Reliable and Secure Distributed Storage of Critical Information

Paper ID: 1569169570 (12 pages)

ABSTRACT

Distributed storage systems aim at providing reliable and secure access to critical data over large networks, by utilizing a distributed collection of parallel storage servers which may be individually unreliable and insecure. Other than the cryptographic approach that relies on secret keys and computational hardness assumptions to provide security, non-cryptographic algorithms have been developed as an efficient way to enhance reliability of distributed storage systems. However, such a non-cryptographic approach puts security at risk and is vulnerable under joint reliability and security breaches. In this paper, we propose a non-cryptographic algorithm for reliable and secure distributed storage, by invoking results on linear error control codes. Our algorithm achieves a combination and tradeoff among three important functionalities: reliability, confidentiality, and integrity, which are collectively measured using a new unifying metric, resilience vector, defined in this paper. A rigorous security and complexity analysis is provided and allows our algorithm to be optimized under different environments. Implementation and simulation show that our algorithm improves both reliability and security of distributed storage systems by three 'nines' at low computation and storage overhead, requiring only bitwise XOR and table lookup operations.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Systems and Software—*distributed systems, performance evaluation*; K.6 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Algorithms, Security, Reliability

Keywords

Distributed Storage System, Security, Reliability, Performance Evaluation

1. INTRODUCTION

With the increasing growth in the number of applications relying on electronic data and online access, distributed storage of critical data over large networks has become an essential component of the distributed computing environment and are evolving into complex, networked and distributed storage models. In order to provide reliable and secure storage over long periods of time, distributed storage systems exploit the degree of spatial freedom by pooling together a large number of parallel storage servers, while each individual server is assumed to be unreliable and insecure. Current application scenarios of distributed storage include storage in online servers, storage in wireless networks, and peer-to-peer storage systems, such as OceanStore [1], Total Recall [2], and DHash++ [4].

In view of security (i.e. confidentiality and integrity, in this paper), distributed storage servers normally have no clear defense boundary and are exposed to threats from the entire network. The growth in network size and connectivity has made distributed storage systems more vulnerable to security breaches. Currently, in distributed storage systems, confidentiality and integrity are provided by performing cryptographic operations that relies on secret keys and computational hardness assumptions. However, such an approach does not provide a perfect solution to security due to several reasons: First, securing the secret keys that are used to access and decrypt the data is of paramount importance, but can be very difficult to achieve in practice, since the keys have to be secured as long as the data is not deleted. Second, cryptographic operations only guarantee conditional security. This means that an attacker can still use methods such as cryptanalysis to break cryptographic protocols and decipher confidential information without the use of secret keys. Third, a digital signature can only be used to check the integrity of the data and provides no correction in case of unauthorized modifications. Even if access control and authentication are employed on the server side, an attacker may still be able to break into storage servers in an attempt to modify the data without making sense of it, by spoofing the identity of a legitimate user or exploiting system weakness. In addition, a malicious server can also replace current files with valid old versions [5]. Therefore, one important question that arises from these observations is the following: knowing that cryptographic methods could fail, how can we ensure the security (i.e. confidentiality and integrity) of distributed storage systems with a non-cryptographic approach?

For reliability, a non-cryptographic approach has been

taken. Since server failures and denial of service attacks (DoS) are very difficult to prevent in practice, a system that embeds strong cryptographic techniques, but does not ensure reliability is of little use. The reliability for storage systems is also referred to as availability in security literatures, where availability, confidentiality, and integrity, are the three cornerstones of security. Ensuring reliability (or availability) typically requires the introduction of redundancy in storage. The simplest form of redundancy is replication, which stores multiple copies of the same data on different servers and is adopted in many practical storage systems. This implies that reliability of distributed storage systems can be improved by increasing the number of storage servers. As an extension of the replication scheme, erasure coding offers better storage efficiency [1, 2, 3]. It divides a critical data into smaller pieces, encode them using a erasure code, and allows the original data to be recovered from any subset of a sufficient number of coded pieces. In another line of work [6, 7, 8, 9, 10, 11], a different approach based on network coding has been proposed to improve storage efficiency and facilitate repair after server failures and data losses.

However, all of these previous non-cryptographic algorithms provide reliability by putting security at risk. Increasing the number of storage servers effectively makes the system more exposed to security breaches, since a malicious attack is more likely to prevail at one of the distributed storage servers and to compromise confidentiality and integrity of the critical data. To address this issue, in this paper, we propose a novel non-cryptographic algorithm for distributed data storage, which achieves both reliability and security at the same time. The algorithm guarantees unconditional security (i.e. absolutely no information about the critical data can be disclosed) if less than a number of storage servers are compromised, and it can also be implemented on top of cryptographic algorithms to provide additional security and reliability in case the cryptographic approach fails.

More precisely, this paper considers a unifying threat model consisting of attacks on reliability, confidentiality, and integrity, respectively. The attack on confidentiality reveals stored server contents to attackers; the attack on integrity modifies data in victim storage servers without being noticed; and the attack on reliability makes storage servers unavailable to legitimate users. As we will show in Section II, this threat model incorporates a wide range of popular attacks in practical distributed storage systems. Further, previous work on non-cryptographic algorithms has only dealt with special subsets of these three attacks: The erasure coding approach [1, 2, 3] and the network coding approach [6, 7, 8, 9, 10, 11] consider only the attack on reliability, while a straightforward extension to general error control coding works for both the attack on integrity and the attack on reliability. In a separate work [12], a scheme based on polynomial evaluation and interpolation is applied to defend against the attack on reliability and the attack on confidentiality. In this paper, we propose a new algorithm for distributed data storage, which is the first one to provide a counter-measure against all three attacks. Our main contributions are summarized as follows.

- Exploring properties of linear error control codes in GF_2, we propose a non-cryptographic algorithm for distributed storage, which we prove achieves both reliability and security at the same time. The algorithm incurs low computation and storage overhand, since it only requires bit-wise XOR operations and simple table lookups. Data losses on distributed storage servers can also be repaired efficiently without retrieving and decoding the critical data.

- For our threat model, we introduce a new metric to quantify the resilience of distributed storage algorithms with respect to the three attacks on confidentiality, integrity, and reliability (or availability). We will show that under realistic assumptions on attack statistics, the probability of secure and reliable data storage is directly determined by the new resilience metric. The number of 'nines'[1] achieved by a distributed storage system can also be obtained in a close-form.

- By varying its parameters, our algorithm illustrates a tradeoff between security and reliability that can be achieved by distributed storage systems. Previous distributed storage algorithms are shown to be special cases, focusing on a subset of confidentiality, integrity, and reliability functionalities. Our analysis and simulation also show that providing all the three functionalities simultaneously is critical for secure and reliable data storage.

- Since distributed storage systems must provide high performance access for hundreds of distributed users concurrently, complexity is an important performance metric. In this paper, we implement our distributed storage algorithm in C and evaluate its performance in terms of computational complexity, storage space and execution time. The proposed algorithm is verified to provide efficient protection against confidentiality, integrity and reliability (availability) attacks.

The rest parts of this paper are organized as follows: In Section 2, we describe the distributed storage problem and its threat model. A new resilience metric for measuring security and reliability is defined. Section 3 presents our non-cryptographic algorithm for distributed storage systems and an algorithm for repairing server data loss. Its resilience measured by our new metric is proven in Section 4. In Section 5, we give an analysis for the security, reliability, and complexity performance of our distributed storage algorithm. In Section 6 and 7 we discuss the implementation of our algorithm, present simulation results, and summarize the paper.

2. ATTACK MODEL AND OUR RESILIENCE METRIC

2.1 Attack Model

In this paper, we focus on the problem of achieving both security and reliability in distributed storage systems using a non-cryptographic approach. A distributed storage system normally contains three components: an encoding algorithm that maps a piece of critical information to a set of messages and stores each message on a separate storage server, a decoding algorithm that retrieves messages from distributed

[1]In security literatures, system availability is often measured by the number of 'nines'. For instance, five 'nines' means 0.99999 availability, which is the gold standard for availability.

servers and derives the original critical information correctly, and a repair algorithm that efficiently recovers a message in case of server data loss.

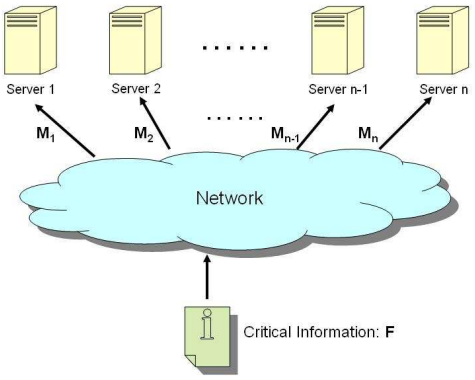

Figure 1: This figure shows the storage of a piece of critical information, denoted by $\mathbb{F}$, on n distributed servers over a network.

