A Secure, Structured, Distributed Caching System for Providing Availability of Mission-Critical Reference Data

Robert C. Durst, Sushil Jajodia, *Senior Member, IEEE,* Alessandro Mei, and Susan F. Symington

***Abstract*—Mission-critical information is typically stored in the clear on some trusted portion of a network and only encrypted when sent elsewhere. If the network is penetrated, the information becomes vulnerable to disclosure, modification, and deletion, thus jeopardizing the mission. In response to such an attack or the imminent threat of attack, the information may be disconnected from the network, but the resulting lack of availability may also jeopardize the mission. We define requirements for a data caching system that is designed to maintain availability of mission-critical reference information, despite network penetration by an adversary, without sacrificing the information's security. We describe a basic network model and three alternative caching architectures to address these requirements: a secure, centralized (SCCA); a secure, unstructured, distributed (SUDCA); and a secure, structured, distributed (SSDCA) caching architecture. We define availability and confidentiality models and apply them to characterize these three architectures and compare their relative performance. We show that the SSDCA outperforms the alternatives in providing data availability and data confidentiality, assuming the compromise of data caches and the presence of eavesdropping. Lastly, we recommend related areas for further exploration.**

***Index Terms*—Distributed Caching, Erasure Coding, Fragmentation, Confidentiality, Integrity, Availability, Resiliency**

I. Introduction

RUNNING a mission requires access to information that is stored on network systems. Some of this information is relatively static (e.g., map data) as opposed to being subject to frequent updates. We call such static data *reference information*. In today's networks, mission-critical information is typically protected from disclosure, modification, and deletion using perimeter protection, in which some inner sanctum of the network is assumed to be trusted because measures are in place between it and the Internet to protect it from penetration from outside.

Often, the information in this trusted portion of the network is stored in the clear and is only encrypted when sent outside of the trusted portion of the network. If the trusted portion of the network is penetrated by an intruder or if it is subjected to an insider threat, the information on it is vulnerable to disclosure, modification, and deletion, thereby jeopardizing the mission's success. As a result, if a network penetration is detected or if an insider threat is discovered, the typical response is to disconnect the trusted portion of the network from the internet or to shut down the network altogether. Such self-partitioning jeopardizes availability of mission-critical data, however, and may mean having to abort the mission or may cause the mission to fail.

In some cases, high-value information may be encrypted. However, such encryption is typically at the file-system level (whole-disk encryption) rather than the object (individual file) level, precluding the information from being searched or made available in encrypted form at the object level. In other cases, systems storing extremely high value data are not permitted to be connected to the Internet. Such policies achieve security at the cost of limiting availability. Ideally, mission-critical reference information should be both available and secure throughout the lifetime of a mission, even if the network on which the information resides has been penetrated.

Objectives and Approach

We define requirements for a data caching system that is designed to maintain availability of mission-critical reference information, despite network penetration by an adversary, without sacrificing the information's security. We design a basic network model and three alternative architectures to address these requirements: a secure, centralized caching architecture (SCCA), a secure, unstructured, distributed caching architecture (SUDCA), and a secure, structured distributed caching architecture (SSDCA). We define availability and confidentiality models to characterize these architectures and compare their relative performance. Lastly, we recommend additional related areas for further exploration.

Threat Model and Trust Requirements

We assume an internal adversary who can gain standard-user privilege access to client nodes and who can exfiltrate, modify, and delete stored information (except for stored keys)

Manuscript received June 15, 2009. This work was supported in part by The MITRE Corporation.

R. C. Durst, S. Jajodia, and S. F. Symington are with The MITRE Corporation, McLean, VA 22102 USA (corresponding author to provide phone: 303-555-5555; fax: 303-555-5555; e-mail: {durst, susan}@mitre.org susan@mitre.org).

S. Jajodia is also with the Center for Secure Information Sciences, George mason University, mail Stop 4A4, Fairfax, VA 22030-444 USA (email: jajodia@gmu.edu).

A. Mei is with the Dipartimento di informatica of the Università di Roma "La Sapienza," Via Salaria 113, 00198 Roma, Italy (e-mail: mei@dsi.uniroma1.it).

on both clients and data caches. The adversary is assumed to understand the network topology and know on which nodes information is stored; he is also assumed to have the ability to eavesdrop on traffic throughout the network.

