2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Deceptive Secret Sharing

Lei Zhang and *Douglas M. Blough*
School of Computer Science
School of Electrical and Computer Engineering
Georgia Institute of Technology
lz@gatech.edu, doug.blough@ece.gatech.edu

***Abstract*—Confidentiality is a fundamental goal in many security contexts. Deception is another goal in which the intention is to mislead adversaries, for example by planting false information in a system. In this paper, we consider an approach that combines confidentiality and deception using secret sharing, which has traditionally been used strictly for confidentiality purposes. The motivation for this is to protect confidentiality as far as possible while acknowledging that no confidentiality scheme provides perfect protection. If confidentiality is breached and information is accessed by unauthorized individuals, our techniques will reveal, with high probability, only false information. This provides deception on top of the confidentiality provided by ordinary secret sharing. We refer to our approach as "deceptive secret sharing" and we present techniques that work with both XOR secret sharing and Shamir's polynomial-based threshold secret sharing. We provide extensive evaluations of both overhead and security of our techniques and we also show how they provide tunable security that can trade off security and overhead by varying a single parameter of the schemes.**

I. Introduction

Confidentiality has long been one of the most important system security goals. Access control, encryption, secret sharing, and data obfuscation are common techniques that attempt to prevent individuals from unauthorized access and/or use of information. Unfortunately, none of these techniques is perfect and so, despite designers' best efforts, confidentiality is often compromised in today's systems. Confidentiality can be breached even when encryption is deployed, because of errors in integrating crypto protocols with broader security protocols, when encryption keys are protected with weak techniques such as passwords, or a variety of other design weaknesses.

Deception is a less common goal, in which the intention is to mislead adversaries. This can help to detect or monitor attacks and interfere with unauthorized access/use of information. In this paper, we use deception to mean the planting of false information in a system to mislead adversaries. False information might primarily be intended to mislead but it can also serve to detect or track adversaries based on their attempts to use the false information after it has been accessed. A prominent example of the use of false information involved the recent email hack of the policital campaign staff of Emmanuel Macron, now the president of France. Macron's campaign reported that, in their email accounts, they planted "numerous false documents intended to sow doubt and disinformation" [7]. Pointing to definitively false information among leaked emails casts doubt upon the veracity of any potentially damaging emails in the collection.

In this paper, we consider ways in which secret sharing, which has traditionally been used strictly for confidentiality purposes can be used to efficiently provide confidentiality *and* deception. The motivation is to protect confidentiality as far as possible while acknowledging that confidentiality protection cannot be guaranteed. With high probability, if confidentiality is breached, only false information will be revealed with our techniques, thus providing a layer of deception on top of the confidentiality. Efficiency of such an approach is a critical issue since secret sharing and deception, by themselves, can incur high overheads so that naive combinations of the techniques can cause overheads to explode.

Our main contributions are:

- novel and efficient schemes for deception within both XOR-based and polynomial-based secret sharing,
- quantitative security analyses for these schemes and other baseline approaches,
- demonstration that our schemes provide tunable security, which allows overhead vs. security trade-offs, and
- evaluation and comparison of the performance and availability of the proposed schemes through a prototype implementation on CloudLab [4].

II. Background

We use XOR-based secret sharing to illustrate some fundamental ideas underlying our techniques. XOR-based secret sharing is an (n, n) secret sharing scheme, which encodes a secret of length b bits as follows. First, generate $n-1$ random b-bit sequences to form the first $n-1$ shares. Then, make the n^{th} share the XOR of the first $n-1$ shares and the secret. The secret can be reconstructed simply by taking the XOR of all shares. Possession of even $n-1$ shares does not reveal any information about the secret, because any secret value is still possible since any b-bit value can be produced from the known shares with an appropriate choice of the final share.

Suppose we want to plant $m-1$ fake secrets in a system with one real secret using (n, n) XOR-based secret sharing. One approach, which we call NAIVE, would be to simply encode each secret separately, which would result in mn total shares. Since, with XOR-based secret sharing, each share is the same size as the secret, this results in a storage blowup of mn, which is most likely unacceptably large if we want to deceptively share a large number of secrets. If $m = 50$ and $n = 10$, then the storage requirements for NAIVE are 500 times what is needed without secret sharing and deception.

2158-3927/18/$31.00 Â©2018 IEEE
DOI 10.1109/DSN.2018.00053

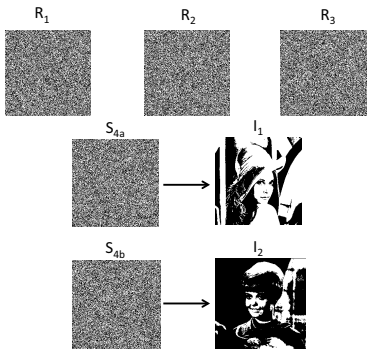

Fig. 1. Different Images XOR Secret Shared with Shares in Common

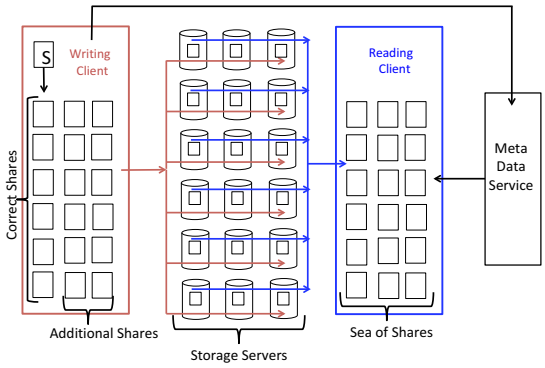

Fig. 2. "Sea of Shares" Approach

XOR-based secret sharing allows for a more efficient way to plant fake secrets. If the real secret is encoded with $n-1$ random shares and the n^{th} share to allow it to be reconstructed from all n shares, different n^{th} shares can be generated to produce different (fake) secrets. Figure 1 shows an example with B/W images. This figure shows two images, three random shares R_1, R_2, R_3, and two 4^{th} shares S_{4a} and S_{4b}, such that the XOR of R_1, R_2, R_3, and S_{4a} yields the first image and the XOR of R_1, R_2, R_3, and S_{4b} yields the second image. With this approach, which we call SIMPLE, m secrets are encoded using $n+m-1$ shares. Thus, the storage blowup of SIMPLE is reduced to approximately $m+n$ (a factor of 60 for $m=50, n=10$ instead of 500 with NAIVE).

SIMPLE reduces the storage blowup by a large factor but we will show later that it is vulnerable to an attack that reduces significantly the computational burden on an adversary to break confidentiality. This discussion illuminates the goals of a deceptive secret sharing scheme:

1) *Overhead:* use as few shares as possible to construct a given number of secrets (one real and the rest fake)
2) *Security:* enforce a very high computational bound on adversaries attempting to recover a secret after compromising some or all of the secret shares

These are competing goals, i.e. reducing overhead by reusing shares can reveal information that reduces the adversary's computational burden. In Sections IV and V, we present two deceptive secret sharing schemes, one using XOR-based secret sharing and the other using Shamir's threshold scheme [18], which allow trade-offs between these competing objectives. It would also be interesting to explore deception within other more efficient secret sharing schemes, e.g. Krawczyk's [13]. In this paper, we lay the foundations for combining secret sharing and deception by exploring the two most commonly used secret sharing schemes and we leave extensions to other schemes for future work.

III. System and Threat Models

A. System Model

We primarily target distributed storage systems such as the cloud, where a large number of storage servers are distributed across one or more data centers. Prior works have suggested using secret sharing to scatter information about a data object across multiple storage servers, e.g. [20], [21]. With (k,n) secret sharing, it is assumed that fewer than k storage servers among the n storing one data object leak data at any time. This might be difficult to achieve, in practice, for several reasons:

- a vulnerability that is common to most or all of the storage servers could lead to a high percentage of the servers being compromised simultaneously,
- given enough time, an adversary could compromise more than the threshold number of servers, or
- a cloud administrator could bypass access control mechanisms and read data from all storage servers.

