

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

NETSKOPE, INC.,

Petitioner,

v.

FORTINET, INC.,

Patent Owner.

PTAB Case No. IPR2023-00458

Patent No. 9,280,678 B2

**DECLARATION OF WENKE LEE, PHD IN SUPPORT OF PETITION FOR
INTER PARTES REVIEW OF U.S. PATENT NO. 9,280,678 B2**

TABLE OF CONTENTS

I.	INTRODUCTION	1
A.	Qualifications	2
1.	Education	2
2.	Work Experience.....	2
3.	Publications.....	5
4.	Curriculum Vitae.....	5
B.	Materials Reviewed.....	5
C.	Level of Ordinary Skill in the Art	7
D.	Summary of Opinions	9
II.	OVERVIEW OF THE TECHNOLOGY.....	9
A.	Priority Date of the Claims.....	9
B.	Overview of Relevant Technology When the '678 Patent Was Filed.....	10
1.	Cloud Storage.....	10
2.	Searchable Encryption	11
3.	Namespace	13
4.	Gateway	15
5.	File Chunking.....	16
C.	The '678 Patent Patent Overview.....	16
1.	Alleged invention.....	16
D.	The Challenged Claims	21
E.	Claim Construction.....	21
III.	UNPATENTABILITY OF THE '678 PATENT CLAIMS	21
F.	Standards for Invalidity	21
1.	Obviousness	21
IV.	SPECIFIC GROUNDS FOR PETITION	23

A.	Ground 1: Claims 1-3, 8-14, and 16-27 were obvious over Cidon, Shikfa, and Herrmann.....	23
1.	Overview of the Prior Art	23
2.	Motivations to Combine	31
3.	Claim 1	33
4.	Claim 16.....	62
5.	Claims 2/17: “...wherein the step of managing comprises allowing[/the management module further allows] the one or more users to download the searchable encrypted file to a local device”	69
6.	Claim 3: “wherein the local device comprises a mobile computing device.”	69
7.	Claim 8: “wherein the step of encrypting the file is carried out independent of the service provider of the one or more cloud platforms.”	69
8.	Claims 9/23: “...wherein the policy is implemented for a group of users.”	70
9.	Claims 10/24: “...wherein the policy defines the manner in which the file is stored, accessed, and processed.”	70
10.	Claims 11/25: “...wherein the file is stored across two or more cloud platforms.”	72
11.	Claim 12: “...wherein said storing the searchable encrypted file on the one or more cloud platforms is carried out independent of the service provider of the one or more cloud platforms.”	72
12.	Claims 13/26: “...wherein the policy is selected from a plurality of policies stored in a database, wherein the database is operatively coupled with the gateway device.”	74
13.	Claims 14/27: “...wherein the policy defines [/and manages] one or more of a set of encryption keys, a set of decryption keys, search indices, a type of encryption and one or more authorized locations to which the file may be stored.”	75

14.	Claim 21: “...wherein one or more of the policy assignment module, the encryption module, the storage module, and the management module are implemented on different devices, wherein the devices are selected from one or more of access management devices, proxy devices, gateway devices, and network controllers.”	79
15.	Claim 22: “...further comprising a mediation module, which, when executed by the one or more processors makes implementation of one or more of the policy assignment module, the encryption module, the storage module, and the policy management module independent of the service provider of the one or more cloud platforms.”	81
16.	Claim 26: “...wherein the group of policies are stored in a database operatively coupled with the system.”	83
B.	Ground 2: Claim 6 was obvious over Cidon, Shikfa, Herrmann, and Fellenstei.....	84
1.	Claim 6: “wherein the step of configuring the namespaces comprises storing content of the file as filenames.”	84
C.	Ground 3: Claim 15 was obvious over Cidon, Shikfa, Herrmann, and Hirai.....	85
1.	Claim 15: “...wherein said encrypting further comprises encrypting one or more of filename and search indices.”	85
D.	Ground 4: Claims 1-3, 8-19, 21-27 were obvious over Auradkar, Chiueh, and Chambers	87
1.	Overview of Prior Art	87
2.	Motivations to Combine	95
3.	Claim 1	96
4.	Claim 16.....	128
5.	Claims 2/17	139
6.	Claim 3.....	140
7.	Claim 8.....	140
8.	Claims 9/23	141

9.	Claims 10/24	141
10.	Claims 11/25	142
11.	Claim 12	143
12.	Claims 13/26	145
13.	Claim 14/27	147
14.	Claim 15	150
15.	Claim 21	151
16.	Claim 22	154
E.	Ground 5: Claim 6 was obvious over Auradkar, Chiueh, Chambers, and Inoue	157
1.	Claim 6: “wherein the step of configuring the namespaces comprises storing content of the file as filenames.”	157
II.	CONCLUSION.....	161

LIST OF APPENDICES

Appendix A	<i>Curriculum Vitae</i> of Dr. Wenke Lee
Appendix B	Louden, Bill (September 1983). "Increase Your 100's Storage with 128K from Compuserve," Portable 100, New England Publications Inc. 1 (1): 22. ISSN 0738-7016
Appendix C	"Dropbox Features," archived at the Internet Archive Wayback Machine on July 30, 201 at https://web.archive.org/web/20100730120646/http://www.dropbox.com/features
Appendix D	Muthitacharoen, A., Chen, B., & Mazieres, D. (2001, October). A low-bandwidth network file system. In <i>Proceedings of the eighteenth ACM symposium on Operating systems principles</i> (pp. 174-187)
Appendix E	Ghemawat, S., Gobioff, H., & Leung, S. T. (2003, October). The Google file system. In <i>Proceedings of the nineteenth ACM symposium on Operating systems principles</i> (pp. 29-43)

I. INTRODUCTION

1. I, Wenke Lee, have been retained by Petitioner Netskope, Inc. (“Petitioner”) to investigate and opine on certain issues relating to United States Patent No. 9,280,678 (“the ’678 patent”) in their Petition for *Inter Partes* Review. The Petition requests that the Patent Trial and Appeal Board (“PTAB” or “Board”) review and cancel claims 1-6, 8-19, and 21-27 of the ’941 patent.

2. The opinions set forth in this declaration are based on my personal knowledge, my professional judgment, and my analysis of the materials and information referenced in this declaration and its exhibits.

3. I am being compensated for consulting services including time spent testifying at any hearing that may be held. I am also reimbursed for reasonable and customary expenses associated with my work in this case. I receive no other forms of compensation related to this case. My compensation does not depend on the outcome of this post-grant review or the co-pending district court litigation, and I have no other financial interest in this post-grant review.

4. I understand that the ’678 patent has been assigned to Fortinet, Inc.

5. This declaration is based on the information currently available to me. To the extent that additional information becomes available, I reserve the right to continue my investigation and study, which may include a review of documents and information that may be produced, as well as testimony from depositions that

have not yet been taken.

A. Qualifications

1. Education

6. I obtained a Bachelor of Science in Computer Science from Sun Yat-Sen University in China in 1988. In 1990, I obtained my Master of Science in Computer Science from the City College of New York. In 1999, I obtained my Ph.D. in Computer Science from Columbia University.

2. Work Experience

7. I am currently a Professor of Computer Science at the Georgia Institute of Technology, focusing on systems and network security, applied cryptography, and machine learning. Since 2016, I have also been the John P. Imlay Jr. Chair in the College of Computing at the Georgia Institute of Technology.

8. Since I obtained my Master of Science in Computer Science in 1990 and to 1994, I was a software engineer at Intergraph Corporation.

9. From 1994 to 1999, I was a Ph.D. student in Computer Science. My thesis research focused on machine learning and cybersecurity. While I was a Ph.D. student, I was a Summer Intern researcher at AT&T Research in 1996 and IBM T. J. Watson Research Center in 1997.

10. From 1999 to 2001, I was a tenure-track Assistant Professor in the Department of Computer Science at The North Carolina State University.

11. From 2001 to 2005, I was a tenure track Assistant Professor in the College of Computing at The Georgia Institute of Technology (Georgia Tech). In 2005, I was promoted to Associate Professor with tenure. In 2009, I was elevated to Full Professor with tenure.

12. From 2012 to 2015, I was the Director of the Information Security Center at the Georgia Institute of Technology. From 2015 to 2021, I was the Executive Director of the Institute for Information Security and Privacy at the Georgia Institute of Technology.

13. In 2006, I co-founded Damballa, Inc. to commercialize the botnet and malware detection technologies developed in my lab. Damballa was sold to Core Security in 2016.