We trust all key servers to manage and distribute cryptographic keys correctly and without compromise. We trust that all reference data is correct and trustworthy upon initial entry into the network. We do not trust client or caching systems; these are vulnerable to compromise. We assume that both client and caching systems operate on high assurance platforms that prevent an adversary from gaining access to key material stored on them. We assume that all data in persistent storage on clients is stored in encrypted form. If an adversary gains control of a valid user account on a client, he will be able to view in plaintext all reference information that the account holder is authorized to access. However, his privileges will be limited to the account holder's privileges; he will not be able to access or corrupt any other reference information.

Requirements

The overarching requirement of our caching system is to maintain availability of the reference information that is stored, despite network penetration and without sacrificing the information's security. In the context of our threat model, this means that our system must prevent disclosure of both data at rest (DAR) and data in motion (DIM), and prevent disclosure and ensure the integrity and availability of information despite some eavesdropping and despite some number of caches being compromised or made unavailable. Furthermore, our system must prevent disclosure of information even in the case in which an adversary absconds with some number of data caches and subjects their contents to high-powered, nation-state-capable, brute force attempts to crack the encryption therein. To promote availability, our system must be able to locate and retrieve information based on its content in addition to its filename. Our system must preserve availability despite periods of network partitioning and despite a lack of continuous connectivity to the data owner, which was the original source of the reference information and which controls the information's access control policy. If the access control policy changes due to a user privilege revocation or some other reason, then to take advantage of the availability of the information that is already stored on the caches our system must honor this revised access control policy to the extent possible without requiring that any cached reference information be replaced and without requiring continuous connectivity to the Data Owner.

II. The Basic Network Model

We assume a network model consisting of four types of nodes as shown in Fig. 1. *Data Owners* insert reference information into the network and set the information access control policy. *Key Servers* store and distribute all necessary keys via secure channels as determined by the access control policy. *Data Caches* store reference information, and *Clients* consume it. Our model supports encryption of both data at rest and data in motion, encryption on a per-file basis, data integrity, and content-based searching. It also provides dynamic access control using the over-encryption technique proposed in [2]. Data Owners Base Encryption Layer (BEL)-encrypt data to protect its confidentiality from the Data Caches that will store it, and Data Caches Surface Encryption Layer (SEL)-encrypt this data upon providing it to clients, using a key appropriate to the requesting user's privileges as defined by the most recent access control policy.

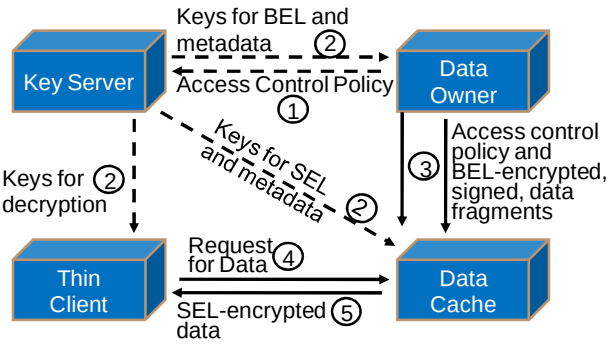

Fig. 1. The Basic Network Model

Data Owners

Before inserting a file into the network, a Data Owner signs the file, BEL-encrypts the file and signature, associates (possibly encrypted) metadata with the encrypted file, signs the combined encrypted file and metadata, and sends the result to one or more Data Caches for storage.

The metadata that is associated with the file indicates attributes such as file content, associated mission, or security level. Metadata enables the file to be subjected to content-based search as well as filename-based access. If metadata confidentiality must be maintained, the Data Owner encrypts the metadata using a different encryption key from that which was used to BEL-encrypt the file. The metadata key is shared with the Data Caches so that they can examine the metadata. BEL keys, however, are never shared with any Data Caches; Data Caches never have the capability to decrypt the reference information that they store.