To address these problems, other work has suggested distributing shares across multiple cloud providers [2]. While this can partially deal with these issues, the user must manage relationships with multiple providers and do cross-domain accesses on each operation. It also does not deter a powerful adversary, such as a nation state, that has the ability to gain access to the systems of multiple providers.

Our approach is equally effective within one cloud provider's domain or across multiple providers. For any object, many shares are written to different servers and only a small number of those shares represent the actual object, whereas the remaining ones serve to confuse even an adversary that can access *all* shares. The high-level approach, which we refer to as the "sea of shares" approach, is depicted in Figure 2. Here, many shares are generated when an object is written. Those shares are spread across a large number of storage servers. A legitimate user must then be able to identify the correct shares out of the "sea of shares" when accessing the object.

Our approach has a deceptive feature, which is not shown in the figure. The shares are generated such that putting some combinations of shares together yields fake but correct-looking

information. This serves to mislead the adversary, because they cannot be sure even if they are able to assemble a valid secret, that it reveals the correct information. We assume correct-looking secrets are easily recognizable, either by a human or an algorithm. This assumption holds for images, natural language text, and many other types of data. This approach provides an extra layer of protection in case the threshold assumption of secret sharing is violated. Even if the adversary compromises enough servers to reconstruct a secret, the probability that the secret is real data is small.

An important question with our approach is how legitimate users can identify the correct shares for an object. While there are different ways to address this, for brevity, we describe just one method here. This method uses shared knowledge by legitimate users in combination with a deterministic mapping method, similar to the approach of [5] for a different problem: users have a shared key, there is a set of random salts for each object, and the hash of the key concatenated with the i^{th} salt gives the location of the i^{th} correct share. The salts are stored as metadata within a separate metadata service. This forms a tripartite security approach, wherein information from three separate entities is required to access (and decode) data: 1) shares from the data storage service, 2) salts from the metadata service, and 3) the shared key from the local device. The need to compromise all of these entities significantly raises the bar for an attacker. In this paper, we focus primarily on data server compromises. Although space limitations prevent a full discussion, this tripartite approach also defeats phishing attacks and limits damage from user device compromises, which are difficult to deal with in secure storage systems.

B. Threat Model and Security Goals

We assume that the adversary can potentially access data from all servers in a distributed storage system and that it has a way to match shares on different servers to detect that they belong to the same object.[1] We will show that our approach is effective even against this extremely powerful adversary. If it is difficult for the adversary to match shares across servers, then our approach will be even more effective.

Assuming that an adversary has access to all shares of an object, both real and fake, our security goals are several:

1) to maximize the computation time required by the adversary to reconstruct one secret,
2) to maximize the computation time required by the adversary to reconstruct all secrets given that they have already reconstructed one secret, and
3) to minimize the probability that any given secret reconstructed by an adversary is the correct secret.

We do not assume perfect security and thus, we allow for the possibility that a determined and resource-powerful adversary might be able to reconstruct a secret or a few secrets. The goal of 1) is to make the reconstruction of *any* secrets as hard as possible. The goal of 2) is to not provide information from one or a few secrets' reconstructions that can make it easy to reveal the remaining secrets. The goal of 3) is to make the probability that an adversary learns the true secret small, even if a few secrets are reconstructed.

[1]This can occur if the adversary is able to eavesdrop network traffic to capture shares as they are written or is present on the servers and performs timing analysis to determine which shares belong together.

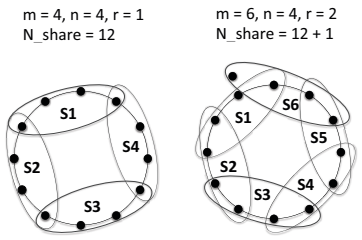

Fig. 3. Examples of Cyclic XOR-Based DSS

IV. Cyclic XOR-Based DSS

A major problem with the SIMPLE scheme described in Section II is that some shares are present in many secrets while other shares are present in only one secret. This imbalance can be exploited by an attacker to learn information about which shares are part of the true secret. In this section, we present and analyze a family of schemes, which we call Cyclic XOR-Based Deceptive Secret Sharing (or CYCLIC for short). With CYCLIC, the number of secrets that a share is part of is nearly identical for every share in the system. Furthermore, CYCLIC has a parameter, which controls the amount of overlap between different secrets that have common shares. Later analysis will show that this parameter can be used to tune the storage overhead and security of CYCLIC. At one extreme when overlap is small, CYCLIC is similar to NAIVE with high storage overhead but strong security.[2] At the other extreme, i.e. large overlap, CYCLIC is similar to SIMPLE with much lower storage overhead but weaker security. In between the extremes, CYCLIC provides a range of options that provide both storage savings, as compared to NAIVE, and security improvements relative to SIMPLE.

A. Description of CYCLIC Scheme

CYCLIC is based on (n, n) XOR secret sharing. Suppose we need m secrets, where one is the real secret and $m - 1$ are fake secrets planted for deception purposes. The scheme is illustrated with examples in Figure 3. We draw all shares on a circle. A set of neighboring shares make up the shares of one secret, as indicated by the oblong groupings. Note that some shares belong to more than one grouping meaning those shares are part of more than one secret. Note also that the number of secrets which each share is a part of is nearly identical for all shares. In the example on the left, each share is part of either one or two different secrets. In the example on the right, most shares are part of exactly two different secrets.

[2]How we measure security will be described later.

We now present CYCLIC in a more general fashion. We denote the overlap between two neighboring groups on the circle by r. In Figure 3, $r = 1$ for the example on the left and $r = 2$ for the example on the right. Most secrets have exactly r shares in common with the previous secret on the circle. For certain values of m, n, and r, there are some secrets that share fewer than r shares with the previous secret, as happens in the example on the right-hand side of Figure 3.

The following describes a method to generate share values in the CYCLIC scheme. For the first secret, we generate $n-1$ random shares and one final share to produce the proper secret. Other secrets contain some shares that belong to the previous secret on the circle and some new shares. For example, the second secret uses the last r shares from the first secret, generates $n - r - 1$ shares randomly, and generates its n^{th} share so as to produce its secret value. This continues until the generated number of secrets is approaching m. As can be seen in the figure, some secrets on the end of the circle wrap around and overlap with the first secret. As long as these wrap-around secrets have at least one share that is not in common with any other secret, their wrap-around parts use the existing shares from Secret 1 (see the example on the left of Figure 3). In some cases, e.g. the example on the right of Figure 3, the secrets that wrap around use some of the shares from the first secret, but generate one additional share at the same position on the circle as one of the first secret's shares. In the example on the right in Figure 3, note that Secret 6 has one share outside the circle in the same position as one of Secret 1's shares. This new share takes the same position on the circle as the corresponding share of Secret 1 and can be thought as a second version of the original share. This share is necessary, because all other shares of Secret 6 are pre-determined by either Secret 5 or Secret 1. The new share is unique to Secret 6 and can be generated so as to produce the proper secret value, despite all other shares being pre-determined.

Since the total number of shares is an important overhead measure of a secret sharing scheme, we now analyze the number of additional shares that are needed when wrapping around. Since a "wrap-around secret" has r shares in common with the previous secret and r shares in common with Secret 1, a new share at an existing position is needed if $r + r \geq n$, meaning that the existing shares would cover all the shares of the wrap-around secret. So, the situation on the right of Figure 3 occurs if and only if $r \geq n/2$ (note that $r < n/2$ in the example on the left of Figure 3 while $r = n/2$ in the example on the right). We denote the number of additional shares by p. When $r \geq n/2$, p is the number of secrets that wrap around on the circle since one additional share is needed for each such secret. Thus, in this case, p is equal to the number of secrets that have their last $n-r$ positions contained in the positions of the first secret, i.e. $p = \lfloor \frac{n}{n-r} \rfloor - 1$.

B. Overhead of CYCLIC Scheme

The overhead of a deceptive secret sharing scheme is determined by the total number of shares that are used to secret share a single object, including shares created for deception purposes. The number of shares corresponds to the storage blow-up, as compared to storing an object without secret sharing and deception. The number of shares also indicates the communication overhead of reading or writing an object. This can be seen from Figure 2 since all shares are written (read) each time an object is written (read). Here, we evaluate the number of shares required by CYCLIC and we compare it against the NAIVE and SIMPLE schemes mentioned earlier.

