easily understood by unauthorized people. Decryption is the process of converting encrypted data back into its original form, so it is easily understood. Encryption is a mechanism for hiding information by turning readable text into a stream of gibberish in such a way that someone with the proper key can make it readable again.

Encryption helps to you protect the privacy of your messages, documents and sensitive files.In its earliest form, people have been attempting to conceal certain information that they wanted to keep to their own possession by substituting parts of the information with symbols, numbers and pictures, this paper highlights in chronology the history of Cryptography throughout centuries. For different reason humans have been interested in protecting their messages.

Symmetric key encryption algorithms use a single secret key to encrypt and decrypt data. You must securet he key from access by unauthorized agents because any party that has the key can use it to decrypt data. Secret-key encryption is also referred to as symmetric encryption because the same key is used for encryption and decryption. Secret-key encryption algorithms are extremely fast (compared to public-key algorithms) and are well suited for performing cryptographic transformations on large streams of data. The diagram for Secret Key Algorithms below illustrates the mechanism in a well defined way. The illustrates the secret key algorithm. This algorithm uses the same secret key at both sides i.e sender and receiver side. Both the parties required the same shared secret key. There are various symmetric key algorithms that are used now a day.

Brief definitions of the most common encryption techniques are given as follows:

DES (Data Encryption Standard), was the first encryption standard to be recommended by NIST (National Institute of Standards and Technology) DES is . DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm.

3. SYSTEM AND SECURITY MODEL

In this section, we start by detailing the system and security models that we consider in the paper. We then argue that existing security definitions do not capture well the assumption of key exposure, and propose a new security definition that captures this notion.

3.1 System Model

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 [4], [6], [32] with cloud storage providers such as EMC, IBM, and Microsoft, offering products for multicloud systems [15], [16], [29].

In particular, we consider a system of s storage servers $S_1,...,S_s$, 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 [21] which exports two operations:

write(v)This routine splits v into s pieces $\{v_1,...,v_s\}$ and sends hv_ji to server S_j, for $j \in [1...s]$.

read($\cdot$) The read routine fetches the stored value v from the servers. For each $j \in [1...s]$, piece v_j is downloaded from server S_j and all

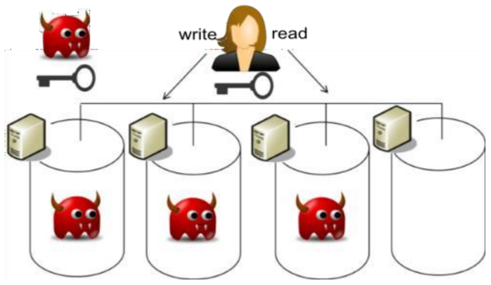


Fig. 1. Our attacker model. We assume an adversary which can acquire all the cryptographic secret material, and can compromise a large fraction (up to all but one) of the storage servers.

pieces are combined into v. We assume that the initial value of the storage is a special value $\perp$, which is not a valid input value for a write operation.

3.2 Adversarial Model

We assume a computationally-bounded adversary A which *can acquire the long-term cryptographic keys used to encrypt the data.* The adversary may do so either *(i)* by leveraging flaws or backdoors in the key-generation software [31], or *(ii)* by compromising the device that stores the keys (in the cloud or at the user). Since ciphertext blocks are distributed across servers hosted within different domains, we assume that the adversary cannot compromise all storage servers (cf. Figure 1). In particular, we assume that the adversary can compromise all but one of the servers and we model this adversary by giving it access to all but λ ciphertext blocks.

Note that if the adversary also learns the user's credentials to log into the storage servers and downloads all the ciphertext blocks, then no cryptographic mechanism can preserve data confidentiality. We stress that compromising the encryption key does not necessarily imply the compromise of the user's credentials. For example, encryption can occur on a specificpurpose device [10], and the key can be leaked, e.g., by the manufacturer; in this scenario, the user's credentials to access the cloud servers are clearly not compromised.