Data Caches

Data Caches do not have user accounts. Data Caches receive access control policies and BEL-encrypted files from Data Owners. They verify the integrity of these files before storing or forwarding them. Upon receipt of a Client's request for information, a Data Cache will forward the request to adjacent Data Caches using diffusion routing. It will also search its store for all information having metadata that satisfies the request. If the request was received from an adjacent Client that has appropriate privileges, the responding cache will SEL-encrypt the data based on the access control policy and send this BEL- and SEL-encrypted data to the Client. If the request was received from an adjacent Data Cache, the responding cache will forward satisfying BEL-encrypted data to the requesting cache.

Within the network, content-based Data Caches are maintained dynamically using diffusion routing. The result is that information is stored in the network near where it has

been used so that when it is requested by subsequent users, it is available even if there is currently no connectivity from the originator to the user because it only has to travel from where it is stored to the user.

Clients

Users have accounts on Clients from which they request reference information according to characteristics that are captured by the information's metadata. Clients send these requests to their neighboring Data Caches.

Upon receiving the requested reference information, the Client verifies its integrity, SEL-decrypts it, and then BEL-decrypts it to recover the plaintext reference information and provide it to the user. The plaintext form of the reference information resides only in the Client's volatile memory; it is never stored in the Client's persistent memory.

Dynamic Access Control

When a Data Cache receives a request from a Client, it consults the access control policy that has been provided to it by the Data Owner to verify that the requesting user is authorized to receive that information, applies SEL-level encryption to the information based on that access control policy, and sends this BEL- and SEL-level encrypted information to the requesting user's Client. If the access control policy changes, the Data Owner must send the new policy to the Key Server and the Data Caches, but availability is not impacted because there is no need for the Data Owner to re-encrypt any reference information, nor is there any need for the reference information that is already stored in the caches to be replaced. Because the information in the caches will have SEL-level encryption applied at the time the information is sent to a Client, the information will be encrypted using a key appropriate to the access control policy that was most recently provided to the Data Cache.

This approach does not require the Data Caches to interact with the Data Owners at each data request, enabling the cached information to remain available despite the fact that the Data Owner may not be available. The application of BEL-encryption by the Data Owner ensures that the Data Caches do not have access to the plaintext of the information that they store. This BEL-encryption protects the confidentiality of the information from attacks on Data Caches and from eavesdropping. The application of SEL-encryption by the Data Caches enables the protection applied to the data to be adapted to changes in access control policy simply by varying the key with which the SEL-level encryption is performed.

If a Data Cache has been compromised such that it is malicious or uncooperative, it may not willingly enforce the new access control policy that is provided to it. Because the information that it stores has been BEL-level encrypted by the Data Owner, however, the confidentiality of this data will not be at risk at the compromised Data Cache.

III. Three Architectures Used for Comparison

Given the basic network model already described, we define three architectures for using this model in support of an information storage and retrieval service:

- Secure, centralized caching (SCCA)
- Secure, unstructured, distributed caching (SUDCA)
- Secure, structured, distributed caching (SSDCA).

The quantity and structure of the Data Caches and the organization of the information as stored on these Data Caches is what distinguishes the three architectures from each other.

Secure, Centralized Caching (SCCA)

In this first alternative architecture, which is representative of current practice, there is a single Data Cache for storing reference information, and perhaps a backup Data Cache that can be accessed in the event that the main Data Cache is not functioning or is otherwise inaccessible. Each file is stored in its entirety in each cache, with the topology shown in Fig. 2.

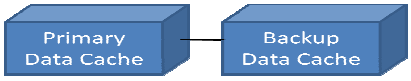

Fig. 2: Data Cache Topology in the SCCA

Secure, Unstructured, Distributed Caching (SUDCA)

In the SUDCA there are many Data Caches rather than just one or two. These caches are not structured insofar as there is no precise control over their topology or over which caches are permitted to forward information to other caches. Nor is there any control over the placement of information on the Data Caches. Caches join this architecture by following only loose rules, making it resilient to caches entering and leaving the network arbitrarily. This architecture is not depicted.