Given the number of shares making up each secret n, and the overlap r, we can derive the total number of shares, N_{share}, used by the CYCLIC scheme. We start by counting the number of share positions around the circle. Note that each secret has r positions in common with both the secret that follows it and the secret that precedes it. Thus, the total number of share positions is $m(n - r)$. There is one share for each position on the circle and, when $r \geq n/2$, there are p additional shares, one for each wrap-around secret. Thus, $N_{share} = m(n - r) + p = (n - r)m + \lfloor \frac{n}{n-r} \rfloor - 1$. Note that this formula is valid even when no additional shares are used, because from the equation for p, we see that if $r < n/2$ and no additional shares are needed, then $p = 0$.

We assume that m, which is the total number of secrets (real and fake) for one object, is given as a parameter. This parameter measures how deceptive the approach is, i.e. how many fake secrets the true secret is hidden among. With a given m, to reduce the number of shares with CYCLIC we should increase the overlap r, since a larger overlap produces a higher share re-use.

When $r = 0$, the formula for N_{share} given above for CYCLIC simplifies to mn, which is the number of shares for NAIVE (m secrets independently secret shared with n shares). With $r = n-1$, the formula simplifies to $m+n-1$, which is the number of shares for SIMPLE (see Section II). Thus, the number of shares with the two extreme overlaps, $r = 0$ and $r = n - 1$ for CYCLIC, produce the same number of shares as the NAIVE and SIMPLE schemes, respectively.

Figure 4 shows how the overhead of the CYCLIC scheme varies with r, for different values of n and m. In Figure 4a, we fix $n = 15$ and vary m from 50 to 200. In Figure 4b, we fix $m = 50$ and vary n from 15 to 30. In both figures, r is varied across its full range, i.e. from 0 to $n-1$. From the two figures, we see that when the overlap increases, the number of shares decreases linearly. More importantly, depending on the choice of r, the overhead of the CYCLIC scheme varies from the highest possible, which is the same as NAIVE, to the lowest, which is the same as SIMPLE. This provides the flexibility to vary the deceptive secret sharing scheme across a wide range of overheads. In the next subsection, we will see how this flexibility can be used to produce tunable security.

C. Security of CYCLIC Scheme

As mentioned in Section III-B, we are interested both in the amount of work an adversary must do to reconstruct one secret and the amount of work required to reconstruct all secrets given one secret has already been reconstructed. Next,

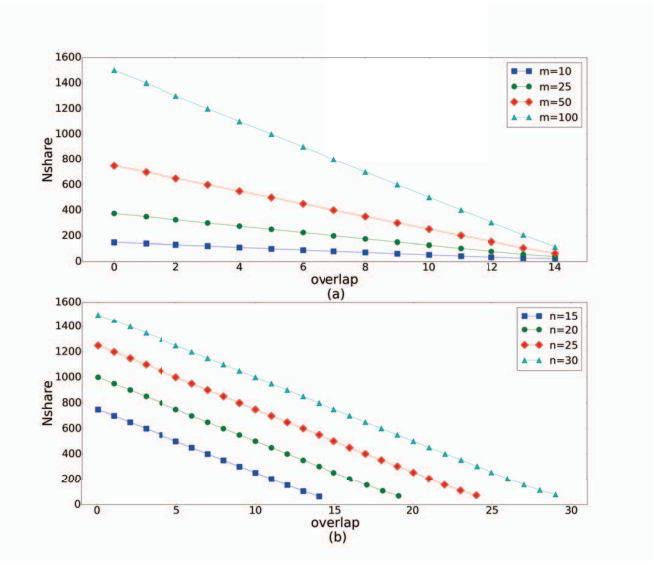

Fig. 4. Number of shares for CYCLIC scheme. We fix $n = 15$ in (a), and $m = 50$ in (b).

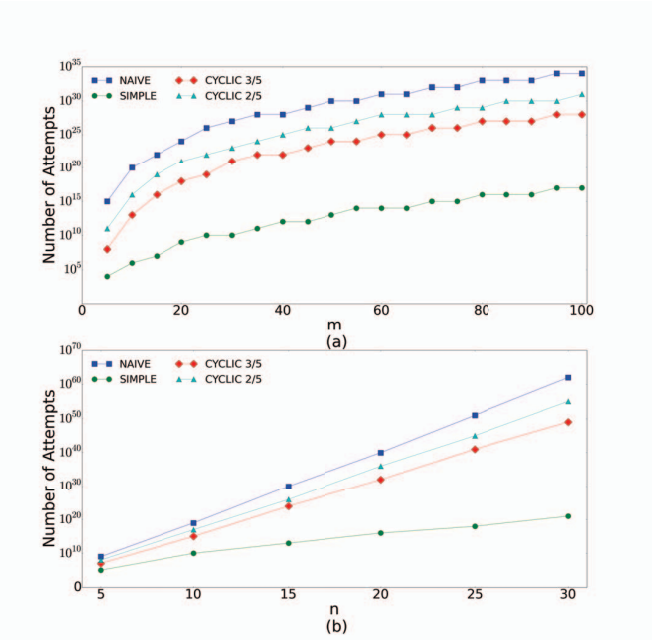

Fig. 5. Number of attempts to reconstruct one secret with probability 0.1. We fix $n = 15$ in (a), and fix $m = 50$ in (b). (log scale on y axes)

we evaluate these quantities for CYCLIC and compare them against SIMPLE and NAIVE.

1) Reconstructing a first secret: As mentioned earlier, we make a worst-case assumption that an adversary has access to all shares of an object (both real and fake). We also do not rely on security by obscurity, in that we assume an adversary knows the parameters of the CYCLIC scheme, i.e. m, n, and r. If N_{comb_all} is the number of possible share combinations that could make up a secret, N_{share} is the total number of shares, and n is the number of shares in one secret, then $N_{comb\_all} = \binom{N_{share}}{n}$. When constructing the first secret, there is no previous information to use, so the adversary can only try different combinations of shares until one produces a correct reconstruction. If an adversary picks combinations at random to try in this manner, then the number of attempts required to reconstruct the first secret is a random variable. Thus, we quantify the security of this step as the probability that an adversary can reconstruct at least one secret within t attempts.

Note that, when an adversary tries a particular combination of shares and fails to reconstruct a valid secret, they get very little information that can help guide future attempts. This is because it is still possible for any of the other combinations to produce a valid secret and the probability of any combination being correct is affected very little by a single incorrect attempt. Given this, we assume that the adversary's strategy before it has reconstructed any valid secret is to choose a combination at random from all unattempted combinations. Under this strategy, the probability of reconstructing at least one secret in t attempts can be calculated as:

$$\begin{aligned} P(t) &= P(\text{at least one secret reconstructed within } t \text{ attempts}) \\ &= 1 - \prod_{i=1}^{t} P(\text{no secret reconstructed on } i^{th} \text{ attempt} \mid \\ &\quad \text{no secret reconstructed on all previous attempts}) \\ &= 1 - \frac{N-m}{N} \times \ldots \times \frac{N-m-t+1}{N-t+1} \end{aligned} \quad (1)$$

In the above analysis, N is used as a shorthand for N_{comb_all}.

A secure secret sharing scheme should guarantee that the adversary has to make a large number of attempts to retrieve one secret with even a small probability. We define N^p_{comb} as the minimum number of attempts an adversary has to make to have a probability of at least p of reconstructing at least one secret. In other words, N^p_{comb} is the minimum t such that $P(t) \geq p$. A higher N^p_{comb} means an adversary must work harder to achieve a given probability of secret reconstruction.

Note that a larger N_{comb_all} will reduce the probability to retrieve one secret and will, therefore, increase N^p_{comb}. N^p_{comb} should be large enough to make the reconstruction process impractical for the adversary. We compare the security levels, quantified by N^p_{comb}, of the three different schemes in Figure 5. Figure 5a and 5b show the number of combinations when $p = 0.1$ and m and n are varied, respectively. We can see that NAIVE and CYCLIC always provide security levels that are significantly higher than SIMPLE.