Consider a distributed storage system with n separate storage servers, which may be individually unreliable and insecure. To store a piece of critical information denoted by $\mathbb{F}$ distributively, a user employs an encoding algorithm to computes n messages $\mathbb{M}_1, \ldots, \mathbb{M}_n$ from $\mathbb{F}$. Each message is of m bits, contains a fragment of the critical information, and is stored at a separate server. To restore the information $\mathbb{F}$, the user retrieves the messages from storage servers and performs a decoding algorithm to recover $\mathbb{F}$. In our consideration, servers and messages that are protected by secret keys and cryptographic operations are not entirely confidential or reliably re-constructable. An attacker may still be able to obtain the content of some messages or to prevent the user from reconstructing the critical information. In order to provide the three functionalities required for a successful distributed storage, we consider a unifying threat model, which consists of three classes of attacks and corresponding resilience measures defined as follows:

- Attack on confidentiality: Attackers try to obtain some knowledge of the critical information $\mathbb{F}$ by sniffing message content stored in distributed storage servers. This includes a variety of practical attacks, such as identity spoofing, cryptanalysis, and stealing secret keys. To quantify resilience against attacks on confidentiality, we consider a threshold c, such that if no more than c messages (which are stored on c different servers separately) are disclosed, the critical information $\mathbb{F}$ remains completely unknown to attackers.
- Attack on integrity: Attackers break into storage servers and modify the messages to prevent the user from reconstructing the critical information. Another example of such an attack is that a malicious server can also replace current files with valid old versions. Since this attack breaches the integrity of distributed storage servers, we use a threshold d to measure integrity, such that a user can surely derive $\mathbb{F}$ in case of no more than d erroneous messages.
- Attack on reliability: This attack class includes denial of service attacks, traffic jamming attacks, and storage server failures, in which some stored messages become unavailable to their intended users. If the reconstruction of critical data $\mathbb{F}$ is guaranteed given any subset of no less than $n - e$ messages, we refer to the threshold e as the resilience for reliability.

Under our threat model, attackers are able to perform any combination of the three classes of attacks. This implies that a good defense mechanism must take into account the resilience against all attacks jointly. To measure security and reliability for a given distributed storage algorithm with n storage servers, we stack the three individual resilience thresholds into a vector $(c, d, e)_n$ and define a new metric called a resilience vector. Given n storage servers spread over a network, a distributed storage algorithm achieves resilience vector $(c, d, e)_n$, if the critical information $\mathbb{F}$ can be successfully retrieved and decoded, from any subset of $n - e$ messages that contains no more than d errors, while no attacker can derive any information about $\mathbb{F}$ from no more than c disclosed messages.

The resilience vector $(c, d, e)_n$ gives a unifying basis for comparing the reliability and security of different distributed storage algorithms, regardless of their individual threat models and implementation details. Toward this end, it is immediate to see that resilience vectors of previous distributed storage algorithms always contain zero components, since they only deal with a subset of the three attack classes. Thus, these algorithms are vulnerable under the threat model described in this paper. In Section III, we will propose an algorithm that achieves resilience vectors with strictly positive $c, d, e > 0$ at the same time.

Another advantage of our new metric is that for given distributed storage algorithms, the set of achievable resilience vectors forms a 3-dimensional region, which illustrates a tradeoff among confidentiality, integrity, and reliability functionalities provided by the system. Fig.2 illustrates an example of such region, where previous distributed storage algorithms are only able to achieve 2-dimensional planes in the region as they are limited to a subset of the three attacks. In Section V, we will show that for known attack statistics, the probability of reliable and secure data storage can be derived directly from the resilience vectors. The achievable resilience region $\{(c, d, e)_n\}$ serves as an important benchmark fo system designers for selecting distributed storage algorithms and optimizing their system parameters.

3. OUR ALGORITHM FOR DISTRIBUTED STORAGE

3.1 Linear Error Control Coding Background

Linear coding theory focuses on error and erasure correction. A linear code $\mathcal{C}$ over a finite field with q elements is a linear subspace of the field GF_q^n. If $\mathcal{C}$ is an (n, k, s)-code, then it encodes a vector $\vec{x}$ of length k into a codeword $\vec{y} = G^T\vec{x}$ of length n, where G has size $k \times n$ and is a generator matrix for the linear code. The parameter s is the Hamming distance of the linear code, which is equal to the minimal weight (i.e. number of non-zero components) among all non-zero codewords and measures the error correcting capability of the code $\mathcal{C}$. In this paper, we focus on

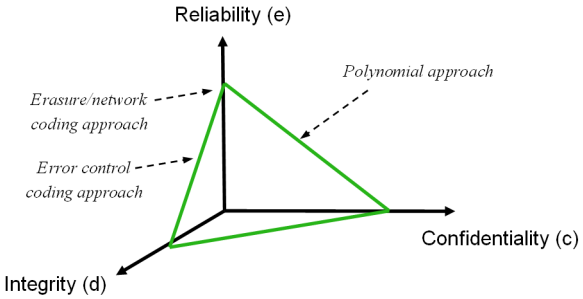

Figure 2: The erasure coding approach [1-3] and the network coding approach [6-11] only consider the attack on reliability. A straightforward extension to general error control coding deals with the attack on reliability and the attack on integrity. The polynomial approach in [12] works under the attack on reliability and the attack on confidentiality. Our proposed algorithm is able to achieve resilience vectors with strictly positive $c, d, e > 0$ at the same time.

binary linear codes in GF_2^n, although most results can be extended to linear codes in general.

To describe the error correcting procedure, we first introduce the concept of dual code and parity check matrix. The orthogonal complement of $\mathcal{C}$, i.e. the set of all vectors in GF_2^n which are orthogonal to every vector in $\mathcal{C}$, is also a subspace and thus another linear code called the dual code of $\mathcal{C}$, denoted by $\mathcal{C}^\perp$. It is easy to see that if $\mathcal{C}$ is an (n, k, s)-code, then $\mathcal{C}^\perp$ is an $(n, n-k, s')$-code, with distance s'. A generator matrix, denoted by H, for $\mathcal{C}^\perp$ has size $(n-k) \times n$ and is known as a parity check matrix for $\mathcal{C}$. A parity check matrix H can be used to recover the codewords of $\mathcal{C}$ because they must be orthogonal to every row of H. Suppose $\vec{\hat{y}} = \vec{y} + \vec{t}$ is a faulty codeword with an error vector $\vec{t}$. Then we can compute $r = H\hat{y} = Hy + Ht = Ht$. The vector r is called the syndrome of $\hat{y}$, which voices information about the error vector $\vec{t}$, since $H\vec{y} = 0$ for all codewords $\vec{y} \in \mathcal{C}$. To recover the original codeword $\vec{y}$ from the faulty codeword $\vec{\hat{y}}$, we only need to store a syndrome table containing error patterns addressed by syndromes. In decoding, when $\vec{\hat{y}} = \vec{y} + \vec{t}$ is received, we first calculate the syndrome $\vec{r} = H\vec{\hat{y}}$, look up the syndrome table with index $\vec{r}$ to find $\vec{t}$, and then recover codeword $\vec{y}$ by $\vec{y} = \vec{\hat{y}} - \vec{t}$.

When both error and erasure occur, the following syndrome decoding procedure for binary linear codes is employed in this paper: We first fill the erased coordinates by all zeros and all ones, and compute two different syndromes (i.e. $\vec{r^0}$ and $\vec{r^1}$), respectively. After looking up $\vec{r^0}$ and $\vec{r^1}$ in the syndrome table to obtain two different error vectors $\vec{t^0}$ and $\vec{t^1}$, the one that contains fewer number of errors on non-erased coordinates gives us the correct syndrome that should be chosen. More precisely, if $\vec{r^0}$ (or $\vec{r^1}$ instead) gives less error, then the original codeword can be recovered by inserting zeros (or ones) on the erased coordinates and then abstracting the error vector $\vec{t^0}$ (or $\vec{t^1}$). In coding theory, it has been proven that an (n, k, s)-code is able to correct any e erasures and d errors at the same time, given that $2d + e \leq s - 1$. The following example contains a generator matrix and a parity check matrix for an $(8, 2, 5)$ linear binary code

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

For an input vector $\vec{x} = [1\ 1]^T$, the corresponding codeword is given by $\vec{y} = G^T\vec{x} = [1\ 1\ 1\ 0\ 0\ 1\ 1\ 1]^T$. Now, suppose that the first two bits of $\vec{y}$ are erased and the third bit is flipped, i.e. $\vec{\hat{y}} = [*\ *\ 0\ 0\ 0\ 1\ 1\ 1]^T$. In order to recover the original codeword from $\vec{\hat{y}}$, we compute two syndromes respectively, $\vec{r^0} = [0\ 0\ 0\ 0\ 0\ 1]^T$ and $\vec{r^1} = [0\ 1\ 0\ 0\ 0\ 1]^T$. By looking up the syndrome table for this $(8, 2, 5)$-code, we get $\vec{t^0} = [0\ 0\ 0\ 1\ 1\ 0\ 0\ 0]^T$ and $\vec{t^1} = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]^T$. Since $\vec{t^0}$ contains two errors on non-erased coordinates, while $\vec{t^1}$ contains only one error, we choose all ones on the erased bits in $\vec{\hat{y}}$ and subtract $\vec{t^1}$ from it. This gives us the correct codeword $\vec{y}$. In the next section, we generalize this syndrome decoding method and derive an algorithm for distributed storage. The proposed algorithm not only corrects errors and erasures (i.e. to provide integrity and reliability), but also guarantees confidentiality of critical information.