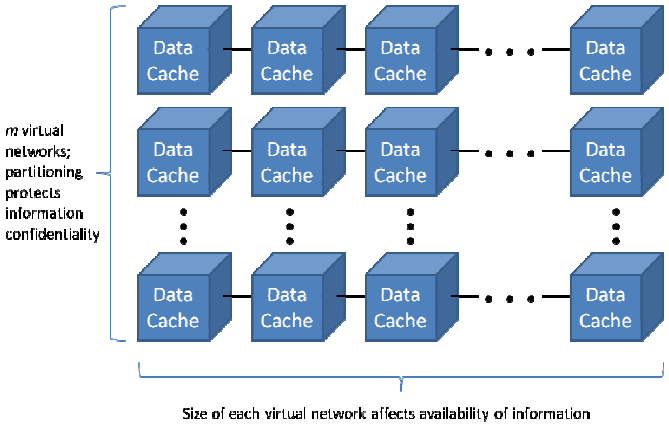

Fig. 3: Data Cache Topology in the SSDCA

Secure, Structured, Distributed Caching

In the SSDCA, which is depicted in Fig. 3, as in the SUDCA, there are many Data Caches. However, they have a significant amount of structure. Specifically, the set of connections among the Data Caches is tightly controlled to enforce partitioning of the network into *m* separate virtual networks (VNs), and reference information is stored on these VNs in a highly structured manner.

Storing Data in a Distributed Cache

Fig. 4 depicts the steps that are performed on a file before a

Data Owner sends it to a cache. In all three caching architectures, the Owner signs the file (depicted by "$") and encrypts the file and its signature. In the SCCA, the file is then ready to have metadata associated with it before being stored. In both distributed caching architectures, however, the Data Owner next fragments the file into *m* shares that are the result of a non-systematic erasure coding algorithm such that the file as a whole can be reconstructed by accessing *k* (where $k \leq m$) shares that are arbitrarily chosen, however, having *k-1* or fewer shares provides no benefit toward being able to reconstruct the file. Next, the Data Owner associates (possibly encrypted) metadata with the encrypted fragments, signs each fragment/metadata combination, duplicates the fragments, and disperses them among multiple Data Caches using diffusion based routing, which distributes the copies according to how they are requested.

In the case of the SUDCA, fragments may be placed in any Data Caches, and multiple fragments from the same file may (or may not) be on a single Data Cache. In the case of the SSDCA, the set of Data Caches in the network is partitioned into *m* subsets, as shown on the vertical axis in Fig. 3. Each of the Data Caches in subset *i* stores at most a single fragment, fragment *j*, of any given file. When duplicate fragments are inserted into the network by data owners, these fragments are all sent to Data Caches in the same subset *i* for $0 <= i <= m$. When fragments are forwarded between two Data Caches in the network, they must be forwarded between Data Caches that are in the same subset. In this sense, the network of Data Caches is partitioned into *m* VNs of Data caches, each of which stores and forwards only one of the *m* fragments of any given file. This dimension of structure serves to protect the confidentiality of the reference information by breaking it up into *m* fragments, while providing availability by making each fragment available on some VN. Within each VN, there are *x* Data Caches. This dimension of structure promotes availability, making multiple copies of a given fragment available within the network.

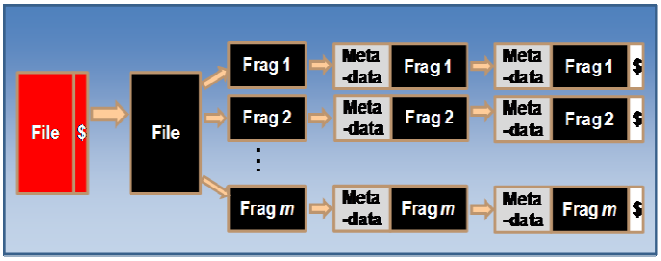

Fig. 4: Steps performed before storing information in one of the Secure, Distributed Architectures (both structured and unstructured)

Reassembly at the Client

When a Client requests information, it must receive at least *k* out of *m* fragments to reconstitute the requested file. Upon receiving *k* of *m* fragments, it validates and SEL-decrypts each fragment (not shown in Fig. 5), reassembles the original BEL-encrypted file, BEL-decrypts and validates this file, and provides it to the requesting user (see Fig. 5). Because not all *m* fragments of the file are required in order to reconstruct it, the Client will still be able to recover the information if up to *m-k* fragments are unavailable or corrupted.