From the results above, we can see that CYCLIC, with the chosen overlaps, always provides better security than SIMPLE and we know from Section IV-B that it has lower overhead than NAIVE when the overlap is less than $n-1$. This illustrates the trade-off between overhead and security possible with CYCLIC. While SIMPLE has the lowest overhead, in some cases, its security level can simply be too low. We give a concrete example next to illustrate this.

Assume a powerful adversary has a thousand computational nodes, each with a 5 GHz processor. Total computational cycles for the adversary are $1.6 * 10^{20}$ per year. We ignore memory and disk I/O time, and just consider the XOR operation, where each operation takes one clock cycle. For a 256*256 B/W bitmap image and $n = 15$, reconstructing a

	SIMPLE	NAIVE	CYCLIC 9	CYCLIC 13
attempts	10^{13}	10^{30}	10^{24}	10^{17}
time	1 month	10^{16}years	10^{8}years	1000 years
no. shares	64	750	301	106

TABLE I
COMPARISON OF DIFFERENT SECRET SHARING SCHEMES

secret requires $256*256*(15-1) \approx 9*10^5$ XOR operations. Then to guarantee that an adversary cannot reconstruct one secret within one year with a probability of at least 0.1, the scheme needs to provide at least $\frac{1.6 10^{23}}{9*10^5} = 1.8 * 10^{14}$ possible combinations. For $m = 50$, the computation times for SIMPLE, NAIVE, and CYCLIC under these assumptions are shown in Table I ($r = 9$ and $r = 13$ for CYCLIC). For SIMPLE, an adversary has a 1 in 10 chance to retrieve the first secret within a month, but an adversary would need many years to reconstruct at least one secret with probability 0.1 for both NAIVE and CYCLIC.

Table I also shows the numbers of shares generated by these schemes for the same scenario. As expected, SIMPLE produces the fewest shares (less than 1/10 of NAIVE) but this comes at the expense of security. CYCLIC with an overlap of 13 increases the number of shares by about 65% over SIMPLE. However, CYCLIC 13 uses less than 1/7 the number of shares of NAIVE, while providing far greater security than SIMPLE.

2) Reconstructing all secrets with one secret reconstructed: A major weakness of SIMPLE is that if the adversary is able to to reconstruct one secret, even a fake one, they know that $n-1$ of those shares must appear in all other secrets of the same object. This dramatically reduces the number of combinations the adversary must try to reveal all remaining secrets. Even if this happens, we note that the adversary will not be able to tell which is the correct secret but they will know that the correct secret is among the ones that they have found. In some scenarios, this could be highly undesirable. Next, we will show that CYCLIC, with a reasonable choice of overlap, is far more secure in terms of this second security measure.

For both CYCLIC and SIMPLE, once one secret is reconstructed, some information is revealed so that the adversary can reconstruct more secrets in a more efficient way than simply attempting random combinations. Since secrets have overlap of shares, the adversary can focus on secrets that are neighbors of the reconstructed secret in the schemes to more efficiently reconstruct additional secrets. Note, however, that the security of remaining secrets in NAIVE is affected very little by reconstruction of other secrets. This is because there is no share overlap between secrets in NAIVE and so the adversary can still only try random combinations of the remaining shares. Since NAIVE is affected very little by the first secret reconstruction, in this section, we focus on a comparison between CYCLIC and SIMPLE.

We first calculate how many attempts an adversary needs to guarantee they can reconstruct all remaining secrets in the SIMPLE scheme, given one secret. Since the number of shares in SIMPLE is $m + n - 1$, if one secret is known, there are

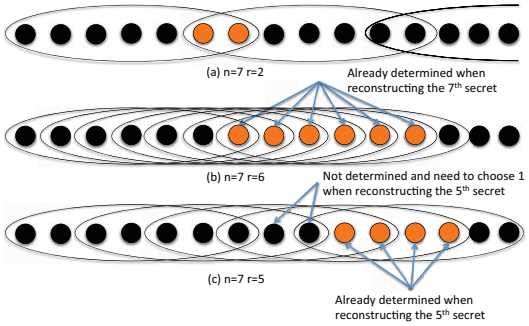

Fig. 6. Different cases for reconstructing all secrets with one leaked secret for the CYCLIC scheme

$m-1$ shares left. We know that $n-1$ of the n shares of the first secret are also part of every other secret. To find those $n-1$ shares only takes n attempts, since the adversary can pick one share from all remaining shares, and use it to replace each share in the reconstructed secret to see if a new secret is revealed. After that, the adversary can easily use the $n-1$ identified common shares to reconstruct the remaining secrets with the last $m-2$ shares. The overall process takes only $n+m-2$ attempts, which is highly insecure.

In the CYCLIC scheme, some information is also leaked from the first secret, but we will show that the adversary still needs a lot of attempts to guarantee reconstruction of the remaining secrets as long as the overlap between secrets is not too high. Since in CYCLIC, a secret shares r shares with its neighboring secret, with one reconstructed secret, the best choice is to try to reconstruct the neighboring secret next, to maximize the information from the first secret.

Let the secret that is known be S_c and let the next secret to be reconstructed be S_n, where S_c and S_n are neighbors in the CYCLIC scheme. The number of remaining shares that are not part of the known secret is $N_{share} - n$. The idea to reconstruct S_n with information from S_c is to find the overlapping shares from S_c, to reduce the number of attempts. Based on the specific parameters for CYCLIC, we divide the process into three cases, as shown in Figure 6.

The first case, e.g. Figure 6a, is $r < \frac{n}{2}$, where the overlapping shares cannot be precisely determined from S_c. For example, 2 out of the 5 non-overlapping shares in the right circled secret will be part of the next secret but the adversary has no way to know which of the 5 are the correct ones. The adversary needs to pick the r overlapping shares from all shares of S_c other than the ones that S_c had in common with the previous secret, which means choosing r shares out of $n-r$ shares (2 out of 5 in the figure). The adversary must put those r shares together with $n-r$ shares from all remaining shares that are not part of any reconstructed secret. Summing these two expressions over all the remaining secrets gives the number of combinations for this case as:

$$\sum_{k=0}^{\frac{N_{share}-n}{n-r}-1} \binom{n-r}{r} \times \binom{N_{share}-n-k(n-r)}{n-r}$$

The second case, e.g. Figure 6b, is $r \geq \frac{n}{2}$, and $n \bmod n - r = 0$. In this case, there is an initial period where the adversary does not know exactly which shares from the previous secret will be reused in the next secret. However, after a small number of consecutive secrets are reconstructed, the common shares can be exactly identified, which reduces the number of combinations that must be attempted. In the Figure 6b example, when reconstructing the 2^{nd} secret, the adversary knows there are 6 shares that overlap with the 1^{st} secret, but he has no idea which 6 of the 7 shares in the 1^{st} secret those are, which means he needs to consider up to $\binom{7}{6}$ combinations to find the correct common shares and, for each of those, he needs to combine them with combinations of the remaining unused shares. However, after the first 6 secrets are revealed, the 6 overlapping shares between the 6^{th} and 7^{th} secrets can be definitively identified since the other share was already known to be part of the 5^{th} and 6^{th} secrets. So, from this point forward, the adversary only needs to consider the combinations of the remaining unused shares each time. Generalizing this logic yields the number of combinations as:

$$\sum_{k=0}^{\lfloor \frac{r}{n-r} \rfloor} \binom{n-k(n-r)}{r-k(n-r)} \times \binom{N_{share}-n-k(n-r)}{n-r} + \sum_{k=\lfloor \frac{r}{n-r} \rfloor+1}^{\frac{N_{share}-n}{n-r}-1} \binom{N_{share}-n-k(n-r)}{n-r}$$