3.2 A New Algorithm for Distributed Storage

Consider a system with n distributed servers, each of which is able to store m bits of information. In order to check the integrity of critical information $\mathbb{F}$, we concatenate the information $\mathbb{F}$ with its hash value and store $< \mathbb{F}|h(\mathbb{F}) >$ in the distributed storage system. Let f be the size of $< \mathbb{F}|h(\mathbb{F}) >$. Since each server is capable of storing a message of m bits, we divide the information $< \mathbb{F}|h(\mathbb{F}) >$ into $k = \lceil \frac{f}{m} \rceil$ equal-length fragments, denoted by k column vectors $\mathbb{S}_1, \ldots, \mathbb{S}_k$, such that each piece has exactly m bits and is suitable for storage at one distributed server.

The fragments contains $\mathbb{S}_1, \ldots, \mathbb{S}_k$ confidential information about the critical data $< \mathbb{F}|h(\mathbb{F}) >$. To prevent disclosure of the information, instead of applying the error control coding to these fragments directly, we first construct t pseudo-random binary vectors $\mathbb{X}_1, \ldots, \mathbb{X}_t$, each of m bits. These pseudo-random vectors are used to add randomness and freshness to the system. In the next section, we will give a rigorous proof of the confidentiality achieved by our distributed storage algorithm.

Now, we stack the k information fragments and the t random vectors into an information matrix $[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t]$, and apply a $(n+k, t+k, s)$ binary linear error control encoding to each row of this matrix, using the following generator matrix

$$G = \begin{bmatrix} 1 & 0 & \ldots & g_{1,1} & \ldots & g_{1,n} \\ 0 & 1 & \ldots & g_{2,1} & \ldots & g_{2,n} \\ \vdots & \vdots & \ldots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & g_{k+t,1} & \ldots & g_{k+t,n} \end{bmatrix}_{(k+t)\times(k+n)} \quad (1)$$

Note that the first k columns of G are systematic, i.e. the

upper left $k \times k$ submatrix of G forms an identity matrix. Such generator matrices can be easily constructed by performing simple eliminations. Applying the generator matrix (1) to encode the information matrix row by row, we obtain a binary (coded) message matrix $[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot G$ of size $m \times (n+k)$. In our construction, it is easy to see that the first k columns are just the original information fragments $\mathbb{S}_1, \ldots, \mathbb{S}_k$, and thus should be destroyed for confidentiality. For $i = 1, \ldots, n$, we choose message $\mathbb{M}_i$ as the $(k+i)$'th column of the the coded message and send it to server i to be stored there, i.e.

$$[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{M}_1, \ldots, \mathbb{M}_n] = [\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot G,$$

or in a more explicit form,

$$\mathbb{M}_i = (g_{1,i}\mathbb{S}_1) \oplus \ldots \oplus (g_{k,i}\mathbb{S}_k) \oplus (g_{1+k,i}\mathbb{X}_1) \oplus \ldots \oplus (g_{t+k,i}\mathbb{X}_t)\,.$$

The algorithm for storing is summarized as follows:

Algorithm 1 (Storing Phase)

Parameters: k, t, n, G.

Input: information $\mathbb{F}$.

$[\mathbb{S}_1, \ldots, \mathbb{S}_k] \leftarrow [\mathbb{F}, h(\mathbb{F})]$
$[\mathbb{X}_1, \ldots, \mathbb{X}_t] \leftarrow random()$
for $i = 1, \ldots, n$ **do**
  $\mathbb{M}_i \leftarrow 0$
  for $j = 1, \ldots, k$ **do**
    if $g_{j,i} = 0$ **do**
      $\mathbb{M}_i \leftarrow \mathbb{M}_i \oplus \mathbb{S}_j$
    end if
  end for
  for $j = 1, \ldots, t$ **do**
    if $g_{j+k,i} = 0$ **do**
      $\mathbb{M}_i \leftarrow \mathbb{M}_i \oplus \mathbb{X}_j$
    end if
  end for
  send $\mathbb{M}_i$ to server i
  delete $\mathbb{M}_i$
end for
delete $\mathbb{F}$, $[\mathbb{S}_1, \ldots, \mathbb{S}_k]$, $[\mathbb{X}_1, \ldots, \mathbb{X}_t]$
:END

To retrieve the critical information from servers, the user needs to successfully decode $< \mathbb{F}|h(\mathbb{F}) >$ from the distributively stored messages. To state our algorithm for retrieving phase, without loss of generality, we assume that the last e messages are erased due to attacks on system reliability, while the rest $n-e$ retrieved messages contain d faulty ones due to attacks on system integrity. Let H be a parity check matrix of size $(n-t) \times (k+n)$ for the error control code given by (1). We define an auxiliary $\tilde{H}_{(n+k-t)\times(n-e)}$ by the submatrix of H, consisting of columns $k+1$ to $k+n-e$ in unerased coordinates of H, i.e. $\tilde{H}_i = H_{i+k}$, for $j = 1, \ldots, n-e$. The collective XOR of the other $k+e$ columns of H gives another auxiliary vector $\tilde{r} = H_1 \oplus \ldots \oplus H_k \oplus H_{n-e+1} \ldots \oplus H_n$.

Let $\left[\hat{\mathbb{M}}_1, \ldots, \hat{\mathbb{M}}_{n-e}\right]$ be a matrix of retrieved messages. Each row of the matrix is a valid codeword of the error control code generated by (1), with $k+e$ erasures and d errors. According to the syndrome decoding procedure described in Section 3.1, if we assume that the erased messages are all zero vectors, we can compute a syndrome matrix $R^0 = \tilde{H} \cdot \left[\hat{\mathbb{M}}_1, \ldots, \hat{\mathbb{M}}_{n-e}\right]^T$, where each column R_i^0 for $i = 1, \ldots, m$ is a syndrome vector. On the other hand, if we assume that the erased messages are all one vectors, it is easy to show that the syndrome vector for the i'th codeword is simply $\tilde{r} \oplus R_i^0$ for $i = 1, \ldots, m$. Thus, by looking up the syndrome table and comparing the error vectors corresponding to the two syndromes, we can recover the original codeword matrix $[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{M}_1, \ldots, \mathbb{M}_n]$ from $\left[\hat{\mathbb{M}}_1, \ldots, \hat{\mathbb{M}}_{n-e}\right]$ row by row. This decoding procedure is very efficient, since syndrome vectors can be used to address the syndrome table to facilitate lookups. Further, multiple entries in the syndrome table can be accessed simultaneously to speed up the decoding procedure.

Recall that the critical information $< \mathbb{F}|h(\mathbb{F}) >$ is the first k columns of the codeword matrix $[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{M}_1, \ldots, \mathbb{M}_n]$. This means that for retrieving the critical information, we only need to recover the first k columns of the codeword matrix. Let $\vec{t^0}$ and $\vec{t^1}$ be the two error vectors for syndromes R_i^0 and $\tilde{r} \oplus R_i^0$ respectively, for $i = 1, \ldots, m$. According to Section 3.1, if $\vec{t^0}$ contains less number of errors on the non-erased coordinates, we conclude that the assumption of all zero vectors is correct and $\vec{t^0_{1:k}}$ gives i'th row of $[\mathbb{S}_1, \ldots, \mathbb{S}_k]$. Otherwise, if $\vec{t^1}$ contains less number of errors on the non-erased coordinates, we choose $\mathbf{1} \oplus \vec{t^1_{1:k}}$. In our algorithm, we define a length-$(n+k)$ mask vector A and use a *popcnt* instruction to compute the number of errors on unerased coordinates. Finally, by concatenating $\mathbb{S}_1, \ldots, \mathbb{S}_k$, we obtain the critical information $< \mathbb{F}|h(\mathbb{F}) >$. The hash value $h(\mathbb{F})$ can be used to check the integrity of $\mathbb{F}$. The algorithm for retrieving critical information is summarized below.

3.3 Algorithm for Repairing Server Data Loss

Given the possibility of malicious attacks in distributed storage systems that we consider, messages stored on distributed servers must be continually refreshed as servers fail or leave the system. To repair a message loss, a storage server needs to communicate with existing healthy servers and re-create the same message, such that reliability and security of the distributed storage system can be maintained. In this section, we propose a message repair algorithm that allows server data loss to be repaired without retrieving and decoding the original critical information $\mathbb{F}$, by deriving a lost message from a proper linear combination of existing messages.

According to error control coding theory, the messages $[\mathbb{M}_1, \ldots, \mathbb{M}_n]$ generated by the last n columns of the generator matrix G in our distributed storage algorithm in Section 3.2, forms a valid codeword for an $(n, t+k, s-k)$ error control code, whose generator matrix is obtained by the last n columns of G. As we will prove in the next section, to guarantee reliability and integrity, our distributed storage system requires $s-k > 2$. This implies that the last n columns of G form a generator matrix with a minimum Hamming distance $s-k > 2$. Thus, any single message can be repaired by a proper linear combination of a subset of other messages.