14. I have been elevated to Fellow of the Institute of Electrical and Electronics Engineers (IEEE), the global engineering society. Fellow is the highest of three grades of membership that are awarded based on merit. In every year, IEEE limits the number of new Fellows to one tenth of one percent of the membership.

15. I have also been elevated to Fellow of the Association for Computing Machinery (ACM), the global professional society for computer scientists. ACM's rules for Fellows are even more restrictive than IEEE's, limiting the total number of Fellows in absolute terms to 1% of the membership.

16. Over the course of my career so far, I have been the Principal Investigator or co-Investigator on 50 approved and funded grants. My expertise both as a developer and researcher on computer systems and information security, cybersecurity and machine learning, computerized intrusion detection methods, applied cryptography, data mining, digital watermarking, and packet watermarking has been sought out by the Federal Government. For example, I have been the Principal Investigator on various grants funded by the Defense Advanced Research Projects Agency (DARPA), the Office of Naval Research (ONR) and the Department of Homeland Security (DHS).

17. From 2010 to 2011, I served on the Editorial Board of one of the flagship journals of the IEEE, the IEEE Transactions on Dependable and Secure Computing (TDSC). From 2005 to 2011, I served on the Editorial Board of an ACM journal, the ACM Transactions on Information and System Security (TISSEC). I have also served on more than 90 conference committees and have served on more than a dozen of committees at the Georgia Institute of Technology. I have also served as the primary advisor to 8 Postdoctoral Fellows and 25 completed Ph.D.s, and currently serve as the primary advisor on more than ten further dissertations in progress.

18. The area of my current research is malware analysis and detection. A fuller description of my current interests and areas of research can be found at the

bottom of page 2 and top of page 3 of my CV.

19. As reflected in the succeeding pages of my CV (pp. 3 - 6), I have taught courses directed to Network and Information Security essentially continuously since 1999.

3. Publications

20. I have published more than a hundred academic papers in top journals and conference proceedings, and participated in well over a hundred conference presentations and over a dozen workshop presentations, the vast preponderance of which relate to Network Security and most of which relate to Intrusion Detection in particular.

21. I am a named inventor in ten issued patents, most of which relate to the field of invention at hand.

4. Curriculum Vitae

22. A copy of my curriculum vitae is attached as Appendix A to this declaration.

B. Materials Reviewed

23. My opinions expressed in this declaration are based on documents and materials identified in this declaration, including the '678 patent, the prior art references and background materials discussed in this declaration, and the other references specifically identified in this declaration. I have considered these materials in their entirety, even if only portions are discussed here. The following

is an exemplary list of the materials on which I based on opinion.

Exhibit	Description
Ex. 1001	U.S. Patent No. 9,280,678 to Redberg (“the ’678 Patent”)
Ex. 1002	Supporting Declaration of Dr. Wenke Lee (“Lee”)
Ex. 1003	File History of U.S. Patent No. 9,280,678
Ex. 1004	U.S. Patent Application Publication No. 2014/0013112 (“Cidon”)
Ex. 1005	U.S. Patent Application Publication No. 2003/0055994 (“Herrmann”)
Ex. 1006	U.S. Patent Application Publication No. 2014/0359282 (“Shikfa”)
Ex. 1007	U.S. Patent Application Publication No. 2005/0010593 (“Fellenstein”)
Ex. 1008	U.S. Patent No. 6,622,248 (“Hirai”)
Ex. 1009	U.S. Patent Application Publication No. 2011/0119481 (“Auradkar”)
Ex. 1010	U.S. Patent Application Publication No. 2014/0068030 (“Chambers”)
Ex. 1011	U.S. Patent Application Publication No. 2013/0159694 (“Chiueh”)
Ex. 1012	U.S. Patent Application Publication No. 2003/0063321 (“Inoue”)
Ex. 1013	U.S. Patent Application Publication No. 2002/0099666 (Dryer)
Ex. 1014	Dictionary of Computer and Internet Terms, Tenth Edition (2009)
Ex. 1015	U.S. Patent Application Publication No. 2011/0145594 (“Ex. 1015”)
Ex. 1016	U.S. Patent Application Publication No. 2009/0300351 (“Ex. 1016”)
Ex. 1017	U.S. Patent Application Publication No. 2008/0069480 (“Ex. 1017”)
Ex. 1018	D. Song, D. Wagner, A. Perrig, Practical Techniques For Searches On Encrypted Data, Proceeding 2000 IEEE Symposium on Security and Privacy, S&P 2000, May 14, 2000 (“Song”)

Ex. 1019	R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order Preserving Encryption For Numeric Data, In Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, Paris, France, June 2004 (Agrawal)
Ex. 1020	Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. 2002. Farsite: federated, available, and reliable storage for an incompletely trusted environment. SIGOPS Oper. Syst. Rev. 36, SI (Dec. 2002), 1-14 (“Ex. 1020”)
Ex. 1021	Mika, P., & Potter, T. (2012). Metadata Statistics for a Large Web Corpus. LDOW, 937 (“Ex. 1021”)
Ex. 1022	Microsoft Computer Dictionary, Fifth Edition (2002)
Ex. 1023	Balakrishnan, H., Lakshminarayanan, K., Ratnasamy, S., Shenker, S., Stoica, I., & Walfish, M. (2004). A layered naming architecture for the Internet. ACM SIGCOMM Computer Communication Review, 34(4), 343-352 (“Ex. 1023”)

24. I have also relied on my own experience and expertise in Web technologies, distributed (client-server) service architectures, and computer network security.

C. Level of Ordinary Skill in the Art

25. I am not an attorney and offer no legal opinions. I have been informed about certain aspects of the law for purposes of my analyses and opinions.

26. I understand that in analyzing questions of invalidity and infringement, the perspective of a person having ordinary skill in the art (“POSITA”) is often implicated, and the Court may need assistance in determining that level of skill.

27. I understand that the claims and written description of a patent must

be understood from the perspective of a POSITA. I have been informed that the following factors may affect the level of skill of a POSITA: (1) the educational level of the inventor; (2) the type of problems encountered in the art; (3) the prior-art solutions to those problems; (4) the rapidity with which innovations are made; (5) the sophistication of the technology; and (6) the educational level of active workers in the field. A person of ordinary skill in the art is also a person of ordinary creativity in the art.

28. Based on my experience in client-server system design, as well as my reading of the '678 Patent, it is my opinion that a person of ordinary skill with respect to the subject matter of the '678 Patent at the time of the alleged priority date of the '678 Patent in December 2013 would have had at least a bachelor's degree in computer science, computer engineering, or electrical engineering and would have had at least two years of experience working on network security design and related applications. This definition is flexible, and additional educational experience could make up for less work experience and vice versa.

29. I am at least a person of ordinary skill in the art and was so on the date to which the '678 Patent claims priority. As shown by my qualifications and my curriculum vitae attached as Appendix A, I am aware of the knowledge and skill possessed by a person of ordinary skill in the art at the time of the priority date of the '678 Patent. In performing my analysis, I have applied the standard set forth

above.

D. Summary of Opinions

30. I have reviewed and analyzed the '678 Patent.

31. Based on my review and analysis, it is my opinion that claims 1-6, 8-19, 21-27 of the '678 Patent are invalid based on the following grounds:

Ground	Basis	Reference(s)	Challenged Claims
1	§103	Cidon, Shikfa, Herrmann	1-5, 8-14, and 16-19, 21-27
2	§103	Cidon, Shikfa, Herrmann, Fellenstei	6
3	§103	Cidon, Shikfa, Herrmann, Hirai	15
4	§103	Auradkar, Chiueh, Chambers	1-5, 8-19, 21-27
5	§103	Auradkar, Chiueh, Chambers, Inoue	6

II. OVERVIEW OF THE TECHNOLOGY

A. Priority Date of the Claims

32. I have been informed that a U.S. patent application may claim the benefit of the filing date of an earlier patent application if the earlier patent application disclosed each limitation of the invention claimed in the later-filed U.S. patent application. I have also been informed that priority is determined on a claim-by-claim basis so that certain claims of a patent may be entitled to the priority date of an earlier-filed patent application even if other claims of the same patent are not entitled to that priority date.

33. I have also been informed that a patented claim is invalid if the

claimed invention was patented, described in a printed publication, or in public use, on sale, or otherwise available to the public before the effective filing date of the claimed invention, or the claimed invention was described in an issued patent or a published patent application that was effectively filed before the effective filing date of the claimed invention. For the purposes of this declaration, I have presumed the priority date of the '678 patent to be May 10, 2018, which is the filing date of the '678 patent listed on the front page of the '678 patent.