The third case, e.g. Figure 6c, is $r \geq \frac{n}{2}$, and $n \bmod n - r \neq 0$. Similar to the previous case, some shares are determined from S_c, while some shares cannot be determined from S_c. In the figure's example, when reconstructing the 2^{nd} secret, the adversary knows that 5 of 7 shares from the first secret are included. However, when reconstructing the 5^{th} secret, 4 shares are predetermined from the 4^{th} secret, while 1 share cannot be determined (1 of the 2 orange shares should be included and the other should not). Generalizing this yields the following number of combinations:

$$\sum_{k=0}^{\lfloor \frac{r}{n-r} \rfloor} \binom{n-k(n-r)}{r-k(n-r)} \times \binom{N_{share}-n-k(n-r)}{n-r} + \binom{n-r}{r \bmod n-r} \times \sum_{k=\lfloor \frac{r}{n-r} \rfloor+1}^{\frac{N_{share}-n}{n-r}-1} \binom{N_{share}-n-k(n-r)}{n-r}$$

We again give an example to make the number of combinations concrete. As before, assume there are 50 secrets, each has 15 shares, and the overlap is varied from 0 to $n-1$. The results are shown in Figure 7. We can see that when we choose an overlap of 13 as in the previous subsection, the adversary needs about 10^6 attempts to retrieve all secrets. When we choose an overlap of 9, the number of combinations reaches 10^{15}, which is 10 years using the computational numbers from the earlier example. This is another example of the overhead/security trade-off. If we only consider retrieving the first secret, $r = 13$ provides very strong security as shown in Table I by the huge number of attempts an adversary needs. However, if we also are concerned about the security after one secret is reconstructed, we need to further reduce the overlap, which again increases the number of shares. Recall that, with the SIMPLE scheme, the adversary only needs $n + m - 2 = 113$ attempts with these parameter values to reconstruct all secrets given the first secret. We can conclude that, after one secret is retrieved by an adversary, SIMPLE provides almost no security for the remaining secrets, while CYCLIC can still provide very strong security with a properly chosen overlap, at the expense of an additional increase in the number of shares.

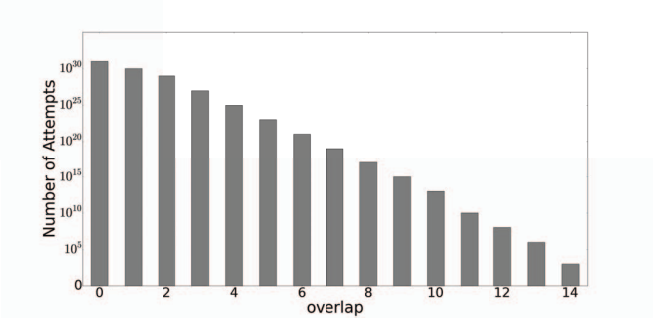

Fig. 7. Number of attempts needed to reconstruct all secrets given one secret in CYCLIC scheme ($m = 50$, $n = 15$, log scale on y-axis)

D. Discussion

As we showed in Subsection IV-C1, reconstructing the first secret is hard for any of the three schemes. The computation time depends on many factors including not only the parameters of the schemes but also the computational power of the adversary and the size of the secret. In some scenarios, it might be that the computational difficulty yielded by SIMPLE for reconstruction of the first secret with a small probability is high enough. However, in other scenarios, the system designer might not be satisfied that the SIMPLE's computational difficulty is high enough or they might not be comfortable with the probabilistic nature of the computational effort required. In these situations, CYCLIC allows the designer to, with a modest increase of overhead, significantly increase the overall security in the event that one secret is revealed.

V. Notched Cyclic Polynomial-Based DSS

In this section, we investigate a deception scheme that works with Shamir's secret sharing scheme [18], which is a (k, n) threshold-based scheme, where $k < n$ and any k of the n shares allow the secret to be reconstructed, while any $k - 1$ shares do not reveal any information about the secret. (k, n) secret sharing can offer better fault tolerance than (n, n) schemes such as XOR, because the secret can still be reconstructed even if some shares are not available. We present and analyze a scheme, which we call Notched Cyclic Polynomial-Based Deceptive Secret Sharing (or NCP for short) that performs deceptive secret sharing using Shamir's scheme.

A. Deception and Shamir's Scheme

In Shamir's scheme, each secret corresponds to a degree $k-1$ polynomial $q(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{k-1} \cdot x^{k-1}$, where a_0 is the value of the secret. The n shares that make up the secret are n points $(i, D_i), i = 1, 2, ..., n$, on the polynomial, i.e.

$$D_1 = q(1), D_2 = q(2), \ldots, D_n = q(n)$$

Share i is represented by the pair (i, D_i) and we call i the share's index. The secret can be expressed as D_0. Any k shares with k different indexes from $1, 2, ..., n$ determine the secret's polynomial and can therefore be used to recover the secret's value, and any $k-1$ shares reveal no information since there are an infinite number of degree $k-1$ polynomials that go through the $k-1$ points corresponding to the share values.

Analogous to NAIVE for XOR secret sharing, with Shamir's scheme, we could simply choose independent polynomials for each secret. We call this scheme P_NAIVE and it requires mn shares, just as NAIVE did. An analogous scheme to SIMPLE, with Shamir's scheme, is to choose $k-1$ polynomial points to be the same for all secrets. Each of the m secrets is then defined by a distinct k^{th} point, which is the secret value. To complete the (k, n) scheme, we would then generate $n-k+1$ additional points on the corresponding polynomial for each secret. We refer to this scheme as P_SIMPLE.

We assume throughout this section that an adversary who compromises a share knows its index. This comports with our assumption of a powerful adversary and also reduces the amount of information that we assume a valid user is able to determine but is not available to the adversary. This would also simplify implementation of the approach considerably since a share's index can simply be stored with the share as is the case in many implementations of Shamir's scheme. Unfortunately, under this assumption, the P_SIMPLE scheme is highly insecure. This is because the $k-1$ shares that are common to all secrets can be immediately identified since they are the only ones that have their particular index. Suppose those indexes are $1, 2, \ldots, k-1$. Then, the adversary can simply put those shares together with each share that has index k, in turn, and recover all m secrets with only m reconstructions. Thus, with Shamir's scheme and with this assumption, P_SIMPLE is not a viable approach.[3] For this reason, we primarily compare our proposed NCP scheme against P_NAIVE in the remainder of this section.

B. Description of NCP scheme

The NCP scheme is similar to the CYCLIC scheme in the way that it reuses shares across multiple secrets and attempts to balance the number of secrets of which each share is a part. Because of their similarity, we will mainly describe how NCP is different from CYCLIC in this section. We still denote the number of secrets by m and the overlap between two neighboring secrets by r. A significant difference of NCP, as compared to CYCLIC, is that the overlap r between two consecutive secrets has to be less than k. If this were not true, then two consecutive secrets would have the same value since they would share at least k points on their degree $k-1$ polynomials, meaning the polynomials (and hence the secret values) would actually be the same.

Next, we describe two aspects in which the NCP scheme is different from CYCLIC, namely the process of generating shares and the structure of the circle.

Since two secrets may have common shares, two polynomials may share some points. To simplify the scheme, we give each share the same index in all secrets it belongs to, so there is a bijection between the shares and the points. The generating process is similar to the CYCLIC scheme, but here each share needs to be given an index. Again drawing the shares and secrets around a circle, the indexes are generated in a round-robin fashion around the circle from 1 to n, wrapping back around to 1, and repeating until all shares are assigned an index. Figure 8 shows the indexes generated for $m = 3$, $n = 6$, $k = 4$, and two different values of $r < k$.

Once share indexes are assigned, shares can be generated. For the first secret, the first $k-1$ shares are generated randomly. With a_0 as the secret, once the first $k-1$ shares are determined, the polynomial $q(x)$ is determined and the remaining $n-k+1$ shares are also determined and are generated as $q(k), q(k+1), \ldots, q(n)$. The second secret shares $r < k$ shares with the first secret, which have indexes $n-r+1, n-r+2, \ldots, n$. These same share values are re-used by the second secret and an additional $k-r-1$ shares with indexes starting from 1 are generated at random. At this point, $k-1$ shares of the second secret are known and, along with its secret value, this determines its polynomial. The remaining $n-k+1$ shares can then be generated according to the polynomial's formula. The process is similar for each new secret around the circle. The first $r < k$ shares are in common with the previous secret, an additional $r-k-1$ shares are generated randomly, and the final $n-k+1$ shares are then generated from the chosen polynomial.