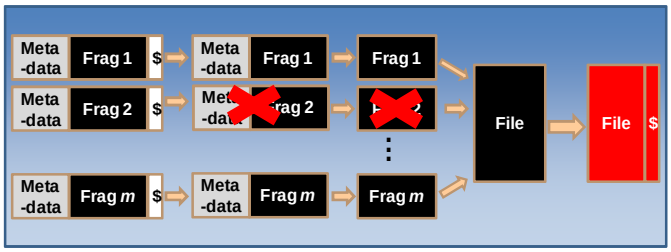

Fig. 5: Steps performed on the BEL-encrypted fragments by the Client before making the requested information available to the user.

Overall, the information assurance support provided by the SSDCA is summarized in Table I. This table describes the mechanisms that address each requirement.

Table I: SSDCA Threats and Associated Requirements

Threat	Requirement	Mechanism
Adversary penetrates up to *k-1* Data Caches to exfiltrate information	Info. remains confidential	Data BEL-encrypted and caches don't have keys. Fragmentation of data and dispersion of shares across *m* VNs prevents adversary from obtaining all *k* shares needed to reconstruct the info.
Adversary can eavesdrop in network	Info. remains confidential	BEL- and SEL-level encryption; fragments dispersed across *m* VNs.
Adversary makes up to *m-k* Data Caches unavailable	Info. remains available	Replication of fragments on multiple Data Caches in each VN; *k* fragments are sufficient to reconstruct the data.
Adversary penetrates up to *m-k* Data Caches to modify or delete information	Integrity can be assured	Info is digitally signed and validated upon receipt by each Data Cache and Client; invalid info. is deleted upon detection; original data can be reconstructed using k valid shares.
Adversary can access data after his privileges have been revoked	User access is limited by his privileges; dynamic access control	Data Cache verifies user privileges and SEL-level encrypts data with a key specific to current access control policy before sending data to Client.
Compromise of both a Client and k-1 Data Caches	Info. user is not authorized to access remains confidential	Client has BEL key and can bypass SEL, but fragmentation prevents Client from having all *k* shares needed to reconstruct info. that the user is not authorized to access.
Compromise of Client	Info. user is not authorized to access remains confidential	No info. is stored persistently on Client in plaintext form; adversary cannot access client keys; Adversary may access only the info. the user is authorized to access, and only until AC policy is changed.

IV. Comparison Analysis

Methodology

We measure availability of a file by determining the amount of effort required on the part of an attacker to deny availability of that file to an authorized user. We define *H* to be the amount of effort required to deny availability to the first Data Cache and *h* as the amount of effort required to deny availability to subsequent caches of that same type. We

assume that $H > h$ because once an adversary has figured out how to deny availability to a single cache, he can use the same method to deny availability to a subsequent cache of the same type, and this subsequent attempt will require less effort than the initial one. Analogously, when measuring confidentiality, we define H to be the amount of effort required to hack into the first Data Cache to exfiltrate information and h to be the amount of effort required to hack subsequent caches of that same type, where $H > h$.

Despite the fact that the files are BEL-encrypted before being fragmented and stored, we do not take the confidentiality that is provided by this encryption into consideration in our confidentiality model. We consider only the confidentiality that is provided by the fragmentation of the information via non-systematic erasure coding. We assume that if an adversary is able to obtain k out of m fragments, confidentiality has been breached. In reality, the confidentiality of the information is greater, due to BEL-encryption. Our model may be considered to be calculating a lower bound on confidentiality, which is the effort needed to reach the point at which an adversary could begin to try to decrypt the BEL-encrypted file.

In both our availability and confidentiality models we assume that m is the total number of fragments of a file and k is the number of fragments needed to reconstruct the file. A_1, A_2, and A_3 are the three architectures: SCCA, SUDCA, and SSDCA. In A_3, m is also the number of VNs. We assume that there is one copy of each fragment of a given file (i.e., there is no fragment replication) and the adversary knows in which caches each fragment is located and so can target them. This means that in architecture A_1 there is no backup Data Cache; in A_2, because fragments are distributed randomly, some caches may contain multiple fragments from the same file; and in A_3, because the fragments from a given file must be partitioned across the set of VNs, there is exactly one fragment in each VN.