Algorithm 2 (Retrieving Phase)

Parameters: m, k, t, n, H, $Table$.

initial $A \leftarrow 0$, $r \leftarrow 0$, $e \leftarrow 0$, $i \leftarrow 1$
for $j = 1, \ldots, n$ **do**
  if $\hat{\mathbb{M}}_i \leftarrow retrieve(j)$ is TRUE **do**
    $\tilde{H}_i \leftarrow H_{j+k}$
    $A_{i+k} \leftarrow 1$
    $i \leftarrow i + 1$
  else
    $r \leftarrow r \oplus H_{j+k}$
    $e \leftarrow e + 1$
  end if
end for
for $j = 1, \ldots, k$ **do**
    $r \leftarrow r \oplus H_j$
end for

$R^0 \leftarrow \tilde{H} \cdot \left[\hat{\mathbb{M}}_1, \ldots, \hat{\mathbb{M}}_{n-e}\right]^T$
for $i = 1, \ldots, m$ **do**
  $\vec{t^0} \leftarrow Table[R_i^0]$
  $\vec{t^1} \leftarrow Table[R_i^0 \oplus r]$
  if $popcnt(\vec{t^0}\&A) < popcnt(\vec{t^1}\&A)$ **do**
    $[\mathbb{S}_1, \ldots, \mathbb{S}_k]_{i,1:k} \leftarrow \vec{t^0}_{1:k}$
  else
    $[\mathbb{S}_1, \ldots, \mathbb{S}_k]_{i,1:k} \leftarrow 1 \oplus \vec{t^1}_{1:k}$
  end if
end for
$[\mathbb{F}, h] \leftarrow [\mathbb{S}_1, \ldots, \mathbb{S}_k]$
if $h(\mathbb{F}) = h$ **do**
  output $\mathbb{F}$
end if
:END

More precisely, in order to repair message $\mathbb{M}_{i_0}$ that is generated by the $i_0 + k$'th column of G (denoted by G_{i_0+k}), we only need to perform a Gaussian Elimination [15] to find a set of columns in G that are linearly dependent with G_{i_0+k}, i.e. $G_{i_0+k} = G_{i_1+k} \oplus \ldots \oplus G_{i_l+k}$ for some $i_1, \ldots, i_l$ and $l < n$. Then, message $\mathbb{M}_{i_0}$ can be recovered by

$$\begin{aligned}\mathbb{M}_{i_0} &= [\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot G_{i_0+k} \\ &= [\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot (G_{i_1+k} \oplus \ldots \oplus G_{i_l+k}) \\ &= \mathbb{M}_{i_1} \oplus \ldots \oplus \mathbb{M}_{i_{s'-1}}\end{aligned}$$

This repair procedure can be implemented sequentially at servers $i_1, \ldots, i_l$, each of which XORs $\mathbb{M}_{i_0}$ with its own message and forwards the result to the next server. This algorithm for repairing server data loss is distributive and bandwidth efficient. It is summarized as follows.

Algorithm 3 (Repair Phase)

Parameters: k, n, G.

Input: index i_0.

$[i_1, \ldots, i_l] \leftarrow GaussianElimination(G)$
$\mathbb{M}_{i_0} \leftarrow 0$
for $j = 1, \ldots, l$ **do**
  $\mathbb{M}_{i_0} \leftarrow \mathbb{M}_{i_0} \oplus \mathbb{M}_{i_j}$
  delete $\mathbb{M}_{i_j}$
end for
output $\mathbb{M}_{i_0}$
:END

4. RESILIENCE OF OUR ALGORITHM

In this section, we show that the proposed distributed storage algorithm based on linear error control coding can achieve both reliability and security at the same time. In the following, we analyze our algorithm using the resilience vector metric we introduced in Section 2 and then compare it with previous approaches.

4.1 Resilience Performance Proof

THEOREM 1. *If there exists a linear binary error control code $(k+n, k+t, s)$ with dual code $(k+n, n-t, s')$, then the proposed distributed storage algorithm achieves resilience vectors $(c, d, e)_n$ for $c \leq s' - k - 1$ and $2d + e \leq s - k - 1$.*

PROOF. Consider a scenario of distributed storage with c attacks on confidentiality, d attacks on integrity, and e attacks on reliability. According to our threat model and resilience metric defined in Section 2, we need to prove that using the proposed distributed storage algorithm, a user can successfully retrieve the critical information $\mathbb{F}$, while no attacker can have any information about $\mathbb{F}$. Thus, our proof consists of two parts.

To show that the critical information $\mathbb{F}$ can be successfully retrieved, we consider the message-generation procedure in Algorithm 1 by the generator matrix G in (1):

$$\begin{aligned}&[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{M}_1, \ldots, \mathbb{M}_n] \\ &= [\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot G \\ &= [\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot \begin{bmatrix} 1 & \ldots & g_{1,1} & \ldots & g_{1,n} \\ 0 & \ldots & g_{2,1} & \ldots & g_{2,n} \\ \vdots & \ldots & \vdots & \ddots & \vdots \\ 0 & \ldots & g_{k+t,1} & \ldots & g_{k+t,n} \end{bmatrix}\end{aligned} \tag{2}$$

Therefor, each row of the message matrix $[\mathbb{S}_1, \ldots, \mathbb{S}_k, \mathbb{M}_1, \ldots, \mathbb{M}_n]$ is a valid codeword for the $(n+k, k+t, s)$ error control code. According to coding theory, a linear $(n+k, k+t, s)$ error control code can correct up to $\lfloor \frac{s-1}{2} \rfloor$ errors with a syndrome decoding. In the decoding procedure described in Algorithm 2, we choose the $k+e$ erased messages to be all zeros and all ones respectively. Because the error control code is binary, one of the two choices introduces less than $\lfloor \frac{k+e}{2} \rfloor$ new errors, and thus leads to totally $d + \lfloor \frac{k+e}{2} \rfloor$ errors, which can be efficiently corrected by the syndrome decoding in Algorithm 2, if the following is satisfied:

$$d + \lfloor \frac{k+e}{2} \rfloor = \lfloor \frac{2d+k+e}{2} \rfloor \leq \lfloor \frac{s-1}{2} \rfloor \tag{3}$$

This establishes $2d + e \leq s - k - 1$ as a sufficient condition for recovering $\mathbb{S}_1, \ldots, \mathbb{S}_k$ from the retrieved messages.

Next, we show that the critical information $\mathbb{F}$ remains completely unknown to attacks. Without loss of generality, we assume that the first c messages $\mathbb{M}_1, \ldots, \mathbb{M}_c$ are revealed to attackers. According to (2), attackers have c linear equations in the following vector form

$$\begin{aligned}[\mathbb{M}_1, \ldots, \mathbb{M}_c] = \; & [\mathbb{S}_1, \ldots, \mathbb{S}_k] \cdot \begin{bmatrix} g_{1,1} & \cdots & g_{1,c} \\ \vdots & \ddots & \vdots \\ g_{k,1} & \cdots & g_{k,c} \end{bmatrix} \\ & + [\mathbb{X}_1, \ldots, \mathbb{X}_t] \cdot \begin{bmatrix} g_{k+1,1} & \cdots & g_{k+1,c} \\ \vdots & \ddots & \vdots \\ g_{k+t,1} & \cdots & g_{k+t,c} \end{bmatrix} \end{aligned} \tag{4}$$

Because the dual error control code $(k + n, n - t, s')$ has distance s', coding theory shows that any $s' - 1$ columns of the G matrix are linearly independent. For $c + k \leq s' - 1$ as claimed in the statement of Theorem 1, we obtain

$$\begin{aligned} & \text{Rank} \begin{bmatrix} g_{k+1,1} & \cdots & g_{k+1,c} \\ \vdots & \ddots & \vdots \\ g_{k+t,1} & \cdots & g_{k+t,c} \end{bmatrix} \\ & = \text{Rank} \begin{bmatrix} 1 & \ldots & 0 & g_{1,1} & \cdots & g_{1,c} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 1 & g_{k,1} & \cdots & g_{k,c} \\ 0 & \ldots & 0 & g_{k+1,1} & \cdots & g_{k+1,c} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & g_{k+t,1} & \cdots & g_{k+t,c} \end{bmatrix} - k \\ & \geq s' - 1 - k \\ & \geq c \end{aligned} \tag{5}$$

Furthermore, since s' is the distance of the error control code $(k + n, n - t, s')$, it is bounded by $s' \leq (k + n) - (n - t) + 1$. So we derive $t \geq s' - k - 1 \geq c$, which implies that the first matrix in equation (5) has size $t \geq c$, thus its rank can be no more than c. Combining this with the rank condition in equation (5), we conclude that the following matrix is full rank

$$\text{Rank} \begin{bmatrix} g_{k+1,1} & \cdots & g_{k+1,c} \\ \vdots & \ddots & \vdots \\ g_{k+t,1} & \cdots & g_{k+t,c} \end{bmatrix} = c. \tag{6}$$