We would like to keep a share's index the same in every secret it belongs to, and so we choose not to have the circle wrap around as in the CYCLIC scheme, because choosing shares for the wrapped-around portions can be quite complex. In Figure 8, we see that if the example on the right-hand side wrapped around, we could simply select the shares with indexes 1 and 2 from Secret 1. However, if the example on the left-hand side wrapped around, the final share should have index 4 and so it would need to use the 4th share from Secret 1 instead of wrapping around in the normal way. While it would be possible to define the scheme to wrap around and this might reduce the overall number of shares in some cases, we have chosen not to include the wrap-around feature in order to simply specification and analysis of the scheme.

There are several things to emphasize about the scheme. First, each share has only one index, no matter how many secrets it belongs to. This helps simplify implementation of the scheme, since we don't need to store (or be able to

[3] If we instead assume that indexes are available to valid users but not to the adversary, the security of P_SIMPLE would be similar to that of SIMPLE.

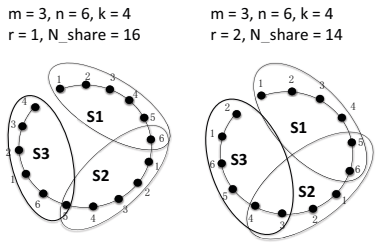

Fig. 8. Examples of NCP scheme

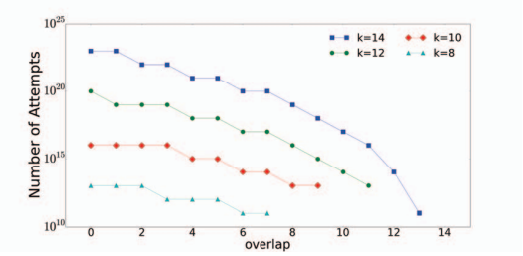

Fig. 9. Number of attempts needed to reconstruct first secret with probability 0.1 for NCP scheme ($m = 50$, $n = 15$, log scale on y-axis)

dynamically generate) multiple indexes for the same share. Second, two consecutive secrets on the circle have r shares in common, where $r < k$ as discussed earlier.

C. Overhead of NCP scheme

We follow a similar analysis to that done for CYCLIC in the previous section. To derive the total number of shares, N_{share}, we see that the first secret generates n shares and each additional secret generates $n - r$ new shares. Thus, $N_{share} = n + (m-1) \times (n-r) = m(n-r) + r = mn - r(m-1)$. Compared with N_{share} for CYCLIC, here it is obvious that, with fixed m and n, a higher overlap r reduces N_{share}. Therefore, to minimize the overhead of NCP, we should maximize r, subject to constraints on security, which we analyze in the next two subsections.

It is clear from the above expression that, for a given m and n, N_{share} decreases linearly with r. Thus, the shape of the N_{share} curves is very similar to those shown in Figure 4, except that r cannot be increased beyond $k-1$. Note that, just as with NAIVE in Figure 4, the number of shares used by P_NAIVE would be the left-most point on the NCP curves (corresponding with $r = 0$).

D. Security of NCP scheme

In this section, we discuss the security of NCP. Reconstructing secrets is very different from CYCLIC because, as mentioned earlier, we assume that an adversary knows the index for any share that it is able to retrieve. Knowledge of the index provides more information about where the share fits and this makes the reconstruction process simpler. Despite this, we will show that NCP can still provide high security with a reasonable overhead. Due to page limitations, we omit some details of the security analyses.

1) Reconstructing a first secret: The number of combinations an adversary must try to reconstruct a secret, N_{comb_all} for NCP with (k, n) secret sharing can be shown to be lower bounded by w^k, where $w = \lfloor \frac{m(n-r)+r}{n} \rfloor$. With this new value of N_{comb_all}, we can use Equation 1 to get the probability of reconstructing at least one secret in t attempts with the same definition of N^p_{comb}. We show N^p_{comb} for $p = 0.1$ and varying k and r, with the y-axis on a log scale in Figure 9. We fix $m = 50$ and $n = 15$ in the example. We choose k as 14, 12, 10, and 8 and vary r from 0 to $k-1$ in each case. We note that $r = 0$ gives the number of combinations for P_NAIVE. We can see that a higher k increases the security level, since w^k will be larger and there are more overall combinations for the adversary to consider.

Note that the calculations for decoding a secret using Shamir's scheme involve polynomial interpolation and, therefore, the process is more complex than for (n, n) secret sharing, which uses only XOR operations. Thus, one secret reconstruction will take considerably longer for NCP than for CYCLIC. Despite this added complexity for NCP, for illustration, we will assume the same security goal from the previous section, where 10^{16} attempts was considered a good security threshold. To reach 10^{16} attempts in Figure 9, k must be at least 12. We can see that there are many choices of r with both $k = 12$ and $k = 14$ that yield at least 10^{16} attempts.

It is also interesting to consider the overhead of NCP configured to achieve the required security level and compare it to P_NAIVE. With $n = 15$ and $m = 50$ as used in Figure 9, the number of shares required by P_NAIVE is 750. If we choose $k = 14$ and $r = 10$, then N_{share} for NCP is 260 and the desired security level is achieved with only about 1/3 the number of shares required by P_NAIVE.

2) Reconstructing all secrets with one secret reconstructed: We still assume that, once a secret is reconstructed, the adversary will next try a neighboring secret in the circle, because the first secret reconstruction reveals some information about the neighboring secrets. As in the analysis of CYCLIC, there are again three cases depending on the overlap. We omit the analysis details due to page limitations and simply state the results for these cases. When $r < k/2$, the number of combinations is:

$$n \times w^{k-r} + \sum_{t=1}^{\frac{wn}{k-r}} (\lfloor w - t \times \frac{k-r}{n} \rfloor)^{k-r}$$

When $k/2 \leq r < n/2$, the number of combinations is:

$$n \times w^{k-r} + \sum_{t=1}^{\frac{2wn}{k}} (\lfloor w - t \times \frac{k}{2n} \rfloor)^{\lfloor \frac{k}{2} \rfloor}$$

Finally, when $k/2 \leq r < n/2$, the number of combinations is:

$$n \times w^{k-r} + \sum_{t=1}^{\frac{wn}{k(n-r)}} (\lfloor w - t \times \frac{k(n-r)}{n*n} \rfloor)^{\lfloor \frac{k(n-r)}{n} \rfloor}$$

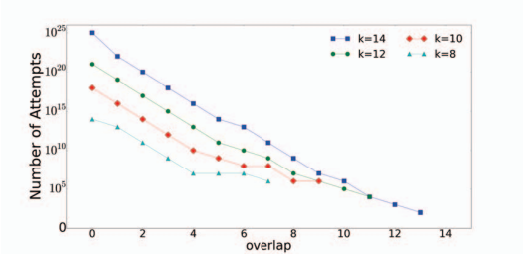

Fig. 10. Number of attempts needed to reconstruct remaining secrets given one secret in NCP scheme ($m = 50$, $n = 15$, log scale on y-axis)

We use the above expressions to generate Figure 10, which shows the number of combinations to guarantee that all remaining secrets can be reconstructed given the first secret. With the example from the previous subsection with $k = 14$ and $r = 10$, which was sufficient to ensure the desired security level for the first secret, an adversary can reconstruct all remaining secrets with only about 10^6 attempts. However, if we reduce r to 5, the number of attempts increases to about 10^{16}, which was our target. This decrease in r causes the number of shares to increase to 505, which is still about 2/3 of the number needed in P_NAIVE but is significantly higher than when only considering the security of the first secret.