B. Overview of Relevant Technology When the '678 Patent Was Filed

1. Cloud Storage

34. Cloud storage has been around since at least 1983, when CompuServe let users upload files to online disk space.

**INCREASE YOUR
100'S STORAGE WITH
128K FROM COMPUERVE**

Having access to 128K of storage from any phone in North America would be a real plus for any portable computer owner, but for the Model 100 owner with CompuServe it's a reality.

Make sure the DIR/ACP (direct/acoustic coupler) switch on the left side of the computer is on DIR and the ORG/ANS (originate/answer) switch is on ORG. Disconnect the telephone line from the telephone

(Appendix B, p. 22, excerpt; annotated)

35. Cloud storage became popular with consumers in the late 2000s. For example, Dropbox had over 1 million users by April 2009. By 2010, Dropbox also offered the ability to search encrypted files.”



Dropbox Features

Web Access

A copy of your files are stored on Dropbox's secure servers. This lets you access them from any computer or mobile device.

Manipulate files as you would on your desktop - add, edit, delete, rename etc.

Search your entire Dropbox for files.

A "Recent Events" feed that shows you a summary of activity in your Dropbox.

Create shared folders and invite people to them.

Recover previous versions of any file or undelete deleted files.

Security & Privacy

Dropbox takes the security and privacy of your files very seriously.

Shared folders are viewable only by people you invite.

All transmission of file data and metadata occurs over an encrypted channel (SSL).

All files stored on Dropbox servers are encrypted (AES-256) and are inaccessible without your account password.

Dropbox website and client software have been hardened against attacks from hackers.

Dropbox employees are not able to view any user's files.

Online access to your files requires your username and password.

Public files are only viewable by people who have a link to the file(s). Public folders are not browsable or searchable.

(Appendix C, excerpt; annotated)

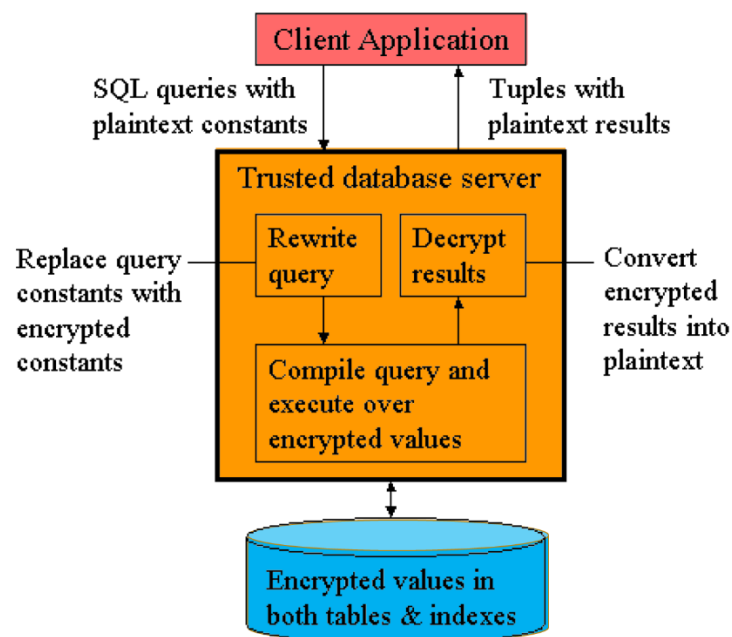
2. Searchable Encryption

36. Searchable encryption is a cryptographic technique that allows a user to search through encrypted data without decrypting the data. This concept can be traced back to the late 1980s, when researchers began exploring the possibility of searching through encrypted data.

37. In the 1990s, several researchers developed more advanced searchable encryption schemes, including "keyword search scheme" introduced by Song et al.

in 2000. (Song.) This scheme allowed users to search documents stored in a remote server using an encrypted index containing encrypted keywords and corresponding pointers to documents where the keywords appear, without revealing corresponding plaintext keywords to the searcher.

38. Agrawal et al. in 2004 proposed a searchable encryption scheme for encrypting and searching records stored in a remote untrusted storage (**database**) using a trusted **database server** as a proxy between a **client** device and the untrusted storage. (Agrawal, §3, FIG. 1.)



Agrawal, FIG. 1 (annotated)

39. In operation, the database server received a query having plaintext from a client application, and encrypted the query, and searched the database using encrypted query without revealing the content to be searched to the database. (*Id.*,

§3.)

40. Since the early 2000s, searchable encryption has continued to evolve and has been applied to a variety of applications, including secure file storage systems, secure email systems, and secure web search engines.

3. Namespace

41. Using a naming convention to identify objects was introduced decades ago and used as a conceptual framework for organizing and managing data and resources. For example, Adya et al. in 2002 described a storage system using a file system as a hierarchical directory namespace. (Ex. 1020.) As another example, Mika et al. tabulated the most commonly used namespaces for URLs in its Table 5 (see below), illustrating that a namespace for URLs can have different sections, such as a domain name (e.g., www.facebook.com) and a subdirectory (e.g., 2008). (Ex. 1021.)

Namespace	URLs
ogp.me/ns#	493,443,016
www.facebook.com/2008/	150,246,016
www.w3.org/1999/02/22-rdf-syntax-ns#	26,402,165
rdf.data-vocabulary.org/#	19,413,470
purl.org/dc/terms/	16,424,800
https:// www.facebook.com/2008/	7,472,815
mixi-platform.com/ns#	6,323,861
ogp.me/ns/fb#	4,636,260
creativecommons.org/ns#	4,622,272
www.w3.org/2006/vcard/ns#	4,205,037
http://	3,881,321
http://www.facebook.com/	3,126,045
http://www.w3.org/2000/01/rdf-schema#	3,042,839
http://developers.facebook.com/schema/	2,720,567
http://search.yahoo.com/searchmonkey/commerce/	2,664,743
http://purl.org/dc/elements/1.1/	2,642,796
http://opengraphprotocol.org/schema/	2,293,024
http://search.yahoo.com/searchmonkey/media/	2,095,577
http://oexchange.org/spec/0.8/rel/	2,034,467
http://xmlns.com/foaf/0.1/	1,837,749

Table 5: Top namespaces in RDFa as measured by the number of URLs

42. A namespace ensures that every object within the context of the namespace can be identified uniquely by its name(s). But namespaces need not be hierarchical or complex. For example, in the above table, one namespace was simply “http://.” In fact, namespaces can have a flat structure forming a single collection of unique names. (Ex. 1022.) As explained in Balakrishnan et al. (2004), a namespace in a flat structure can simply be a number chosen from within a range (e.g., 0 to $2^{128}-1$). (Ex. 1023, p. 344.) As another example, a namespace can provide the scope or context of identifiers, e.g., variables, functions, etc., in a program such that an identifier is referred to by its unique short, local name within

the namespace and by its unique fully qualified name (e.g., that includes the name of the scope) from outside the namespace. Other examples of namespace include assigning names and groupings or scopes to resources such as websites, servers, etc.

4. Gateway

43. In computer networking history, the concept of a gateway was introduced as early as 1976. In general, a gateway refers to a networking device that allows and processes traffic from one network to another.

44. Gateways were ubiquitous before the '678 Patent's 2013 priority date, and a variety of gateways had been developed, depending on the network protocol or application of interest. For example, an application-level gateway at the border of an enterprise network could intercept the traffic between a client computer within the enterprise network and an application server (e.g., a website) on the Internet, and could modify or disallow data to/ from the application server according to the data access and security policies of the enterprise network.

45. As another example, a cloud storage gateway had been developed no later than 2010 as a networking server that translated requests from client computers, e.g., requests for file storage and access, into the corresponding data requests to the cloud storage server. It could enforce client-specific policies, such as data protection policies to allow client computer applications to access data in

third-party cloud storage in a secure manner.

5. File Chunking

46. Chunking refers to the process of splitting a file into smaller pieces, each called a chunk, for efficient data storage, retrieval, and transmission.

Chunking has been used in many aspects of computing for decades. (See, for example, Appendices D and E.)

47. A POSITA would have understood that a file could be partitioned into multiple chunks, and the file could be stored in the cloud on a chunk-by-chunk basis. And accessing files stored in the cloud could also be on a chunk-by-chunk basis.

48. A POSITA would also have understood that when search encryption was used, the file chunks were encrypted, and the encrypted chunks and encrypted searchable indices (e.g., an index file that listed for each encrypted or hashed keyword, or the filenames of the chunks that contained the keyword) could be searched to identify any individual chunks that matched a search query.