3.3 $(n-\lambda)$-CAKE Security

Existing security notions for encryption modes capture data confidentiality against an adversary which does not have the encryption key. That is, if the key is leaked, the confidentiality of data is broken.

In this paper we study an adversary that has access to the encryption key but does not have the entire ciphertext. We therefore propose a new security definition that models our scenario.

As introduced above, we allow the adversary to access an encryption/decryption oracle and to "see" all but λ ciphertext blocks. Since confidentiality with $\lambda = 0$ is clearly not achievable[1], we instead seek an encryption mode where $\lambda = 1$. However, having the flexibility of setting $\lambda \geq 1$ allows the design of more efficient schemes while keeping a high degree of security in practical deployments. (See Remark 7.)

We call our security notion $(n-\lambda)$ Ciphertext Access under Key Exposure, or $(n-\lambda)$*CAKE*. Similar to [12] , $(n-\lambda)$*CAKE* specifies a block length l such that a ciphertext y can be written as $y = y[1]...y[n]$ where $|y[i]| = l$ and $n > 1$.

$(n-\lambda)$ *CAKE*
ExpQ (A,b) $a \leftarrow K(1^k)$ 1 $x_0,x_1,state$ $\leftarrow$
$A^{EFa,Fa}-$ *(find)* $yb \leftarrow EFa,Fa-1(xb)$
1 $b' \leftarrow AYb,EFa,Fa-$ $(guess,state)$

The adversary has unrestricted access to $E^{Fa,Fa-1}$ in both the "find" and "guess" stages. On input j, the oracle Y_b returns $y_b[j]$ and accepts up to $n-\lambda$ queries. On the one hand, unrestricted oracle access to $\mathcal{E}^{F_a,F_a^{-1}}$captures the adversary's knowledge of the secret key. On the other hand, the oracle Y_b models the fact that the adversary has access to all but λ ciphertext blocks. This is the case when, for example, each server stores λ ciphertext blocks and the adversary cannot compromise all servers. The advantage of the adversary is defined as:

$$Adv_{\Pi}^{(n-\lambda)CAKE} \quad (n-\lambda)CAKE$$

$$(A) = Pr[\mathbf{Exp}_{Q} \quad (A,1) = 1] -$$

$$Pr[\mathbf{Exp}(Q\ n-\lambda)CAKE\ (A,0) = 1]$$

Definition 3. An encryption mode = (K,E,D) is

$(n-\lambda)$*CAKE* secure if for any p.p.t. Q adversary A, we have $Adv^{(}{}_{Q}{}^{n-\lambda)CAKE}(A) \leq \varrho$, where ϱ is a negligible function in the security parameter.

Definition 3 resembles Definition 2 but has two fundamental differences. First, $(n-\lambda)$*CAKE* refers to a keyed scheme and gives the adversary unrestricted access to the encryption/decryption oracles. Second, $(n-\lambda)$*CAKE* relaxes the notion of all-or-nothing and parameterizes the number of ciphertext blocks that are not given to the adversary. As we will show in Section 4.2, this relaxation allows us to design encryption modes that are considerably more efficient than existing modes which offer a comparable level of security.

We stress that $(n-\lambda)$*CAKE* does not consider confidentiality against "traditional" adversaries (i.e., adversaries which do not know the encryption key). Indeed, an *ind*-adversary is not given the encryption key but has access to all ciphertext blocks. That is, the *ind*adversary can compromise all the s storage servers. An $(n-\lambda)$*CAKE*-adversary is given the encryption key but can access all but λ ciphertext blocks. In practice,

1. Any party with access to all the ciphertext blocks and the encryption key can recover the plaintext.

the $(n-\lambda)$*CAKE*-adversary has the encryption key but can compromise up to $s-1$ storage servers. Therefore, properties: Q we seek an encryption mode with the following

1) must be *ind* secure against an adversary which doesQ not know the encryption key but has access to

all ciphertext blocks (cf. Definition 1), by compromising all storage servers.