We assume that each of the Data Caches in a given VN is of the same type, but that Data Caches in different VNs are of different types. In A_2, the number and types of Data Caches are assumed to be the same as in architecture A_3. This means that in architecture A_2, there are m different types of Data Cache and the fragments are distributed randomly across these Data Cache types. We calculate the number of Data Cache types that, in architecture A_2, do not have any fragments from a given file on them. This number of Data Cache types that do not have any fragments from a given file on them is also the number of fragments that are located on a type of Data Cache on which other fragments of the same file are stored. Those fragments are of interest to us because an adversary only has to expend h effort to exfiltrate those fragments. This number of Data Cache types is calculated using a variation of the classic randomized "Balls and Bins" load balancing algorithm as follows:

Assume that you have m bins (the number of bins = the number of different Data Cache types in A2). Assume a worst-case scenario in which $k=m$, meaning that an adversary has to obtain all m fragments of a given file in order to reconstruct it. If we randomly distribute the fragments (balls) across these bins, we are interested in knowing the expected value of the random variable x, where x = the number of empty bins.

Define $x_i = 1$ if bin i is empty and 0 otherwise.

$$x = \sum_{i=1}^{m} x_i$$

Ex = expected value of x = $E\sum_{i=1}^{m} x_i = \sum_{i=1}^{m} Ex_i$

$Ex_i = 0 \cdot \text{Prob}[x_i = 0] + \text{Prob}[x_i = 1]$
$Ex_i = \text{Prob}[x_i = 1]$ = the probability that bin i is empty.

For a given ball, the probability that that ball is not in bin i is $(1 - 1/m)$.
So, the probability that none of the m balls is in bin i is $(1 - 1/m)^m$.
$Ex_i = (1 - 1/m)^m$.
As $m \to \infty$, $(1 - 1/m)^m \to 1/e$.
$\left(1 - \frac{1}{m}\right)^m \geq \frac{1}{e}\left(1 - \frac{1}{m}\right)$.

Therefore, $Ex = \sum_{i=1}^{m} Ex_i = m(1/e)(1 - 1/m)$, which is approximately m/e for large values of m. Therefore, the fraction of the m bins that are empty approaches $1/e$.

Availability

For the availability model, we define:

$D(A_x)$ = expected effort to deny availability to a file in architecture A_x.

Because an adversary needs to deny availability to *m-k+1* caches to deny availability to the data, we calculate that:

$D(A_1) = H$

$D(A_2) \leq (1 - f)(m-k+1)H + f(m-k+1)h$, where $f = \frac{1}{e}(1 - \frac{1}{m}) \sim \frac{1}{e} \sim 0.37$

$D(A_3) = (m-k+1)H$

$D(A_1) = H$ because in order to deny availability to a file in A_1, an adversary need only deny availability to one Data Cache, which requires effort H.

$D(A_3) = (m-k+1)H$ because in order to deny availability to a file in architecture A_3, an adversary needs to deny availability to *m-k+1* fragments, each of which is on a separate VN. The effort required to deny availability to each of these caches is H, because each VN is comprised of caches of different types.

Understanding $D(A_2)$ requires making use of the "Balls and Bins" analysis above, which showed that f is the fraction of the bins that are empty. This means that f is the fraction of the Data Cache types that are empty. In order to deny availability to the reference data in architecture A_2, an adversary needs to deny availability to *m-k+1* fragments, all of which are located on one of the fraction $(1 - f)$ of data cache types that contain fragments. The effort to deny availability to the first fragment in a cache of each one of these cache types is H. Fraction f of the fragments are also stored on caches of one of this fraction $(1 - f)$ of data cache types. The effort required to deny

availability to these additional fraction *f* of fragments is only *h*, because the adversary has already figured out how to deny availability to fragments on the data cache types on which they are stored. $D(A_2)$ is the expected effort required to make a file unavailable in A_2; the actual effort required depends on the distribution of fragments across data cache types. In the best-case scenario, all fragments are on different cache types, so $D(A_2)$ approaches $D(A_3)$. In the worst case, all fragments are stored on a single cache, so $D(A_2)$ approaches $D(A_1)$.

Using these formulas for availability, we conducted experiments of 100,000 iterations while varying the values of *k* and *m*. The results are shown in Table II.