C. The '678 Patent Patent Overview

1. Alleged invention

49. The claimed invention is directed to the well-known concept of searchable encryption schemes using encrypted keyword indices.

50. Figure 1 of the '678 Patent illustrates a **cloud gateway 108** that “acts as an interface between the **clients 102** and **stores 114**” and handles “different

file/data read/write requests received from clients 102” to “identify the appropriate set of stores 114 that need to be accessed for processing the requests.”¹ (’678 Patent, FIG. 1, 7:9-14.)

¹ All **bold** or *italic* emphases that appear in quoted text are added by Petitioner unless otherwise specified.

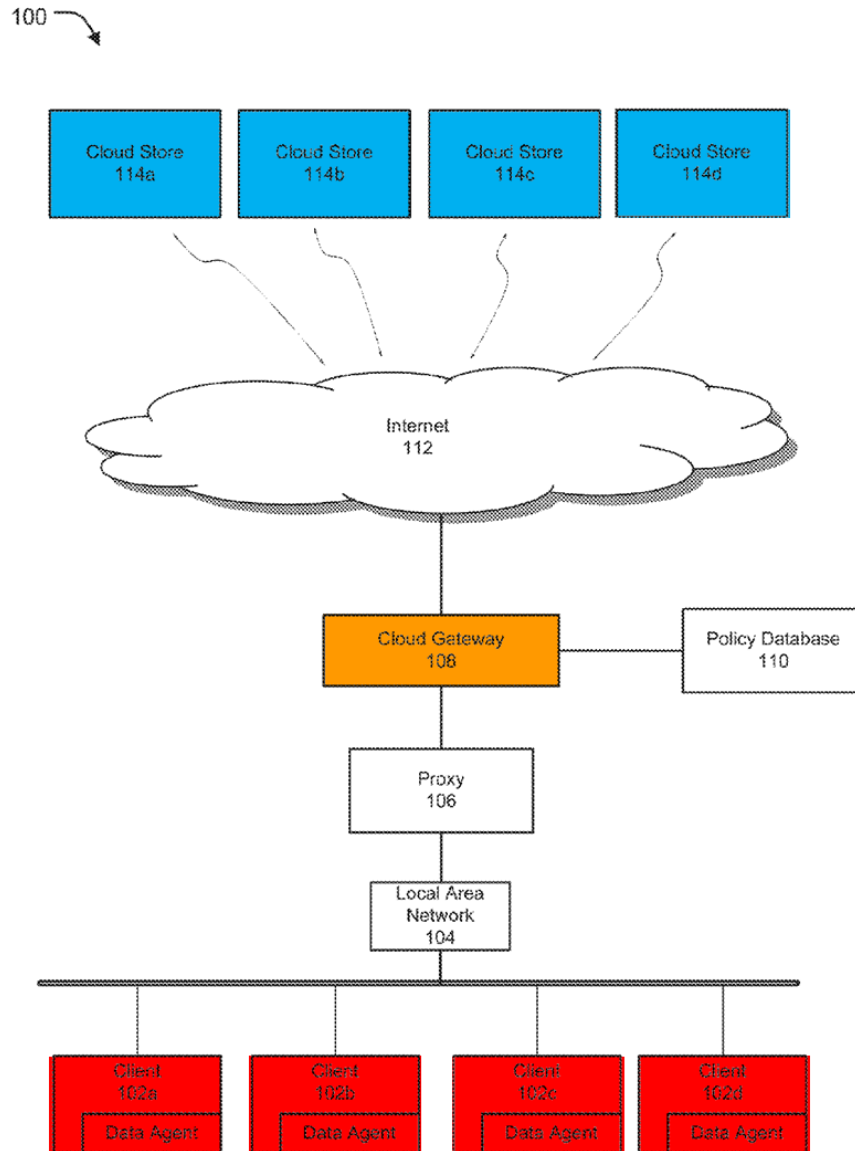


Fig. 1(Annotated)

51. For example, in connection with the flowchart shown in Figure 3A below, the gateway receives a file to be encrypted from a user (step 302), partitions the file into chunks (step 304), and encrypts each of the chunks “such that [n]amespace is [c]reated for each [f]ile [c]hunk” (step 306). (’678 Patent, FIG. 3A, 11:49-64.) The gateway also encrypts “keywords” (“search terms”) into

“searchable indices” and stores them as filenames of the chunks. (’678 Patent, 12:7-18.) The gateway further stores the encrypted chunks to one or more cloud stores. (’678 Patent, 12:24-26.)

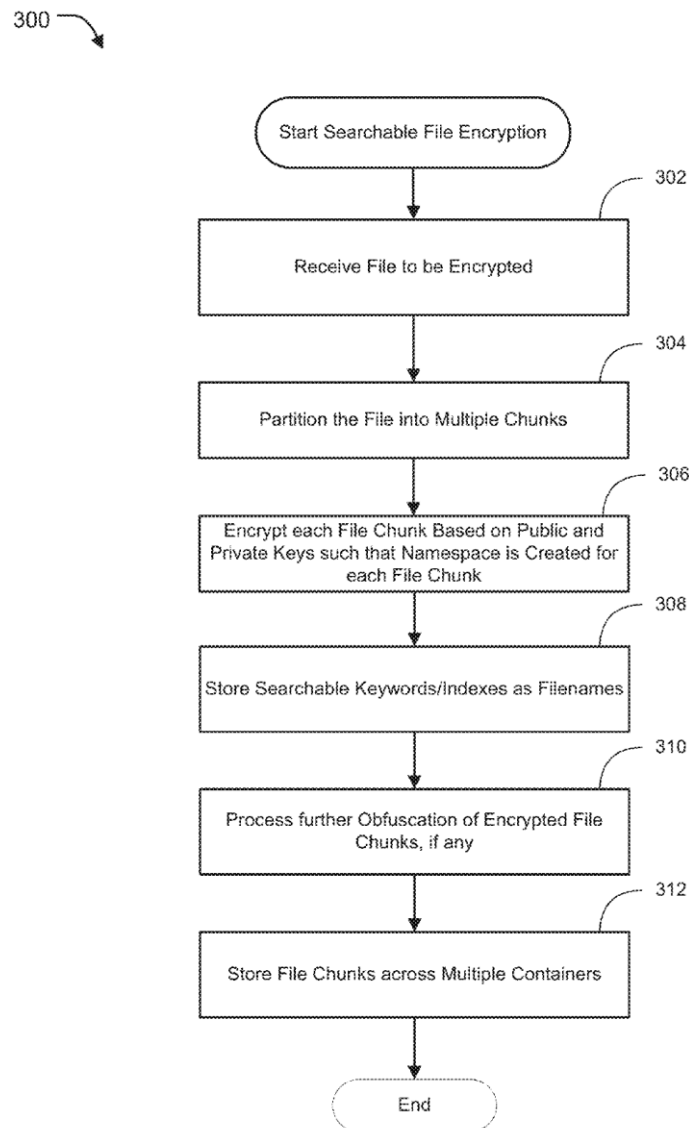


Fig. 3A

52. To search encrypted files, the gateway receives a search query and extracts keywords from the query. (’678 Patent, 12:32-55, FIG. 3B.) The gateway also converts the keywords into searchable indices (i.e., encrypted keywords),

identifies the stored chunks that match searchable indices, and returns any identified chunk(s). ('678 Patent, 12:32-55, FIG. 3B.)

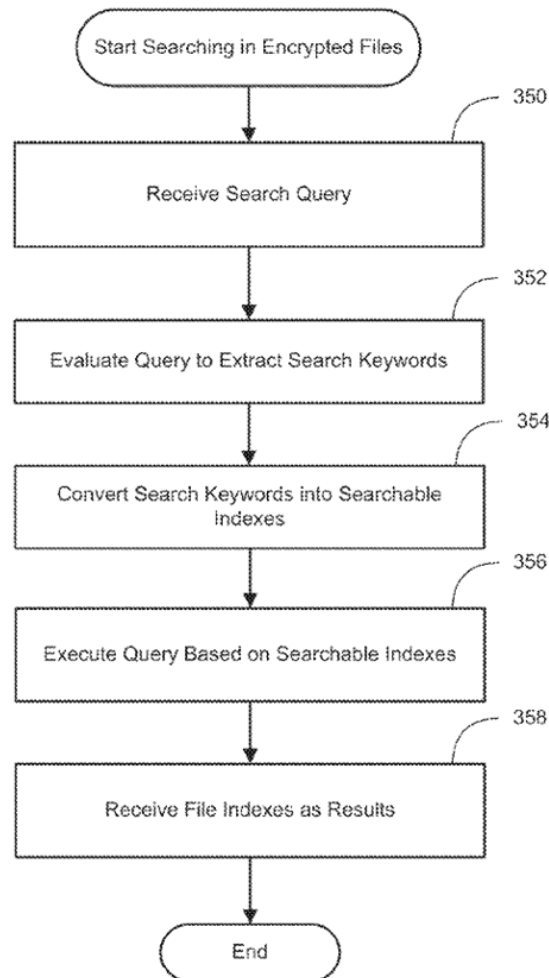


Fig. 3B

53. Additionally, the gateway, via its module, “facilitate[s] assignment of a policy selected from the policy database 218 to one or a group of users 202, where the policy to be assigned can be selected, by an administrator, for example, based on the role, responsibility, enterprise practices, among other attributes.” ('678 Patent, 9:11-16.)