2) must be $(n-\lambda)$*CAKE* secure against an ad-

Qversary which knows the encryption key but has access to $n-\lambda$ ciphertext blocks (cf. Definition 3), since it cannot compromise all storage servers.

Remark 4. Property 2 ensures data confidentiality against the attacker model outlined in Section 3.2. Nevertheless, we must also account for weaker adversaries (i.e., traditional adversaries) that do not know the encryption key but can access the entire ciphertext — hence, *ind* security. Note that if the adversary which has access to the encryption key, can also access all the ciphertext blocks, then no cryptographic mechanism can preserve data confidentiality.

4. IMPLEMENTATION AND EVALUATION

In this section, we describe and evaluate a prototype implementation modeling a read-write storage system based on Bastion. We also discuss insights with respect to the integration of Bastion within existing dispersed storage systems.

4.1 Implementation Setup

Our prototype, implemented in C++, emulates the read-write storage model of Section 3.1. We instantiate Bastion with the CTR encryption mode (cf. Figure 1) using both AES128 and

Rijndael256, implemented using the libmcrypt.so. 4.4.7 library. Since this library does not natively support the CTR encryption mode, we use it for the generation of the CTR keystream, which is later XORed with the plaintext.

We compare Bastion with the AON encryption schemes of Rivest [26] and Desai [12]. For baseline comparison, we include in our evaluation the CTR encryption mode and the AONTs due to Rivest [26] and

7. Security according to Definition 1 is achieved because the key used to create the AONT is always random, even if the key used to add the outer layer of encryption is fixed.

8. Bastion requires $(n-1)$ XOR operations for the CTR encryption and $2n$ XOR operations for the linear transform.

Desai [12], which are used in existing dispersed storage systems, e.g., Cleversafe [25]. We do not evaluate the performance of secret-sharing the data because of its prohibitively large storage overhead (squared in the number of input blocks). We evaluate our implementations on an Intel(R) Xeon(R) CPU E5-2470 running at 2.30GHz. Note that the processor clock frequency might have been higher during the evaluation due to the TurboBoost technology of the CPU. In our evaluation, we abstract away the effects of network delays and congestion, and we only assess the processing performance of the encryption for the considered schemes. This is a reasonable assumption since all schemes are length-preserving (plus an additional block of l bits), and are therefore likely to exhibit the same network performance. Moreover, we only measure the performance incurred during encryption/encoding, since all schemes are symmetric, and therefore the decryption/decoding performance is comparable to that of the encryption/encoding process.

We measure the peak throughput and the latency exhibited by our implementations w.r.t. various file/block sizes. For each data point, we report the average of 30 runs. Due to their small widths, we do not show the corresponding 95% confidence intervals.

4.2 Evaluation Results

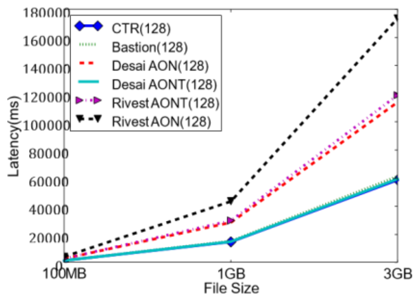
Our evaluation results are reported in Figure 3 and Figure 4. Both figures show that Bastion considerably improves (by more than 50%) the performance of existing $(n-1)$*CAKE* encryption schemes and only incurs a negligible overhead when compared to existing semantically secure encryption modes (e.g., the CTR encryption mode) that are only 1*CAKE* secure.

In Figure 3, we show the peak throughput achieved by the CTR encryption mode, Bastion, Desai AONT/AON, and Rivest AONT/AON schemes. The peak throughput achieved by Bastion reaches almost 72 MB/s and is only 1% lower than the one exhibited by the CTR encryption mode. When compared with existing $(n-1)CAKE$ secure schemes, such as Desai AON encryption and Rivest AON encryption, our results show that the peak throughput of Bastion is almost twice as large as that of Desai AON encryption, and more than three times larger than the peak throughput of Rivest AON encryption.