Table II: Comparison of Effort Required to Deny Availability to a File in each Caching Architecture

k:m	centralized	Unstructured	structured
1:16	H	10.30H + 5.70h	16H
2:16	H	9.30H + 5.70h	15H
4:16	H	7.33H + 4.67h	13H
8:16	H	4.15H + 4.75h	9H
16:16	H	H	H

Confidentiality

We use an analogous methodology for our confidentiality model.

$S(A_x)$ = *expected effort to exfiltrate a BEL-encrypted file in architecture* A_x.

$S(A_1) = H$

$S(A_2) \leq (1-f)kH + fkh$, *where* $f = \frac{1}{e}(1-\frac{1}{m}) \sim \frac{1}{e} \sim 0.37$

$S(A_3) = kH$

In order to exfiltrate a file in architecture A_2, an adversary needs to exfiltrate *k* fragments, all of which are located on one of the fraction $(1-f)$ of data cache types that contain fragments. The effort to exfiltrate from the first cache of each type is *H*. Fraction *f* of the fragments are also stored on caches of one of this fraction $(1-f)$ of data cache types. The effort required to exfiltrate these additional fraction *f* of fragments is only *h*, because the Data Cache types on which they are stored have already been broken into.

Using these formulas for confidentiality, we conducted experiments of 100,000 iterations while varying the values of *k* and *m*. The results are shown in Table III.

Table III: Comparison of Effort Required to Exfiltrate a File in each Caching Architecture

k:m	SCCA (A_1)	SUDCA (A_2)	SSDCA (A_3)
1:16	H	H	H
2:16	H	H+h	2H
4:16	H	1.77H + 2.23h	4H
8:16	H	3.57H + 4.44h	8H
16:16	H	10.3H + 5.70h	16H

V. Conclusion and Future Work

We have demonstrated that the SSDCA outperforms the alternatives in providing availability and confidentiality, assuming the compromise of data caches and the presence of eavesdropping. Novel aspects of the caching model include providing for continuous and consistent protection of both DAR and DIM on a per data object (e.g. a per file or message) basis; dynamic content-based cache population based on previous information requests; metadata tagging of data objects to facilitate content-based searching and sharing rather than just filename-based searching and sharing and also to provide mission-use marking of information; the ability to encrypt the metadata separately from the data itself so that caches can locate and serve data from their caches despite their inability to decrypt the data itself; no persistent storage of reference data in plaintext form at any data cache or client system; dynamic access control that does not require any stored data to be replaced when the access control policy changes and that does not require continuous connectivity to the access control policy owner; and cross-domain information sharing potential.

For future work, we recommend exploring mechanisms to enhance the SSDCA, such as using a hash of each cache's public key certificate to determine which file fragment that cache may store and thereby avoid relying on hardware to enforce the partitioning of fragments across VNs. We suggest exploring how to move fragments among caches (assuming some caches are more trusted than others) so that the amount of effort required to compromise a file never falls below a certain threshold; investigating the potential to respond to changing threat levels or to the appearance or disappearance of caches by modifying the number of fragment replicas and/or moving their location, or by choosing different values for *k* or *m*; exploring the potential applicability of the architecture to the storage of non-reference information by experimenting with the propagation of updates from writing nodes to the caches and adjusting caching in response to a changing ratio of readers to writers, i.e., increasing caching when read frequency is high and reducing caching when write frequency is high, to balance consistency with availability.

References

[1] A. Mei, L.Mancini, and S Jajodia, "Secure Dynamic Fragment and Replica Allocation in Large-Scale Distributed File Systems," *IEEE Transactions on parallel and Distributed Systems*, vol. 14, No. 9, September 2003, pp. 885-896.

[2] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of Access Control Evolution on Outsourced Data", Very Large Database Endowment, September 23-28, 2007, Vienna, Austria.

[3] A. Shamir, "How to Share a Secret," Comm. ACM, vol. 22, no. 11, 1979.

[4] Y. Deswarte, L. Blain and J.-C. Fabre, "Intrusion Tolerance in Distributed Computing Systems", in *Proc. IEEE Symp. on Security and Privacy*, Oakland California (USA), 1991, pp. 110-121.