D. The Challenged Claims

54. I understand that the Petitioner is challenging claims 1-3, 6, 8-19, and 21-27 in the '678 Patent.

E. Claim Construction

55. I understand that claim terms generally are construed in accordance with the ordinary and customary meaning they would have to a POSITA at the time of the invention in light of the claim language, the specification, and the prosecution history. I understand that dictionaries and other extrinsic evidence may be considered as well, though such evidence is typically regarded as less significant than the intrinsic record in determining the meaning of the claim language

56. For all terms of the challenged claims of the '678 patent, I have interpreted them as they would have been understood by a POSITA at the time of the priority date of the '678 Patent in 2013.

III. UNPATENTABILITY OF THE '678 PATENT CLAIMS

F. Standards for Invalidity

1. Obviousness

57. I am informed and understand that a patent cannot be properly granted for subject matter that would have been obvious to a person of ordinary skill in the art at the time of the alleged invention, and that a patent claim directed to such obvious subject matter is invalid under 35 U.S.C. § 103. It is also my

understanding that in assessing the obviousness of claimed subject matter, one should evaluate obviousness in light of the prior art from the perspective of a person having ordinary skill in the art at the time the alleged invention was made (and not from the perspective of either a layman or a genius in that art). It is my further understanding that the question of obviousness is to be determined based on:

- The scope and content of the prior art;
- The difference or differences between the subject matter of the claim and the prior art (whereby in assessing the possibility of obviousness one should consider the manner in which a patentee and/or a Court has construed the scope of a claim);
- The level of ordinary skill in the art at the time of the alleged invention of the subject matter of the claim; and
- Any relevant objective factors (the “secondary indicia”) indicating nonobviousness, including evidence of any of the following: commercial success of the products or methods covered by the patent claims; a long-felt need for the alleged invention; failed attempts by others to make the alleged invention; copying of the alleged invention by others in the field; unexpected results achieved by the alleged invention; praise of the alleged invention by the alleged

infringer or others in the field; the taking of licenses under the patent by others and the nature of those licenses; expressions of surprise by experts and those skilled in the art at the subject matter of the claim; and whether the patentee proceeded contrary to accepted wisdom of the prior art.

- Any relevant objective factors (the “secondary indicia”) indicating obviousness: independent invention of the claimed invention by others before or at about the same time as the named inventor thought of it; and other evidence tending to show obviousness.

IV. SPECIFIC GROUNDS FOR PETITION

A. Ground 1: Claims 1-3, 8-14, and 16-27 were obvious over Cidon, Shikfa, and Herrmann

1. Overview of the Prior Art

Cidon:

58. Like the '678 Patent, Cidon were directed to systems and methods for implementing policies for encrypting files, storing encrypted files to cloud-based storage devices like Dropbox, iCloud, and accessing encrypted files. (Cidon, Abstract, ¶¶83, 100.)

59. For example, Cidon described a **management server** that could access user accounts, enterprise file systems, and various consumer file

repositories, such as **Dropbox 13**, **Gmail 11**. (Cidon, ¶100, FIG. 2.) Cidon's management server was responsible for "encrypting files in the cloud and enforcing secure access to these cloud files by providing the right decryption keys to the right users." (Cidon, ¶ 102, FIG. 2.)

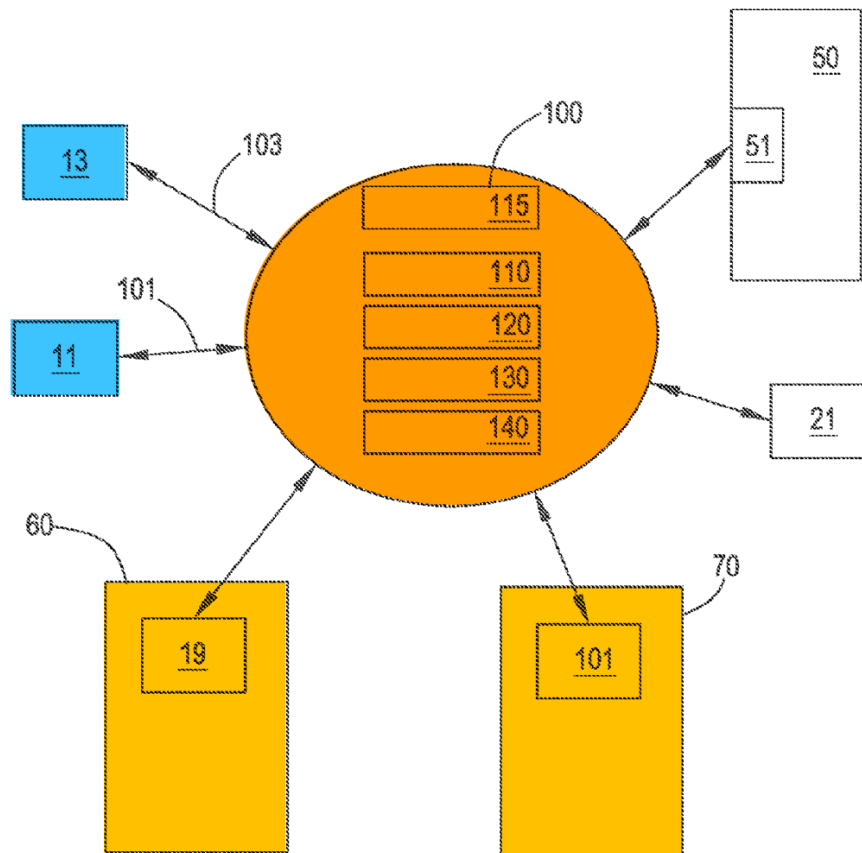


FIG. 2 (Annotated)

60. To encrypt a **file** and store the **encrypted file** to a cloud storage, the management server divided the **file** into multiple **file segments** (step 1620) and encrypted the file segments to "provide multiple encrypted file segments" (step 1630) which "formed an encrypted file." (Cidon, FIGS. 13, 15, ¶¶305, 309.) The

management server also sent the encrypted file to a cloud storage service. (Cidon,
¶309.)

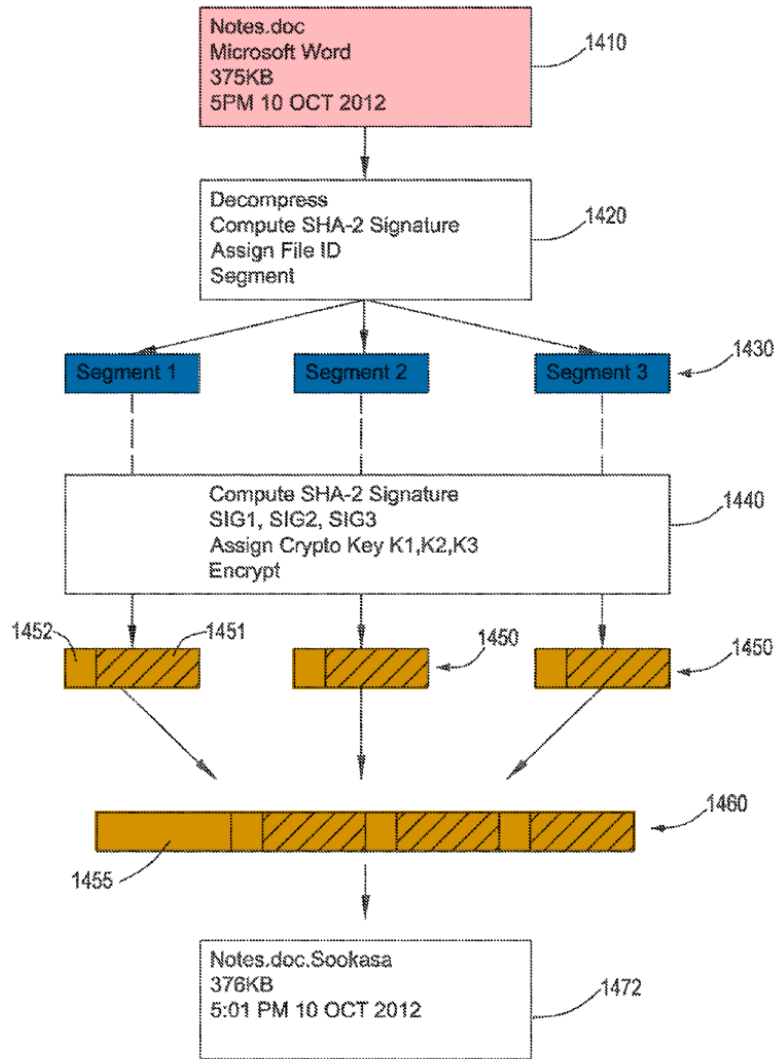


FIG. 13 (Annotated)