If we regard $\mathbb{M}_1, \ldots, \mathbb{M}_c$ and $\mathbb{S}_1, \ldots, \mathbb{S}_k$ as coefficients in (4), then equation (4) defines a set of c linear independent equations (due to (6)) with $t > c$ unknowns, i.e. $\mathbb{X}_1, \ldots, \mathbb{X}_t$. Thus, even if attacks successfully intercept c messages, for each possible choice of $[\mathbb{S}_1, \ldots, \mathbb{S}_k]$, there exists $2^{(t-c)m}$ possible matrices $[\mathbb{X}_1, \ldots, \mathbb{X}_t]$ such that equation (4) is satisfied, since each vector $\mathbb{X}_i$ has m bits. In other words, when vectors $\mathbb{X}_1, \ldots, \mathbb{X}_t$ are generated randomly from a uniform distribution independent of the information vectors, for any realization $\hat{S}$, we have

$$\begin{aligned} & \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S} \middle| [\mathbb{M}_1, \ldots, \mathbb{M}_c] = \hat{M}\right\} \\ & = \frac{\text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S}, [\mathbb{M}_1, \ldots, \mathbb{M}_c] = \hat{M}\right\}}{\sum_S \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = S, [\mathbb{M}_1, \ldots, \mathbb{M}_c] = \hat{M}\right\}} \\ & = \frac{\text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S}, [\mathbb{X}_1, \ldots, \mathbb{X}_t] \in \mathcal{X}_{\hat{S},\hat{M}}\right\}}{\sum_S \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = S, [\mathbb{X}_1, \ldots, \mathbb{X}_t] \in \mathcal{X}_{S,\hat{M}}\right\}} \\ & = \frac{\text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S}, [\mathbb{X}_1, \ldots, \mathbb{X}_t] \in \mathcal{X}_{\hat{S},\hat{M}}\right\}}{\sum_S \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = S\right\} \text{Prob}\left\{[\mathbb{X}_1, \ldots, \mathbb{X}_t] \in \mathcal{X}_{S,\hat{M}}\right\}} \\ & = \frac{\text{Prob}\left\{[\mathbb{X}_1, \ldots, \mathbb{X}_t] \in \mathcal{X}_{\hat{S},\hat{M}}\right\} \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S}\right\}}{\text{Prob}\left\{[\mathbb{X}_1, \ldots, \mathbb{X}_t] \in \mathcal{X}_{\hat{S},\hat{M}}\right\} \sum_S \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = S\right\}} \\ & = \text{Prob}\left\{[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S}\right\} \end{aligned} \tag{7}$$

where $\mathcal{X}_{\hat{S},\hat{M}}$ is the set of all $[\mathbb{X}_1, \ldots, \mathbb{X}_t]$ satisfying equation (4) for $[\mathbb{S}_1, \ldots, \mathbb{S}_k] = \hat{S}$ and $[\mathbb{M}_1, \ldots, \mathbb{M}_c] = \hat{M}$. The fifth step of (7) follows from the fact that $\left|\mathcal{X}_{\hat{S},\hat{M}}\right| = 2^{m(t-c)}$ for any choice of $\hat{S}$ and $\hat{M}$. From (7), we conclude that given messages $\mathbb{M}_1, \ldots, \mathbb{M}_c$, all critical information vectors $[\mathbb{S}_1, \ldots, \mathbb{S}_k]$ remains equally likely. Let $I(\cdot)$ be the mutual information function and $H(\cdot)$ be the entropy function. We have

$$\begin{aligned} & I([\mathbb{S}_1, \ldots, \mathbb{S}_k], [\mathbb{M}_1, \ldots, \mathbb{M}_c]) \\ & \quad = H([\mathbb{S}_1, \ldots, \mathbb{S}_k]) - H([\mathbb{S}_1, \ldots, \mathbb{S}_k] \,|\, [\mathbb{M}_1, \ldots, \mathbb{M}_c]) \\ & \quad = 0. \end{aligned}$$

In other words, attacks can obtain absolutely no information about the critical information $\mathbb{F}$, which is unconditionally confidential. □

COROLLARY 1. *At optimum, the proposed distributed storage algorithm is able to achieve a resilience vector $(c, d, e)_n$ with $c + 2d + e = n - k - 2$, when both the primal code $(k + n, k + t, s)$ and the dual code $(k + n, n - t, s')$ are maximum distance separable (MDS).*

REMARK 1. *Theorem 1 and Corollary 1 provide a rigorous reliability and security analysis for the attack resilience of the proposed distributed algorithm. The relationship between the achievable resilient $(c, d, e)_n$ and the algorithm parameters $\{n, t, k, s, s'\}$ are summarized below*

$$\begin{aligned} & 2d + e + c \leq n - k - 2, \\ & 2d + e \leq s - k - 1, \\ & c \leq s' - k - 1 \leq t. \end{aligned}$$

These constraints can be used to estimate the minimum required number of servers to achieve a certain resilience target.

REMARK 2. *From Theorem 1, it is easy to see that reducing k has a positive impact on both reliability and security. Recall that $k = \lceil \frac{f}{m} \rceil$, where f is size of the critical information and m is the storage space at each server. In practice,*

storage at distributed servers are normally inexpensive compared with the requirement for security and reliability. Thus we can have $m > f$ and $k = 1$. This leads to a special case of Theorem 1 and Corollary 1, where the distributed algorithm achieves $2d+e \leq s-2$ and $c \leq s'-2$, with $2d+e+c = n-3$ at optimum. This is the optimal reliability and security that can be achieved by the proposed distributed storage algorithm. In coding theory, the existence of error control code with given parameters has been well studied. For the case of $k = 1$, we list the maximum achievable resilience vectors $(c, d, e)_n$ with $c, d, e > 0$, achievable by our distributed storage algorithm for different numbers of storage servers: In this paper, we only focus on binary error control codes, although all ideas can be extended to codes with higher dimensions. For example, if a linear ternary error control code is used, reliability and security resilience can both be improved, since the code has a larger Hamming distance.

# of Servers	Code Distance		Tolerance $(c, d, e)_m$	
	s	s'	$2d + e$	c
$n = 6$	4	3	2	1
	3	4	1	2
$n = 14$	8	3	6	1
	7	4	5	2
	4	7	2	5
	3	8	1	6
$n = 31$	16	4	14	2
	12	6	10	4
	10	7	8	5
	8	8	6	6
	7	10	5	8
	6	12	4	10
	4	16	2	14

Table 1: Table of maximum achievable resilience vectors $(c, d, e)_n$ with strictly positive $c, d, e > 0$, achievable by our distributed storage algorithm, for $n = 6, 14, 31$ servers.

4.2 Comparison with Previous Algorithms

The erasure coding approach in [1, 2, 3] utilizes linear erasure codes to generate messages $\mathbb{M}_1, \ldots, \mathbb{M}_n$. It is a special class of binary linear error control code and works only for message erasures. The algorithm for storing phase is similar to the Algorithm 1 in this paper, except that no random vectors are used in the construction. a piece of critical information and its hash value $< \mathbb{F}|h(\mathbb{F}) >$ are divided into k equal-length fragments (each of m bits) and encoded into n messages using a generator matrix of an (n, k, s) erasure code, where s is the distance of the erasure code. An (n, k, s) erasure code has a property that any $n - s + 1$ out of the n messages suffice to recover the original critical information. Thus, this approach only deals with attacks on reliability and achieves resilience vectors $(c, d, e)_n$ for $e \leq s - 1$ and $d = c = 0$. Encoding and decoding complexity for erasure codes has been studied extensively, and certain linear erasure codes have encoding and decoding complexity that is linear to n.

For distributed storage applications, the idea of using network coding was first introduced in [6] in a sensor network scenario, and further explored by [7, 8, 9, 10, 11] to improve storage and repair efficiency. a piece of critical information and its hash value $< \mathbb{F}|h(\mathbb{F}) >$ are divided into k equal-length fragments, each represented by an integer S_i between 0 and $2^m - 1$. Instead of binary XOR operation on GF_2, the network coding approach generates messages by weighted linear sum of the information integers over a finite field GF_q with prime $q > 2^m - 1$. Each message M_i for $i = 1, \ldots, n$ is an integer in GF_q and is given by

$$M_i = a_{i,1}S_1 + a_{i,2}S_2 + \ldots + a_{i,k}S_k, \tag{8}$$

where coefficients $[a_{i,1}, \ldots, a_{i,k}]$ are random integers in GF_q. From known results in network coding, it is shown that the set of linear equations in (8) are linearly independent with probability that can be pushed arbitrarily high by increasing the field size [8]. Therefore, any subset of k messages will be sufficient for solving $S_1, \ldots, S_k$ with high probability. The encoding and decoding algorithm of the network coding approach incurs higher computational complexity compared with that of the erasure coding approach, but allow message repair to be performed more efficiently [11]. The network coding approach only provides reliability for distributed storage and achieves resilience vectors $(c, d, e)_n$ for $e \leq n - k$ and $d = c = 0$. It outperforms the erasure coding approach since $s \leq n - k$ for non-trivial binary erasure codes.