E. Discussion

It is interesting to consider the relationship of the parameters m and k in NCP relative to the parameter k in Shamir's secret sharing scheme (without deception). In Shamir's scheme, k is thought of as a fault threshold, i.e. it is the minimum number of servers that must be compromised by an adversary to reveal a secret. Such threshold schemes have been criticized for security if they are deployed on a set of identical servers, where a vulnerability on one server is likely to be present on all other servers as well. A major benefit of deceptive secret sharing is that k is no longer a strict fault threshold. Even if an adversary is lucky enough to compromise k servers that contain a set of shares that allow a correct-looking secret to be assembled, it is unlikely to actually be a correct secret. Even if all servers, and therefore all shares, are compromised, we have shown that the adversary would still have to do a practically infeasible amount of computation to reconstruct all secrets and, therefore, be guaranteed to know at least one correct secret. In other words, using deceptive secret sharing with a large enough value of m, even with a small k, can provide much stronger security than simply increasing k without deception.

VI. Prototype Evaluation

Here, we evaluate our techniques using a CloudLab [4] prototype. Since security was extensively evaluated in previous sections, we focus on performance and data availability in the experimental evaluation.

A. Overview of Prototype Implementation

The implementation follows the architecture of Figure 2. The storage servers shown in Figure 2 are implemented on servers in CloudLab. The metadata server is deployed on a separate CloudLab server. Each server has eight ARMv8 cores running at 2.4GHz. The client is deployed at the CloudLab site in South Carolina while the servers reside in Utah. The client contacts the metadata server on writes, to determine on which storage servers to write shares, and on reads, to determine from which storage servers to read shares and which shares make up the real secret. All shares (real and fake) are read so as not to reveal to an adversary eavesdropping on the storage servers' network which are the real shares. We were able to maintain up to 150 servers reliably in CloudLab for the durations of our experiments. Since the number of shares in our schemes often exceeds 150, we store multiple shares on each server and the metadata server chooses a server at random for each individual share (random share assignment).

We have implemented both CYCLIC and NCP in the prototype. Since availability is an important issue in cloud storage and CYCLIC cannot tolerate a single crashed server, we also implemented CYCLIC with triple-modular replication, where each primary storage server is matched with two other storage servers that maintain exact replicas.

B. Evaluation

Metrics: We evaluated the performance of the schemes by measuring the latency of reads and writes for different object sizes and different secret sharing parameters. The latency of read and write operations includes the time to communicate with the metadata server and all relevant storage servers, to generate shares for the write, and to reconstruct the secret for a read. We also evaluated the data availability of the approaches. Data availability was measured as follows: we wrote 1000 objects to the prototype using the random share assignment scheme, emulated crashes of some storage servers, and then attempted to read back the 1,000 objects. The fraction of reads that were successful is our data availability metric.

Latency: Figure 11 shows the latency of the three schemes for different object sizes and secret sharing parameters. As can be seen, most of the latencies are between 1.5 and 3.5 seconds to read or write a 256KB file to/from the cloud, which is an acceptable value. The exception is the higher write latency for CYCLIC with replication where, with synchronous replication, each individual share write must occur at 3 different servers. This latency could be reduced to be similar to the read latency using lazy replication techniques, which would on the other hand loosen consistency and reduce availability. It can also be seen that latency increases as overlap decreases ($n - r$ increases), because smaller overlap increases the total number of shares that must be read or written. The bottom left of the figure shows the impact of object size. Performance is reasonable up to 1 MB files but increases significantly for 4 MB files. The bottom right compares all 3 schemes and shows that their latencies are similar except for the replicated write latency as already discussed. Latency is most sensitive to the overlap in our techniques while n and k have much less impact since the number of shares is primarily determined by $n - r$.

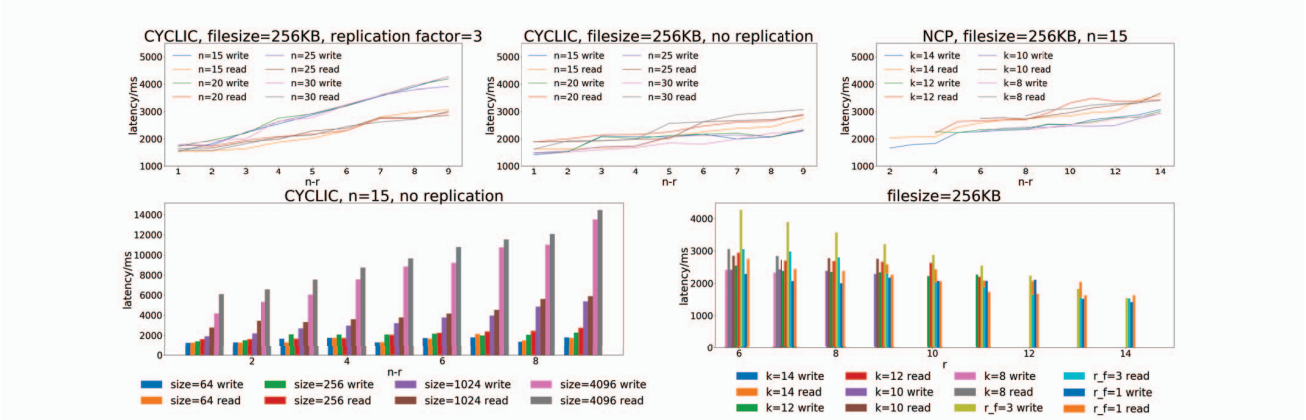

Fig. 11. Latency. The top shows the three schemes with 256KB objects and $m = 50$. The bottom left focuses on impact of object size using CYCLIC without replication for illustration. The bottom right compares the three schemes on one graph.

file size (KB)	no encoding		deceptive secret sharing	
	write	read	write	read
64	1.108s	1.078s	1.256s	1.274s
256	1.205s	1.312s	1.415s	1.624s
1024	1.693s	1.762s	1.905s	2.772s
4096	2.983s	3.207s	4.191s	6.099s

TABLE II
COMPARISON BETWEEN DECEPTIVE SECRET SHARING AND A SYSTEM WITH NO SECRET SHARING AND NO DECEPTION.

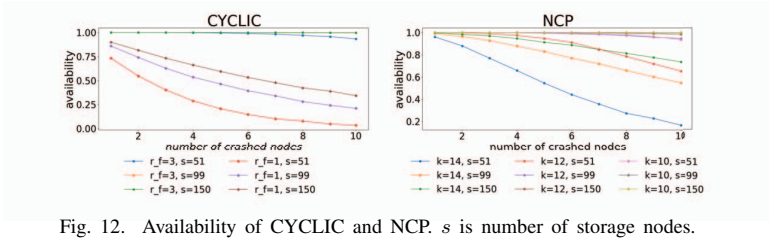

Fig. 12. Availability of CYCLIC and NCP. s is number of storage nodes. ($m = 50$, $n = 15$, $r = 9$)

We also wanted to understand how the performance of secret sharing and deception compares to a system with no protection at all, i.e. one where there is no secret sharing and no deception deployed. The latter situation is equivalent to reading or writing a single share, instead of the N shares required for deceptive secret sharing, where N depends on the parameters of the DSS scheme. Table II compares the cyclic DSS scheme without replication and with $m = 50$, $n = 15$, and $r = 14$, in which each file is encoded with 64 shares, compared with a single share for no secret sharing and no deception. Despite reading or writing 64 times the number of shares, the latency of DSS is always less than twice that of reading a single share. For the smaller file sizes, the DSS latency is actually only 10–20% higher than reading a single share. This is because, with these file sizes, performance is not bandwidth-limited and multiple share reads/writes can overlap substantially. With much larger file sizes, the latency degradation would definitely increase. However, for applications where file sizes are in the range we evaluated such as email, photo sharing, etc., latencies should definitely be in a range that is tolerable for users.

Availability: Figure 12 depicts the data availability of the three schemes. Note that the availability of CYCLIC without replication drops very quickly as the number of crashed servers increases. Each server stores shares of some real secrets and CYCLIC needs all shares to reconstruct the secret, therefore even one crashed server causes some data objects to be unavailable. Notice also that availability decreases with the number of servers since, with fewer servers, more shares need to be stored on each server. CYCLIC with three replicas improves fault tolerance significantly, having availability near one even with up to 10 crashed servers, except when the number of servers is quite small (50). Triplication comes with very high cost, however, as the storage overhead is tripled and write latency is also increased substantially unless lazy replication is employed. Note, however, that NCP with appropriate values of k and a large enough number of servers can also achieve availabilities near one for a relatively large number of crashed servers. Since NCP has similar overheads to CYCLIC without replication, in those scenarios it provides an ideal solution when considering security, performance, and availability. As an example, with $n = 15, k = 12, r = 9, m = 50$, and 150 servers of which 8 are crashed, NCP requires 309 shares and has an availability of 0.9995, while CYCLIC with triplication requires 903 shares and has an availability of 0.9999.