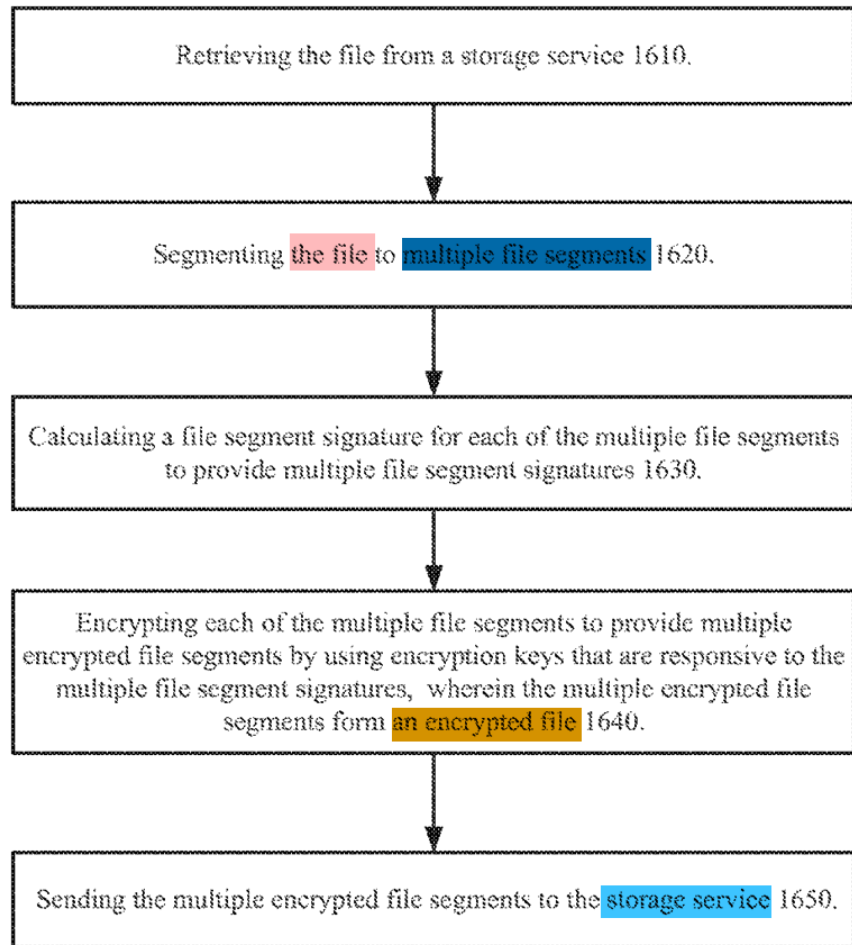


FIG. 15 (Annotated)

61. The management server also “provide[d] access policy configuration for secure file access” for, e.g., users of **user devices 60** and **70**, and “enforce[d] access policy for secure file access.” (Cidon, ¶102, FIG. 2.)

Shikfa:

62. Like the '678 Patent and Cidon, Shikfa was also directed to methods

for encrypting files and storing the encrypted files to cloud storages. (Shikfa, ¶3.)

Shikfa disclosed a “broker system [that] transfer[red] the **encrypted documents** between the **client** and the **storage providers** and to maintain a table indicating the encrypted documents that [were] transferred to each storage provider.”² (Shikfa, ¶3, FIG. 2 (below).)

63. Notably, Shikfa disclosed configuring identifiers (namespaces) which allowed encrypted documents to be searched. (Shikfa, ¶36, FIG. 2(below).). Specifically, Shikfa’s system configured and used these identifiers to map encrypted documents to search keywords, which allowed the encrypted documents to be searched and identified. (Shikfa, ¶¶29, 31.) Shikfa also described that its system can store “***part of a particular encrypted document***” to a cloud storage provider to “guarantee availability of the stored data.” (Shikfa, ¶39.)

² All **bold** or *italic* emphases that appear in quoted text herein are added by Petitioner unless otherwise specified.

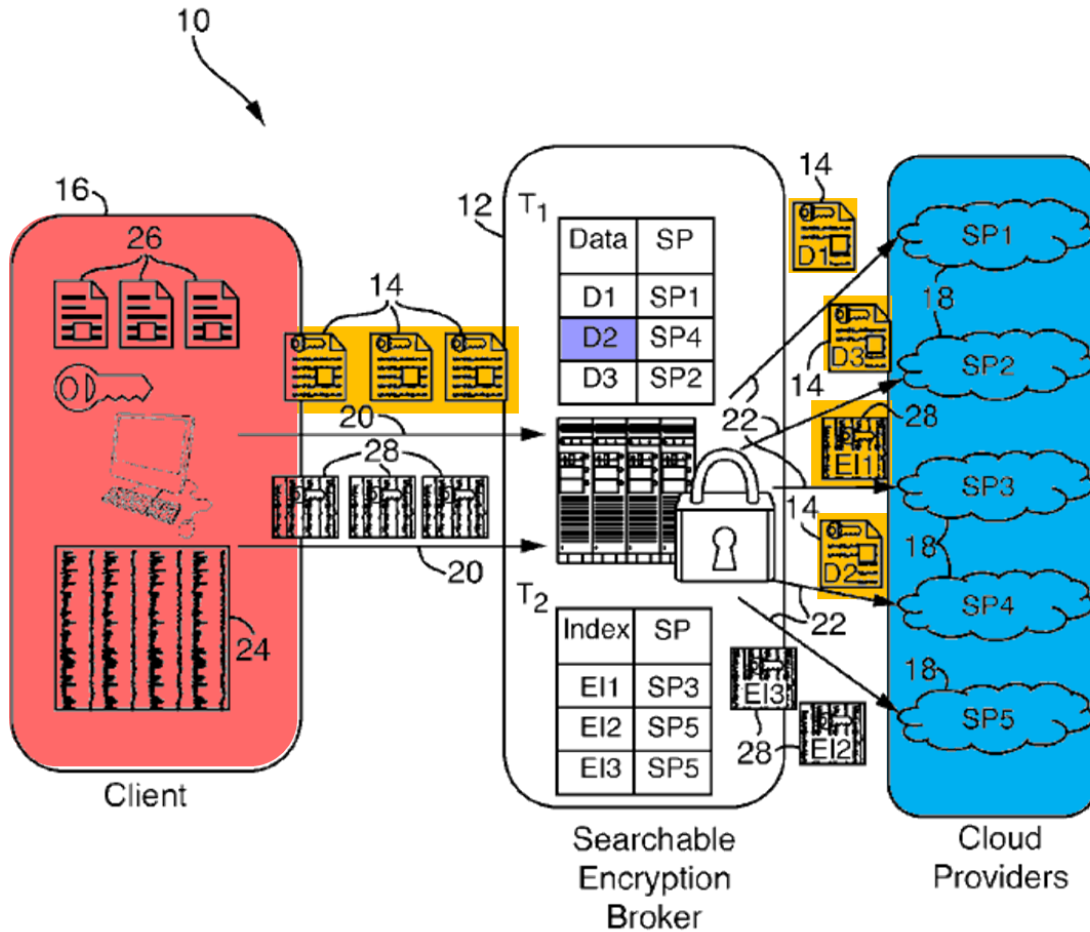


FIG. 2 (Annotated)

64. Like the gateway device of the '678 Patent, when searching on encrypted files, Shikfa's broker system used an encrypted query to search an encrypted index that contained encrypted keywords to identify the encrypted documents stored on the cloud storages. (Shikfa, ¶12, FIG. 3.) The broker system also returned the identified encrypted documents to the client. (Shikfa, ¶12, FIG. 3.)