Another approach proposed in [12] is able to deal with attacks on confidentiality and attacks on reliability. However it requires km bits of storage on each distributed server and needs to perform expensive polynomial evaluation and interpolation over GF_q, where $q > \max(2^{mk}, n)$ is a prime. The algorithm works as follows: a piece of critical information and its hash value (which have mk bits) can be represented by an integer $S \in GF_q$. We generate a random degree-t polynomial in GF_p:

$$f(x) = S + a_1 x + a_2 x^2 + \ldots + a_t z^t, \tag{9}$$

where $a_i \in GF_p$ for $i = 1, \ldots, t$ are randomly chosen integers. Then n messages are computed by evaluating $f(x)$ at n distinct points for $x = 1, \ldots, n$, i.e.

$$[M_1, M_2, \ldots, M_n] = [f(1), f(2), \ldots, f(n)]. \tag{10}$$

Since the polynomial has degree t, it has been shown in [12] that the original polynomial $f(x)$ can be reconstructed from any $t + 1$ evaluations using polynomial interpolation algorithms, while disclosing no more than t evaluates leaves the information coefficient S completely unknown. This means that this approach achieves resilience vectors $(c, d, e)_n$ for $c \leq t$, $e \leq n - t - 1$ and $d = 0$. The complexity for polynomial evaluation and interpolation using a straightforward algorithm is $o(n^2)$ large integer multiplications, which is prohibitive in practical distributed storage systems.

5. PROBABILITY OF SECURE AND RELIABLE DATA STORAGE

From Theorem 1, using different generator matrices, our distributed storage algorithm is able to achieve resilience vectors $(c, d, e)_n$ for $c \leq s' - k - 1$, $2d + e \leq s - k - 1$, and

Approaches	Resilience vector $(c,d,e)_n$			Storage on each server	Operations required
	c	e	d		
Erasure coding [1-3]	$c=0$	$e \le s-1$	$d=0$	m bits	Bitwise XOR
Network coding [6-11]	$c=0$	$e \le n-k$	$d=0$	m bits	Multiplication in GF_q, $q > 2^m$
Polynomial [12]	$c \le n-t$	$e \le n-t-1$	$d=0$	mk bits	Multiplication in GF_q, $q > 2^{km}$
Our Approach	$c = s' - k - 1$	$2d+e \le s-k-1$		m bits	Bitwise XOR, Table lookup

Table 2: Compare the resilience and basic parameters of our distributed storage algorithm and previous algorithms.

different choices of (s, s'), as illustrated in Table 1. This implies that varying the algorithm parameters allows our distributed storage system to offer various tradeoffs among confidentiality, integrity, and reliability functionalities. Let $\mathcal{R}_n$ denote the set of achievable resilience vectors using n servers. If the statistics of the three attacks on confidentiality, integrity, and reliability can be estimated from past history, we can calibrate our distributed storage algorithm over the region $\mathcal{R}_n$ to maximize the probability of secure and reliable data storage. In this section, we address the question of optimizing our algorithm parameters over $\mathcal{R}_n$ for a given security and reliability target and known attack statistics. Our analysis shows that for even attack probabilities, the security and reliability performance of a distributed storage system is primarily determined by the minimum individual resilience $\min(c,d,e)$ over region $\mathcal{R}_n$.

Since storage servers are distributively deployed over a large network, we assume that each server is attacked with independent probability P_c for attacks on confidentiality, P_d for attacks on integrity, and P_e for attacks on reliability. Thus, a storage server remains unharmed with probability $P_0 = 1 - P_c - P_d - P_e$. In practice, these probability can be collected from historical attack statistics. For a given resilience vector $(c,d,e)_n$, a distributed storage system remains reliable under no more than e attacks on reliability, and remains secure under no more than c and d attacks on confidentiality and integrity. The joint probability of secure and reliable data storage is given by

$$\begin{aligned} & P_n(c,d,e) \\ &= \sum_{i\le c, j\le e, l\le d} \binom{n}{i,j,l} P_c^i P_e^j P_d^l P_0^{n-i-j-l} \\ &= \sum_{i\le c, j\le e, l\le d} \frac{n! P_c^i P_e^j P_d^l P_0^{n-i-j-l}}{i!j!l!(n-i-j-l)!} \end{aligned} \quad (11)$$

When the number of storage servers in a system is fixed, to maximize the the probability $P_n(c,d,e)$, we consider an optimization problem over $\mathcal{R}_n$:

$$\begin{aligned} P_n^* = \max_{c,d,e} \quad & P_n(c,d,e) \\ \text{s.t.} \quad & (c,d,e) \in \mathcal{R}_n \end{aligned} \quad (12)$$

where the optimal solution P_n^* is a function of the server number n and the three attack probabilities (P_c, P_d, P_e). In general, problem (12) can be solved off-line by an exhaustive search over Pareto optimal resilience vectors in $\mathcal{B}_n$ (i.e. vectors lies on the boundary of $\mathcal{R}_n$), before a distributed storage system is deployed. Next, we will provide some approximation of the probability $P_n(c,d,e)$, which allows distributed storage systems to adaptively adjust its resilience operating point, in order to counter any change in the attack pattern.

THEOREM 2. *For a resilience vector $(c,d,e)_n$ and given attack probabilities $P_c, P_d, P_e < \frac{1}{n}$, the probability of secure and reliable data storage is lower bounded by*

$$P_n(c,d,e) \ge 1 - \sum_{x\in\{c,d,e\}} \binom{n}{x+1} \frac{P_x^{x+1}(1-P_x)^{n-x}}{1-nP_x/v} \quad (13)$$

Further, when $P_c = P_d = P_e = P$ and $nP << 1$, we have

$$P_n(c,d,e) \approx 1 - \binom{n}{\min(c,d,e)+1} P^{\min(c,d,e)+1}. \quad (14)$$

Equations (13) in Theorem 2 provides a very close lower bound on the probability $P_n(c,d,e)$. It can be easily computed in real time for selecting a nearly-optimal defence operating point, and it guarantees that the probability of secure and reliable data storage is above the threshold value given by (13). In Figure 3, we plot the probability $P_n(c,d,e)$, its lower bound in (13), and its approximation in (14), for all linear codes of $n = 14$ storage servers and an attack pattern $P_c = P_d = P_e = 1\%$. Theorem 2 is observed to provide a good benchmark for selecting the optimal algorithm parameters, as the optimal defence in this case is achieved by a $(15,5,7)$ linear code and a dual code $(15,10,4)$. The linear code and its dual code give the largest $\min(c,d,e) = 2$ among all codes applicable to $n = 14$ servers. This agrees with our intuition from (13), which states that $P_n(c,d,e)$ is maximized by the linear code with the largest $\min(c,d,e)$ value.

Optimization problem (12) shows that for a given security and reliability requirement P_{target}, there are two different ways to achieve $P_n^* > P_{target}$: increasing the resilience region $\mathcal{R}_n$ by installing more server and reducing the attack probabilities by employing stronger cryptographic mechanisms. In security literatures, system security and reliability are often measured by the number of 'nines', that is.

$$\text{Nines} = \lfloor \log_{10} \tfrac{1}{1-P_n^*} \rfloor. \quad (15)$$

For instance, a distributed storage system achieve five 'nines', if its probability of secure and reliable operation is above 999.99%. Therefore, the number of 'nines' can be regarded as a function of the server number n and the three attack probabilities (P_c, P_d, P_e). For $P_c = P_d = P_e = P$ and $nP << 1$, using equation (14), we derive the number of 'nines' achieved by our distributed storage algorithm

$$\text{Nines} \approx \left[\max_{\mathcal{R}_n} \min(c,d,e) + 1 \right] \log_{10} \frac{1}{P} + C_n, \quad (16)$$

where $\max_{\mathcal{R}_n} \min(c,d,e)$ is the minimum individual resilience maximized on region $\mathcal{R}_n$, and C_n is a proper constant independent of attack probability P.

REMARK 3. *For fixed servers,* $\max_{\mathcal{R}_n} \min(c,d,e)+1$ *is an important resilience index, which determines the number of 'nines' that can be improved by reducing attack probability* P*. Equation (16) shows that if we reduce the attack probability* P *by an order of magnitude, we can improve system security and reliability by approximately* $\max_{\mathcal{R}_n} \min(c,d,e)+1$ *'nines'. Figure 4 plot the required attack probability* P_n^* *for achieving different numbers of 'nines' in a system of* $n=14$ *storage servers. According to our discussion above, linear code* $(15,5,7)$ *and dual code* $(15,10,4)$ *are optimal in this case and gives* $\max_{\mathcal{R}_n} \min(c,d,e)+1=3$*. It is observed in Figure 4 that reducing the attack probability* P *by an order of magnitude results in a linear improvement of about three 'nines'.*

6. COMPLEXITY

We analyze complexity of our distributed storage algorithm in terms of network trips, computation overhead, and storage space. For computation overhead, since we are restricted to linear binary codes in this paper, all operations are performed in Gf_2. We observe that the algorithm consists of four basic operations: binary XOR, table lookup, pseudo-random vectors, and assembly instructions (i.e. *popcnt* and *cmp*). For storage space, a syndrome table, generator and parity check matrices, and auxiliary vectors have to be stored at user terminals locally. We derive the complexity of the proposed algorithm using these metrics.