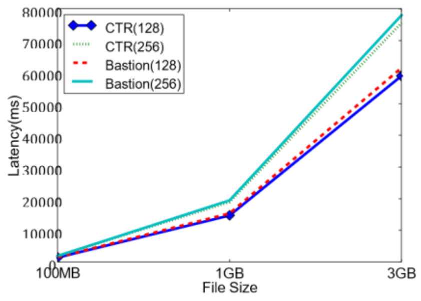
We also evaluate the performance of Bastion, with respect

Fig. 3. Peak throughput comparison. Unless otherwise specified, the underlying block cipher is AES128. Each data point is averaged over 30 runs. Histograms in dark blue depict encryption modes which offer comparable security to Bastion. Light blue histograms refer to encryption/encoding modes where individual ciphertext blocks can be inverted when the key is exposed.

to different block sizes of the underlying block cipher. Our results show that— irrespective of the block size—Bastion only incurs a negligible performance deterioration in peak throughput when compared to the CTR encryption mode. Figures 4(a) and 4(b) show the latency (in ms) incurred by the encryption/encoding routines for different file sizes. The latency of Bastion is comparable to that of the CTR encryption mode—for both AES128 and Rijandael256—and results in a considerable improvement over existing AON encryption schemes (more than 50% gain in latency).



(a) Latency of encryption/encoding for different file sizes.



(b) Latency of encryption/encoding for different block sizes of the underlying block cipher.

Fig. 4. Performance evaluation of Bastion. Each data point in is averaged over 30 runs. Unless otherwise specified, the underlying block cipher is AES-128. CTR(256) and Bastion(256) denote the CTR encryption mode and Bastion encryption routine, respectively, instantiated with Rijandael256.

4.3 Deployment within HYDRAstor

Recall that Bastion preserves data confidentiality against an adversary that has the encryption key as long as the adversary does not have access to two ciphertext blocks. In a multi-cloud storage system, if each server stores at least two ciphertext blocks, then Bastion clearly preserves data confidentiality unless *all* servers are compromised.

In scenarios where servers can be faulty, Bastion can be combined with information dispersal algorithms (e.g., [24]) to provide data confidentiality and fault tolerance. Recall that information dispersal algorithms (IDA), parameterized with t_1,t_2 (where $t_1 \leq t_2$), encode data into t_2 symbols such that the original data can be recovered from any t_1 encoded symbols. In our multicloud storage system (cf. Section 3.1), the ciphertext output by Bastion is then fed to the IDA encoding routine, with symbols of size l bits, and with parameters $t_2 \geq 2s$, $t_1 < t_2$, where s is the number of available servers. Since the output of the IDA is equally spread across the s servers, by setting $t_2 \geq 2s$, we ensure that each server stores at least two ciphertext blocks worth of data. Finally, the encoded symbols are input to the $^{\text{write}}$() routine that distributes symbols evenly to each of the storage servers. Recovering f via the read() routine entails fetching t_1 encoded symbols from the servers and decoding them via the IDA decoding routine. The resulting ciphertext can be decrypted using Bastion to recover file f. By doing so, data confidentiality is preserved even if the key is exposed unless $t = \frac{st_1}{t_2}$ servers are compromised. Furthermore, data availability is guaranteed in spite of $(s - t)$ server failures.

HYDRAstor

We now discuss the integration of a prototype implementation of Bastion within the HYDRAstor grid storage system [13], [23]. HYDRAstor is a commercial secondary storage solution for enterprises, which consists of a back-end architectured as a grid of storage nodes built around a distributed hash table.