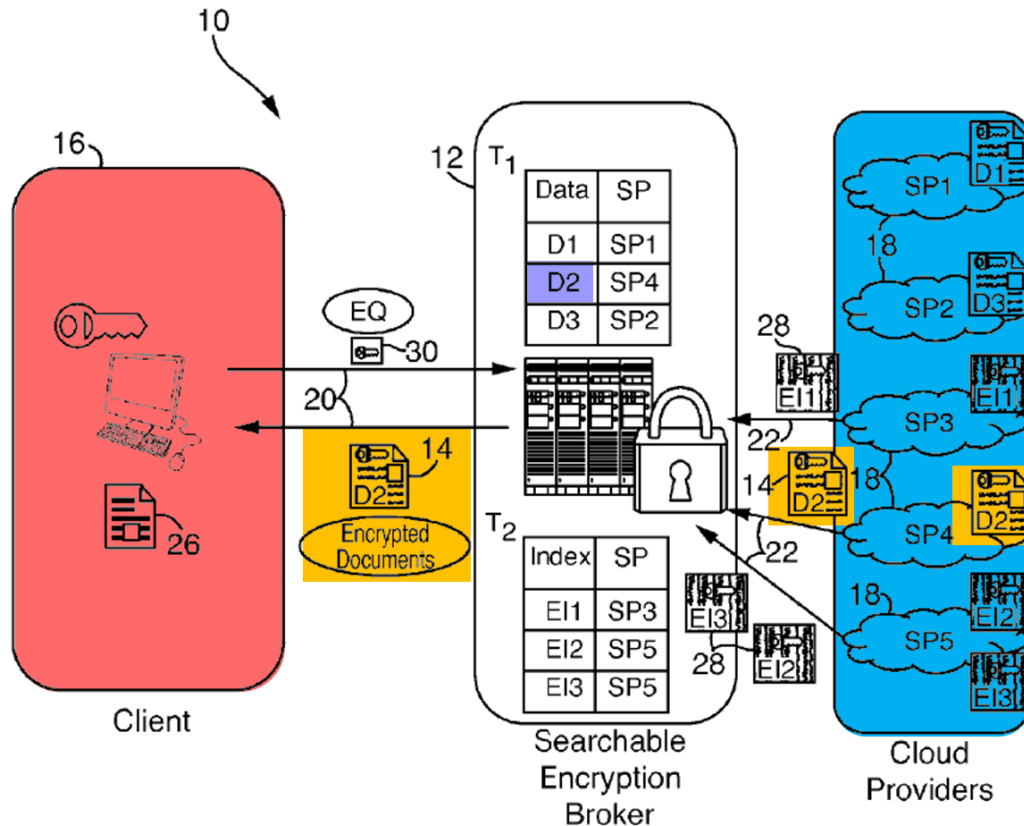


FIG. 3 (Annotated)

Herrmann:

65. Like the '678 Patent and Cidon, Herrmann was concerned with implementing policies for accessing data stored in a protected resource by client devices. (Herrmann, Abstract.)

66. Herrmann disclosed a gateway device (e.g., **gateway server 350** in Figure 3 below) that “act[ed] as a gateway (e.g., a VPN gateway) to a group of resources or collection of data (e.g., the **protected data or resources 390**) and accept[ed] requests for access to such resources from **client machines.**”

(Herrmann, ¶60, FIG. 3.) Herrmann also disclosed a policy store 371, where “appropriate policy specification 373 applicable to the client computer 310 [could be retrieved] from the policy store 371.” (Herrmann, ¶67.) Herrmann’s gateway server 350 “assign[ed] an **access policy** to the connection/session between the gateway server 350 and the client computer 310).” (Herrmann, ¶77.)

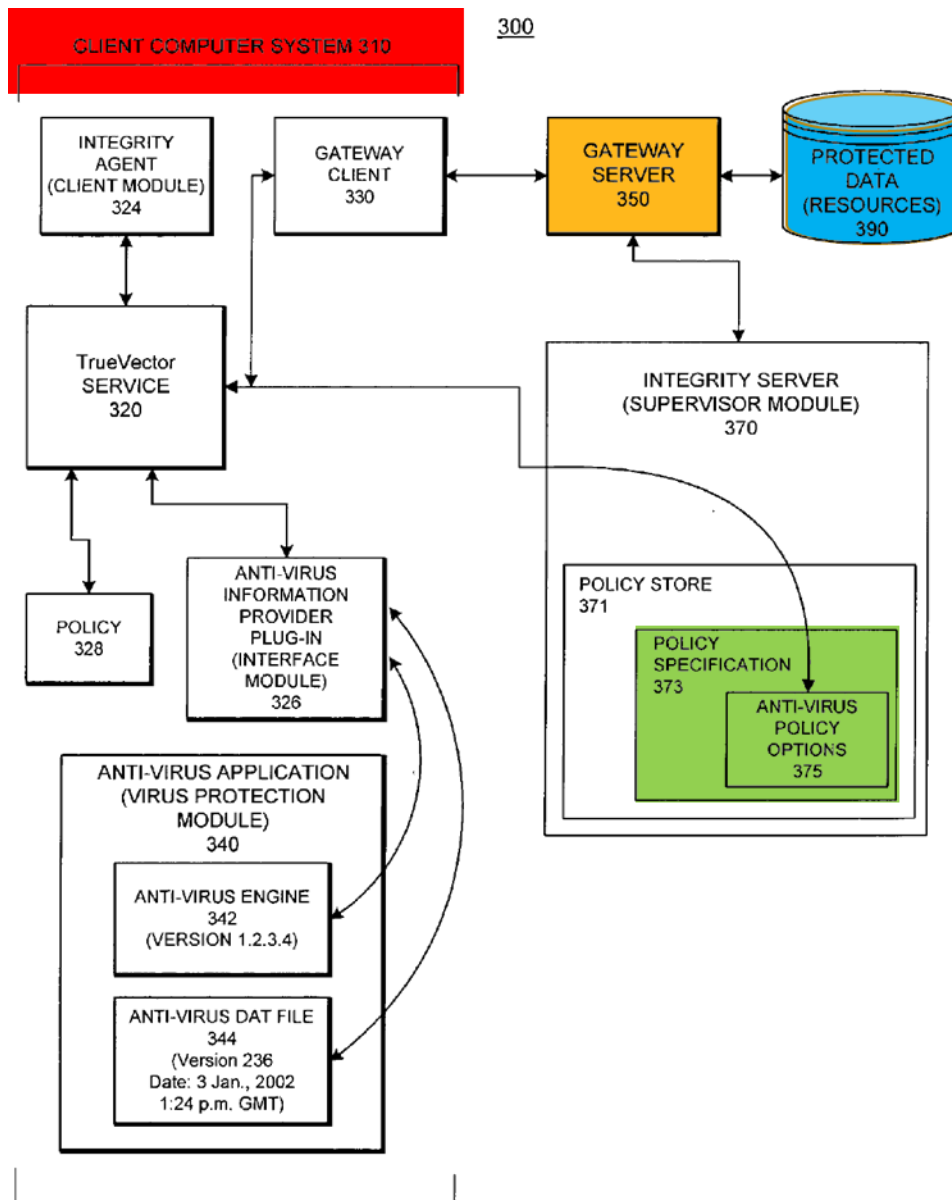


FIG. 3 (Annotated)

2. Motivations to Combine

67. The Cidon-Herrmann-Shikfa combination would have entailed implementing the searchable encryption procedure of Cidon using Herrmann's gateway device, and modifying the searchable encryption methodology per

Shikfa's teachings to allow configuring the namespaces for searching the encrypted documents. A POSITA would have been motivated to combine Cidon, Shikfa, and Herrmann, with a reasonable expectation of success, at least for the following reasons.

68. First, all Cidon, Shikfa, and Herrmann were all concerned with improving computer networking and security for accessing files stored in a remote storage location by client devices. (Cidon, ¶100; Shikfa, ¶3; Herrmann, Abstract.) Thus, a POSITA would have found it obvious that Cidon's system could have benefited from Shikfa and Herrmann's complementary techniques with the same or similar goal.