For the storing phase in Algorithm 1, it is easy to see that t pseudo-random vectors need to be generated and $nm(t+k-1)$ binary XOR operations are used next for constructing n messages. Due to $t \le n$ and $k \le n$, the computation overhead in the storing phase is $o(n)$ pseudo-random vectors and mn^2 binary XOR operations. The required storage space is $m(n+t+k)$ bits (i.e. $o(mn)$) for auxiliary vectors $\mathbb{M}_1,\ldots,\mathbb{M}_n,\mathbb{X}_1,\ldots,\mathbb{X}_t,\mathbb{S}_1,\ldots,\mathbb{X}_k$ and $(k+t)\times(k+n)$ bits (i.e. $o(n^2)$) for the generator matrix.

For the retrieving phase in Algorithm 2, computing the syndrome matrix requires $m(n-t)(n-e-1)+m(n-t)$ binary XOR operations. So, the complexity order is $o(mn^2)$ in total. For the syndrome table lookups, it is clear that $2m$ table lookups are needed. Further, two *popcnt* and one *cmp* instructions are performed for each loop, resulting in a complexity of $3m$. In the retrieving phase, the required storage space for auxiliary vectors is $m(k+n-e)$ bits for messages $\mathbb{S}_1,\ldots,\mathbb{S}_k\hat{\mathbb{M}}_1,\ldots,\hat{\mathbb{M}}_{n-e}$, $m(n-t)$ bits for syndrome matrix R^0, and $n-t+3(n+k)$ bits for vectors $\tilde{r}$, A, $\vec{t^0}$, and $\vec{t^1}$, i.e. totally $o(mn)$ bits. The parity check matrix requires $(2n+k-e)(n-t)$ bits with complexity order $o(n^2)$. Finally, the syndrome table requires $(n+k)2^{n-t}$ bits, because there can be no more than 2^{n-t} different syndromes, each of length $n-t$ bits, and each entry in the syndrome table is an error vector of length $n+k$ bits.

REMARK 4. *The complexity analysis summarized in Table.3 is derived for the worst-case. A practical implementation of the proposed distributed storage algorithm may have much lower complexity by performing the algorithm in parallel. For example, multiple entries in the syndrome table can be accessed at once, such that the complexity for table lookups can be greatly reduced. Similarly, a logic circuit that computes vector XOR operations can be used to replace the bitwise XOR logic gate discussed in this section.*

Complexity Metrics	Storing Phase	Retrieving Phase
Network Trips	1+ACK	2
XORs	$o(mn^2)$	$o(mn^2)$
Random Vectors	$o(n)$	-
Table Lookups	-	$o(m)$
popcnt and *cmp*	-	$o(m)$
Total Computation	$o(mn^2)$	$o(mn^2)$
Syndrome Table	-	$o(n2^{n-t})$
Coding Matrices	$o(n^2)$	$o(n^2)$
Auxiliary Vectors	$o(mn)$	$o(mn)$
Total Storage	$o(mn+n^2)$	$o(mn+n2^n)$

Table 3: Summary of the complexity analysis for our distributed storage algorithm in terms of network trips, computation overhead, and storage space.

REMARK 5. *The proposed distributed storage algorithm employs a syndrome decoding scheme for recovering the critical information* $\mathbb{F}$*. In practice, recovering* $\mathbb{F}$ *can be made much simpler by leveraging the special structure of a given linear code. For example, if Reed-Muller codes are used, syndrome decoding will no longer be necessary for retrieving* $\mathbb{F}$*. Low complexity algorithms for decoding Reed-Muller codes have been extensively studied in the past.*

7. SIMULATIONS

We implemented our distributed storage algorithm in C and evaluate its performance in terms of security and reliability, execution time, and memory requirement. To make a comparison, we also implemented the algorithm based on network coding and the algorithm based on polynomial interpolation discussed in Section 4.2, under the same environment. The distributed storage algorithm using erasure coding is skipped because its resilience vector is upper bounded by that of the network coding algorithm, while its encoding and decoding based on bitwise XOR operations is similar to our algorithm.

For $n=14$ servers and attack probability $P_c=P_d=P_e$, the set of curves in Figure 5 plots the probability P_n^* of secure and reliable data storage, for the three algorithms we implemented. Our algorithm uses a generator matrix of a $(15,5,7)$ linear code and achieves resilience vectors $(c \le 2, 2d+e \le 5)$. According to the result summarized in Table 2, the algorithm based on polynomial interpolation generates a degree-seven random polynomial to achieve resilience vectors $(c \le 7, e \le 6, d=0)$, and the algorithm based on network coding choose $k=1$ to achieve maximum resilience $(c=0, e \le 14, d=0)$. It is observed that when the attack probability at each individual server ranges from 0% to 25%, our distributed storage algorithm exhibits a significant improvement over the other two algorithms. This observation

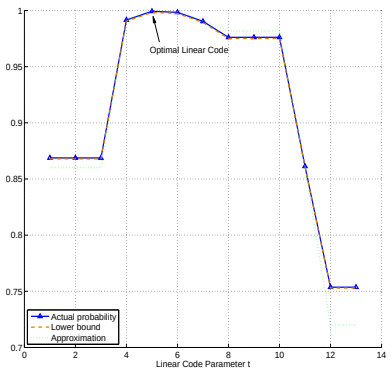

Figure 3: Plot probability of secure and reliable data storage for $P_c = P_d = P_e = 1\%$, $k = 1$, and $n = 14$. Optimal security and reliability is achieved at $t = 5$, by a linear code $(15, 5, 7)$ and a dual code $(15, 10, 4)$. The lower bound of $P_n(c, d, e)$ in (13) and the approximation in (14) both give very good estimation to the actual probability.

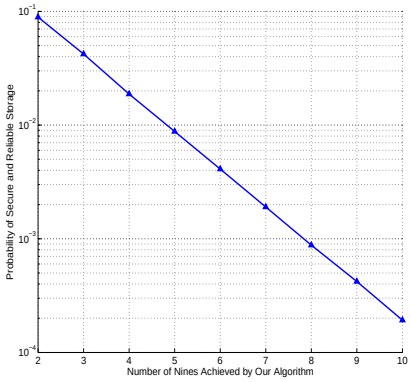

Figure 4: Plot the required attack probability for achieving different numbers of nines in our distributed storage algorithm for $k = 1$ and $n = 14$. It is observed that reducing the attack probability P by an order of magnitude results in a linear improvement of about three 'nines'. Is is consistent with our analysis in Remark 3, as we have $\max_{\mathcal{R}_n} \min(c, d, e) + 1 = 3$.

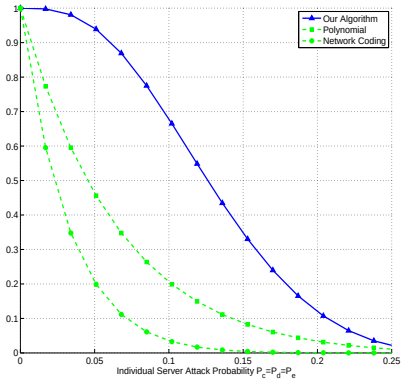

Figure 5: Compare probability of secure and reliable data storage of different algorithms for $n = 14$ servers. Our algorithm achieves resilience vectors $(c \leq 2, 2d + e \leq 5)$, the algorithm based on network coding achieves $(c = 0, e \leq 14, d = 0)$, and the algorithm based on polynomial interpolation achieves $(c \leq 7, e \leq 6, d = 0)$. The ability to deal with all three attacks simultaneously is critical.

substantiates our analysis in Section 5: the security and reliability performance is mainly determined by the minimum individual resilience against the three attacks. The ability to deal with all three attacks at the same time is critical for distributed storage systems.

In the second set of simulations, compare the complexity of the three algorithms we implemented, by plotting the execution time and total memory usage, against the number of servers varying from $n = 5$ to $n = 30$. The critical information $< \mathbb{F}|h(\mathbb{F}) >$ is a randomly generated binary sequence with $m = 1024$ bits. In the implementation our algorithm, for each n, the optimal linear code that maximizes the probability of secure and reliable data storage is chosen by solving the optimization problem (12) off-line. Function $rand()$ in C is used to generate t pseudo-random vectors in Algorithm 1 for storing phase. For the algorithm based on polynomial interpolation, large integers in GF_q are divided into 32-bit segments, each of which can be represented by an *unsigned int* in C, so that large integer multiplications over GF_q are carried out by multiplications and additions of *unsigned int*, segment by segment. For retrieving phase, we implemented Neville's algorithm for polynomial evaluation with complexity $o(n^2)$. For the algorithm based on network coding, we choose $k = 5$ and use a standard Gaussian Elimination for decoding the original critical information from five randomly chosen messages. All algorithms are evaluated on a windows machine with 1.60 GHz CPU and 1.0 GB RAM.