To better assess the performance impact of Bastion in HYDRAstor, we evaluated the performance of Bastion in the newest generation HYDRAstor HS8-4000 series system, which uses CPUs with accelerated AES encryption (i.e., the AESNI instruction set). In our experiments, all written data was unique to remove the effect of data deduplication. Results show that the write bandwidth was not affected by the integration of Bastion. The read bandwidth decreased only by 3%. In both read and write operations, the CPU utilization in the system only increased marginally. These experiments clearly suggest that Bastion can be integrated in existing commercial storage systems to strengthen the security of these systems under key exposure, without affecting performance.

5. RELATED WORK

To the best of our knowledge, this is the first work that addresses the problem of securing data stored in multicloud storage systems when the cryptographic material is exposed. In the following, we survey relevant related work in the areas of deniable encryption, information dispersal, all-or-nothing transformations, secretsharing techniques, and leakage-resilient cryptography.

Deniable Encryption

Our work shares similarities with the notion of

"sharedkey deniable encryption" [9], [14], [18]. An encryption scheme is "deniable" if—when coerced to reveal the encryption key—the legitimate owner reveals "fake keys" thus forcing the ciphertext to "look like" the encryption of a plaintext different from the original one—hence keeping the original plaintext private. Deniable encryption therefore aims to deceive an adversary which does not know the "original" encryption key but, e.g., can only acquire "fake" keys. Our security definition models an adversary that has access to the real keying material.

Information Dispersal

Information dispersal based on erasure codes [30] has been proven as an effective tool to provide reliability in a number of cloud-based storage systems [1], [2], [20], [33]. Erasure codes enable users to distribute their data on a number of servers and recover it despite some servers failures.

Ramp schemes [7] constitute a trade-off between the security guarantees of secret sharing and the efficiency of information dispersal algorithms. A ramp scheme achieves higher "code rates" than secret sharing and features two thresholds t_1,t_2. At least t_2 shares are required to reconstruct the secret and less than t_1 shares provide no information about the secret; a number of shares between t_1 and t_2 leak "some" information.

All or Nothing Transformations

All-or-nothing transformations (AONTs) were first introduced in [26] and later studied in [8], [12]. The majority of AONTs leverage a secret key that is embedded in the output blocks. Once all output blocks are available, the key can be recovered and single blocks can be inverted. AONT, therefore, is not an encryption scheme and does not require the decryptor to have any key material. Resch et al. [25] combine AONT and information dispersal to provide both faulttolerance and data secrecy, in the context of distributed storage systems. In [25], however, an adversary which knows the encryption key can decrypt data stored on single servers. This module is opposite of previous module here first to creak the file then collect the binary message from the LSB bits then every 8 bit once the binary bits is convert into ASCII value then these ASCII values are convert into messages **REFERENCES**

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, “Fault-Scalable Byzantine Fault-Tolerant

Services,” in *ACM Symposium on Operating Systems Principles (SOSP)*, 2005, pp. 59–74. [2] M. K. Aguilera, R. Janakiraman, and L. Xu,

“Using Erasure Codes Efficiently for Storage in a Distributed System,” in *International Conference on Dependable Systems and Networks (DSN)*, 2005, pp. 336–345.

[3] W. Aiello, M. Bellare, G. D. Crescenzo, and R. Venkatesan, “Security amplification by composition: The case of doublyiterated, ideal ciphers,” in *Advances in Cryptology (CRYPTO)*, 1998, pp. 390–407. [4] C. Basescu, C. Cachin, I. Eyal, R. Haas, and M. Vukolic, “Robust Data Sharing with Keyvalue Stores,” in *ACM SIGACTSIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2011, pp. 221–222. [5] A. Beimel, “Secret-sharing schemes: A survey,” in *International Workshop on Coding and Cryptology (IWCC)*, 2011, pp. 11–46.

[6] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, “DepSky: Dependable and Secure Storage in a Cloud-ofclouds,” in *Sixth Conference on Computer Systems (EuroSys)*, 2011, pp. 31–46.