VII. RELATED WORK

Fake resources are often used to detect and/or track attacks. This was first done with honeypots [16], [19], [24], which capture attack traffic. An additional deception is used in honey patches [1], in which an attempt to exploit a patched vulnerability on a system causes the attack to be redirected to a honeypot where the attack is monitored and the attacker is led to believe that the attack succeeded. Deception has also been used inside legitimate systems to detect attacks using fake credentials [3], [10] or fake documents [3], [22]. Access

to fake resources causes alerts to be generated and, in [3], also generates information about where an exfiltrated document was opened. In [12], the authors discuss information deception generally. These works deploy fake resources openly, because they want adversaries to access them. The issue of confidentiality of sensitive information that might exist on the same systems is considered to be orthogonal and not addressed. Our work combines the protection of the real information with provision of fake information to provide two layers of security.

One aspect of our work involves hiding true information among a sea of true and fake information. The earliest use of this idea is Rivest's chaffing and winnowing concept, which he applied in the context of confidential exchange of messages across a network [17]. This idea has also been proposed to protect stored information [5], [6], [23]. The chaffing and winnowing concept was extended to distributed storage in [23]. This work provides a novel encoding to produce wheat and chaff pieces for storage and uses an inverse decoding process to eliminate the chaff when reading the data. In [5], the authors secret share sensitive information and hide the shares within a very large file, which increases the difficulty for an attacker to exfiltrate the information for off-line analysis. In [6], large shares that are hard to exfiltrate are also used together with an interactive secret reconstruction scheme, which is resilient to attackers that can compromise a limited number of rounds of reconstruction. None of these approaches addresses the additional deception aspect that we consider herein.

Secret sharing has been used in various ways in distributed storage systems [2], [8], [14], [15], [20], [21]. One approach is to encrypt data and secret share the key among multiple servers [8]. In [14], secret sharing and replication are combined to reduce write cost compared to (k, n) secret sharing. In [21], secret sharing and XOR replication are combined to provide equivalent guarantees to (k, n) secret sharing but with better performance. Another approach is to secret share data across multiple cloud providers [2], which can protect against insider attacks within a single provider. In [20], the authors propose combining secret sharing with RAID to provide confidentiality and reliability of archival data. In [15], Shamir's secret sharing scheme was extended with Byzantine fault tolerance. All of these works are only concerned with confidentiality and do not address the issue of deception.

The only work of which we are aware that combines confidentiality and deception as is done in our approach is honey encryption [9], [11]. In honey encryption, decryption with the proper key yields the correct plaintext but decryption with other keys yields correct-looking but fake plaintext. The approach relies on distribution-transforming encoders (DTEs), which transform messages into seeds that are then encrypted. To date, DTEs have been given for only a few message types, e.g. RSA keys and credit card numbers in [11] and genomic data in [9]. Since our approach is built upon basic secret sharing schemes, it can be applied to any type of data.

VIII. Conclusion

We presented new deceptive secret sharing techniques that work with both XOR secret sharing and Shamir's polynomial-based threshold secret sharing. We evaluated both the overhead and security of the proposed techniques and showed that they permit tunable security, i.e. a tradeoff between security and overhead, by varying one parameter of the techniques. A Cloudlab prototype was used to show that our techniques have acceptable latency and very high availability.

References

[1] Araujo F, Hamlen K W, Biedermann S, et al, "From patches to honey-patches: Lightweight attacker misdirection, deception, and disinformation," *Proc. ACM SIGSAC Conference on Computer and Communications Security,* 2014: 942-953.

[2] Bessani A, Correia M, Quaresma B, et al, "DepSky: Dependable and secure storage in a cloud-of-clouds," *ACM Transactions on Storage (TOS),* 2013, 9(4): 12.

[3] Bowen B M, Hershkop S, Keromytis A D, et al, "Baiting inside attackers using decoy documents," *Int'l Conf. on Security and Privacy in Communication Systems. Springer Berlin Heidelberg,* 2009: 51-70.

[4] CloudLab Team, "The CloudLab Manual," http://docs.cloudlab.us/.

[5] Dagon D, Lee W, Lipton R, "Protecting secret data from insider attacks," *Int'l Conf. on Financial Cryptography and Data Security,* 2005: 16-30.

[6] Dziembowski S, Pietrzak K, "Intrusion-resilient secret sharing," *Proc. Symposium on Foundations of Computer Science,* 2007: 227-237.

[7] Greenberg, A., "Hackers hit macron with huge email leak ahead of French election," *Wired* Security blog, https://www.wired.com/2017/05/macron-email-hack-french-election. (URL checked on 4/12/2018)

[8] Herlihy, M., and Tygar, J., "How to make replicated data secure," *Proc. of Advances in Cryptology,* pp. 379–391, 1987.

[9] Huang, Z., Ayday, E., Fellay, J., et al., "GenoGuard: Protecting genomic data against brute-force attacks," *Security and Privacy (SP), 2015 IEEE Symposium on. IEEE,* 2015: 447-462.

[10] Juels A., Rivest R., " Honeywords: Making password-cracking detectable," *Proc. ACM SIGSAC Conference on Computer & Comm. Security,* 2013: 145-160.

[11] Juels A, Ristenpart T, "Honey encryption: Security beyond the brute-force bound," *Int'l Conf. Theory & Appl. Crypto. Tech.,* 2014: 293-310.

[12] Kott, A., Swami, A., and West, B., "The fog of war in cyberspace," *IEEE Computer,* pp. 84–87, Nov. 2016.

[13] Krawczyk, H., "Secret sharing made short," *Proc. of the Annual International Cryptology Conference,* pp. 136–146, 1993.

[14] Lakshmanan, S., Ahamad, M., and Venkateswaran, H., "Responsive security for stored data," *IEEE Transactions on Parallel and Distributed Systems,* Vol. 14, pp. 818-828, 2003.

[15] Padilha, R., Pedone, F., "Belisarius: BFT storage with confidentiality," *Proc. Int'l Symp. on Network Computing and Appl.,* pp. 9–16, 2011.

[16] Provos, N., "Honeyd: A virtual honeypot daemon," *Proc. of 10th DFN-CERT Workshop,* pp. 4–9, 2003.

[17] Rivest, R.L., " Chaffing and winnowing: Confidentiality without encryption," *CryptoBytes (RSA laboratories),* 1998, 4(1): 12-17.

[18] Shamir, A., "How to share a secret," *Communications of the ACM,* Vol. 22, pp. 612–613, Nov. 1979.

[19] Spitzner L, "Honeypots: Catching the insider threat," *Proc. Computer Security Applications Conference,* 2003: 170-179.

[20] Storer, M., Greenan, K., Miller E., et al., "POTSHARDS: Secure long-term storage without encryption," *2007 USENIX Annual Technical Conference. USENIX Association,* 2008.

[21] Subbiah, A. and Blough, D.M., "An approach for fault tolerant and secure data storage in collaborative work environments," *Proc. of the ACM Workshop on Storage Security and Survivability,* pp. 84–93, 2005.

[22] Yuill J, Zappe M, Denning D, et al, "Honeyfiles: Deceptive files for intrusion detection," *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC. IEEE,* 2004: 116-122.

[23] Zage, D., Obert, J., "Utilizing linear subspaces to improve cloud security," *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on. IEEE,* 2012: 1-6.

[24] Zhang, F., Zhou, S., Qin, Z., and Liu, J., "Honeypot: A supplemented active defense system for network security," *Proc. Int'l. Conf. on Parallel & Dist. Computing, Applications and Technologies,* pp. 231–235, 2003.