The execution time for storing phase and retrieving phase are plotted in Figure 6 and Figure 7, respectively. For the storing phase in Figure 6, our algorithm exhibits minimum execution time: For as many as $n = 20$ servers, messages can be generated in less than $0.1ms$, since our Algorithm 1 only needs bitwise XOR operation and pseudo-random vector generation. For the retrieving phase in Figure 7, it turns out that syndrome table lookups dominate the computation complexity, and our Algorithm 2 can efficiently decode the critical information in about $10ms$ for $n = 20$ servers. The execution time can be further reduced by using more sophisticated hardware to access multiple entries of the syndrome table at the same time. Figure 8 shows the memory usage of different algorithms measured by bits. Our algorithm requires exponential memory increase as n gets large, since a syndrome table of size $n2^{n-t}$ has to be stored on the user side. However, since practical distributed storage systems normally have less than 20 servers, a memory usage of less than $100K$ bits in our algorithm is almost the same as that of the network coding approach. Our algorithm is able to provide confidentiality, integrity, and reliability functionalities with relatively low computation and storage overhead.

8. CONCLUSION

In this paper, we propose a non-cryptographic algorithm for distributed data storage. The algorithm provides both security and reliability for storing critical information. We analyze and compare our algorithm with previous work under a new resilience metric, and other performance and complexity measures. Our algorithm is the first one to achieve resilience vectors $(c, d, e)_n$ with strictly positive $c, d, e > 0$, thus guarantee confidentiality, integrity, and reliability (availability) at the same time. By optimizing the algorithm parameters under different attack patterns, a significant security and reliability improvement has been observed in both simulation and analytical results. Our algorithm achieves these benefits at low computation and storage over-

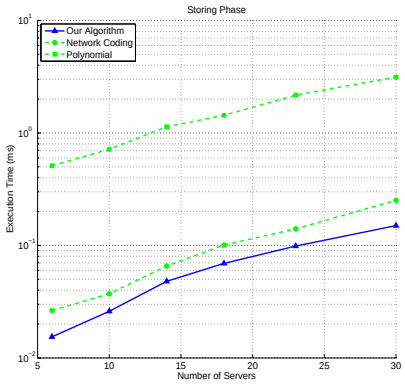

Figure 6: Compare the total execution time for storing phase. Our algorithm exhibits minimum execution time of $0.1ms$ for $n = 20$ servers, since it only needs bitwise XOR operation and pseudo-random vector generation. Algorithm 1 for storing phase is very efficient.

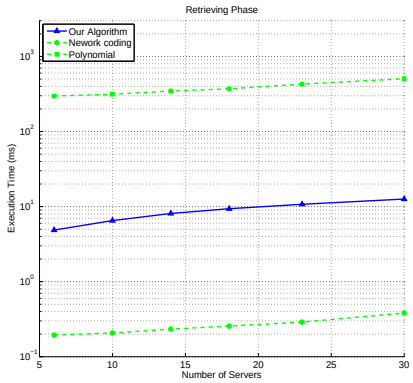

Figure 7: Compare the total execution time for retrieving phase. Our algorithm requires about $10ms$ for decoding with $n = 20$ servers, since the computation complexity is mainly dominated by the number of syndrome table lookups.

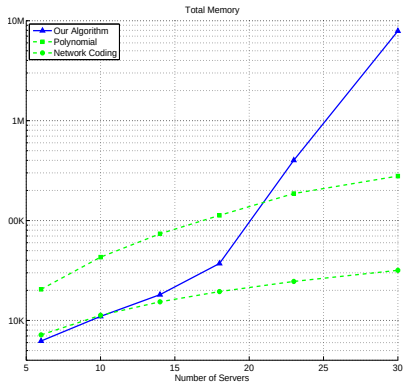

Figure 8: Compare storage complexity by total memory usage. For less than 20 servers (which is an reasonable assumption for a practical system), our algorithm requires about the same complexity as that of the network coding approach.

head, since it only requires $o(mn^2)$ binary XOR operations, $o(m)$ table lookups, and $o(n2^n)$ bits of memory for storing a syndrome table.

9. REFERENCES

[1] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: the OceanStore Prototype", *in Procedings of USENIX File and Storage Technologies*, 2003.

[2] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System Support for Automated Availability Management", *in Procedings of NSDI*, 2004.

[3] H. Xia and A. A. Chien, "RobuSTore: A Distributed Storage Architecture With Robust And High Performance", *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007.

[4] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a DHT for Low Latency and High Throughput", *in Procedings of NSDI*, 2004.

[5] J. Li, M. Krohn, D. Mazires, and D. Shasha, "Secure untrusted data repository (SUNDR)", *in Procedings of OSDI*, 2004.

[6] A. G. Dimakis, V. Prabhakaran, and K. R. Ubiquitous, "Acess to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes", *in Procedings of IEEE/ACM International Symposium on Information Processing in Sensor Networks*, 2005.

[7] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution", *in Proceedings of IEEE Infocom*, 2005.

[8] T. Ho, M. M'edard, R. Koetter, D. Karger, M. Effros, J.Shi, and B. Leong, "A random linear network coding approach to multicast", *Submitted to IEEE Transactions on Infomation Theory*, 2006.

[9] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "On the benefits of random linear coding for unicast applications in disruption tolerant networks", *in Procedings of Second Workshop on Network Coding, Theory, and Applications*, 2006.

[10] D. Wang, Q. Zhang, and J. Liu, "Partial network coding: Theory and application for continuous sensor data collection", *in Procedings of Fourteenth IEEE International Workshop on Quality of Service*, 2006.

[11] A.G. Dimakis, P.B. Godfrey, M.J. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems", *in Proceedings of IEEE INFOCOM*, 2007.

[12] A. Shamir, "How to Share a Secret", *Communications of the ACM*, 1979.

[13] V. Kher and Y. Kim, "Securing Distributed Storage: Challenges, Techniques, and Systems", *in Proceedings of the 2005 ACM workshop on Storage*, 2005.

[14] P. B. Godfrey, Scott Shenker, and Ion Stoica. "Minimizing churn in distributed systems. *in Proceedings of ACM SIGCOMM*, 2006.

[15] A. Bogdanov and M. C. Mertens, "A Parallel Hardware Architecture for fast Gaussian Elimination over GF_2", *in Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006.

APPENDIX

A. PROOF OF COROLLARY 1

PROOF. According to coding theory, for binary error control codes, we have $s+s' = n+k$ when when both the primal code $(k+n, k+t, s)$ and the dual code $(k+n, n-t, s')$ are maximum distance separable. Combining this with Theorem 1, we derive $c+2d+e = s+s'-2k-2 = n-k-2$, which is the desired result. □

B. PROOF OF THEOREM 2

PROOF. Let A_c, A_d, A_e denote the numbers of attacks on confidentiality, integrity, and reliability, respectively. According to our attack model, A_c, A_d, A_e are Binomial-distributed random variables with parameters n and P_c, P_d, P_e. Applying the union bound to the probability $P_n(c,d,e)$, we have

$$\begin{aligned} P_n(c,d,e) &= \text{Prob}\{A_c \le c, A_d \le d, A_e \le e\} \\ &\ge 1 - \sum_{x\in\{c,d,e\}} \text{Prob}\{A_x \ge x+1\} \quad (17) \end{aligned}$$

For each probability $\text{Prob}\{A_x \ge x+1\}$ in (17), we have

$$\begin{aligned} &\text{Prob}\{A_x \ge x+1\} \\ &= \sum_{i=x+1}^{n} \binom{n}{i} P_x^i (1-P_x)^{n-i} \\ &= \binom{n}{x+1} P_x^{x+1}(1-P_x)^{n-x-1}\Bigg[1 + \\ &\qquad \sum_{i=1}^{n-x-1}\left(\frac{P_x}{1-P_x}\right)^i \prod_{j=1}^{i}\frac{n-x-j}{x+1+j}\Bigg] \\ &\le \binom{n}{x+1} P_x^{x+1}(1-P_x)^{n-x-1} \sum_{i=0}^{n-x-1}\left(\frac{P_x(n-x-1)}{(1-P_x)(x+2)}\right)^i \\ &\le \binom{n}{x+1} P_x^{x+1}(1-P_x)^{n-x-1} \frac{1}{1-\frac{P_x(n-x-1)}{(1-P_x)(x+2)}} \\ &= \binom{n}{x+1} P_x^{x+1}(1-P_x)^{n-x} \frac{1}{1-\frac{P_x(n+1)}{x+2}} \\ &\le \binom{n}{x+1} P_x^{x+1}(1-P_x)^{n-x} \frac{1}{1-\frac{nP_x}{x+1}} \\ &\le \binom{n}{x+1} P_x^{x+1}(1-P_x)^{n-x} \frac{1}{1-\frac{nP_x}{x}} \quad (18) \end{aligned}$$

where the forth and the seventh inequality use the assumption $P_x < \frac{1}{n}$, for all $x \in \{c,d,e\}$. Combining (17) and (18), we derive the lower bound for $P_n(c,d,e)$ as claimed in Theorem 2. For $P_c = P_d = P_e = P$ and $nP << 1$, we apply the approximation $1+nP \approx 1$ to the last inequality in (18) and directly derive the desired result in Theorem 2. □