69. Second, the Cidon-Shikfa-Herrmann combination would have also produced a better overall system. For example, while Cidon's system allowed searching encrypted files, a POSITA would have recognized that adding Shikfa's searchable encryption mechanism into Cidon's system would have enhanced the search function of Cidon's system. Specifically, the modified system would have allowed searching encrypted documents stored in a cloud storage that did not have a search capability on a chunk-by-chunk basis. (Cidon, ¶¶91, 166, 171; Shikfa, ¶2.) Additionally, both Cidon and Herrmann aimed to provide flexibility for implementing different policies for different users to access protected data. (Cidon, ¶106; Herrmann, ¶50.) A POSITA would have understood that incorporating

Herrmann's gateway and its functionalities for assigning access policies to Cidon's system would have allowed automatic policy assignment and implementation, which would have improved Cidon's system efficiency and reduced human errors in assigning policies. (Herrmann, ¶77.) Additional motivations are discussed below.

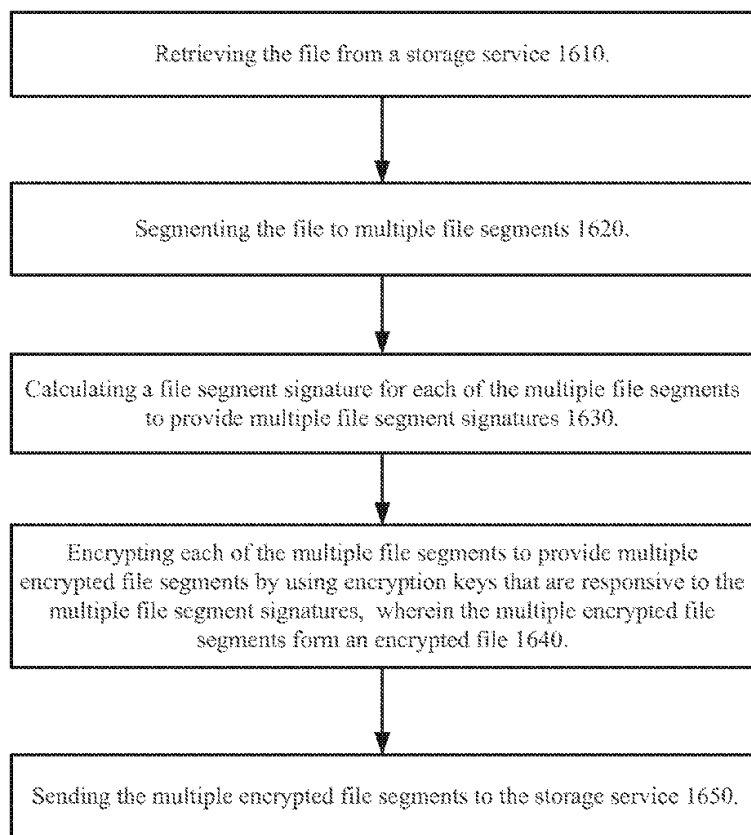
3. Claim 1

70. As I will explain below, in my opinion, the combination of Cidon, Herrmann, and Shikfa rendered obvious claim 1.

a. Claim 1[pre]: "A method comprising:"

71. Cidon disclosed a method for encrypting files and storing encrypted files in cloud storages.

72. For example, Cidon disclosed a method for "encrypting a file," including "segmenting the file into multiple file segments...encrypting each of the multiple file segments to provide multiple encrypted file segments...;...and sending the multiple encrypted file segments to the storage service." (Cidon, ¶¶10, 309, FIG. 15.)



1600

FIG. 15

- b. Claim 1[a][i]: “assigning to one or more users, by a gateway device, a policy for managing access to and processing a file to be stored on one or more cloud platforms,”**

73. Cidon disclosed a **management server** that performed various functions, including providing access policy configuration for one or more **users**, implementing policies, and processing a file to be stored on **cloud platforms**, such as “Dropbox, Box, Sugarsync or a web-based storage system such the one offered

by Amazon S3 or EMC, ...Google Docs, Office 365, Evernote and even cloud-based email accounts such as Gmail.” Cidon, ¶99. Cidon’s management server was “connected to consumer-based file repositories (e.g., Dropbox 13, Gmail 11, or others such as Evernote) through Application Interfaces (API) 101 and 103 or through web access, similar to the way that a browser interact[ed] with a web server, accessing stored files through interface[d] such as FTP, Secured FTP (SFTP) or WebDAV.” (Cidon, ¶100; *see also* Cidon, ¶99 (describing that its disclosed system supported cloud storage services such as “Dropbox, Box, Sugarsync or a web-based storage system such the one offered by Amazon S3 or EMC” and “Google Docs, Office 365, Evernote and even cloud-based email accounts such as Gmail.”).)

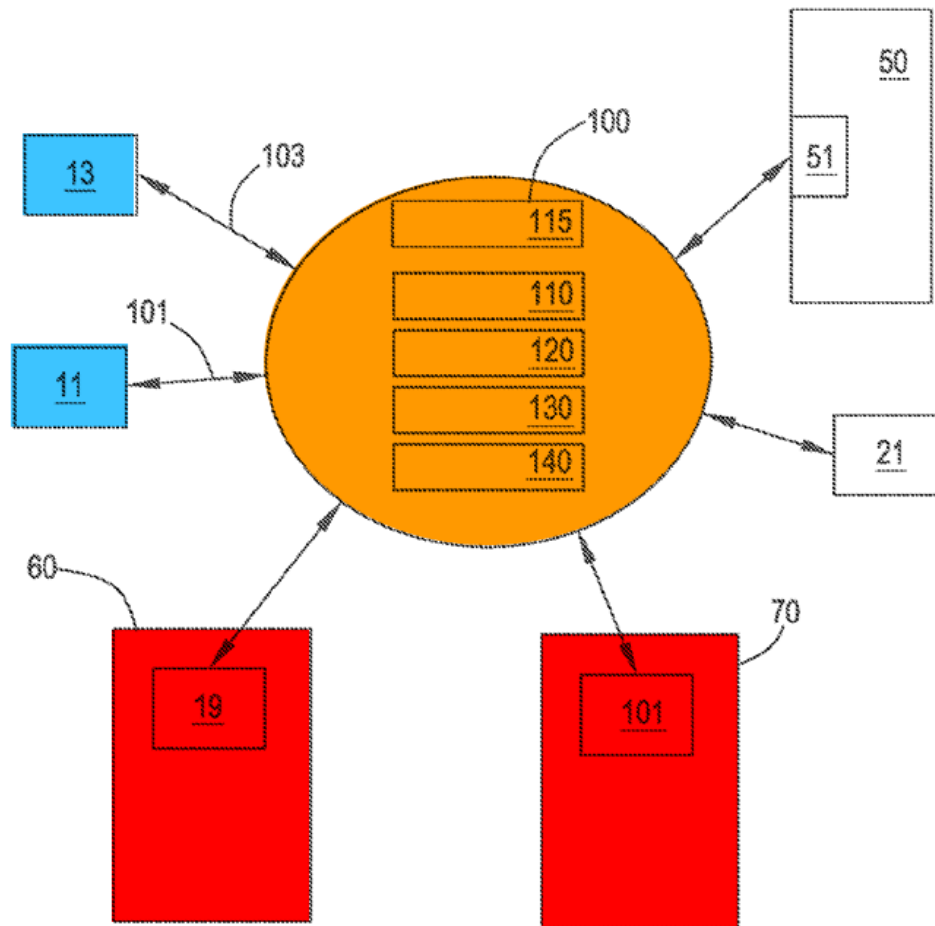


FIG. 2 (Annotated)

74. **Assigning the Policy**: While Cidon disclosed defining an access policy for a user (e.g., ¶108), providing access policy configuration, and enforcing the access policy, it did not explicitly disclose “assigning, by a gateway device, a policy.” Herrmann, however, disclosed or rendered obvious this limitation.

75. Specifically, Cidon described its management server was responsible for, *inter alia*, provid[ing] access policy configuration for secure file access...[and]...enforce[ing] access policy for secure file access.” (Cidon, ¶¶102,

108.)

76. Cidon's policy also "dictate[d]" whether certain files were to be encrypted (policy for processing a file). (Cidon, ¶¶224, 227.)

77. Cidon further recognized that "[d]ifferent IT departments [would] want different policies for access to these secure files" and "[u]sers in a company may be organized in further groups and each group may have different policies." (Cidon, ¶108.) To achieve these goals, Cidon's IT Policy Enforcement Module 120 (part of the management server) "allow[ed] for an administrator to define access policies...through a web browser interface to the module." (Cidon, ¶108.)

78. Herrmann disclosed a **gateway server 350** (gateway device) that "act[ed] as a gateway (e.g., a VPN gateway) to a group of resources or collection of data (e.g., the **protected data or resources 390**) and accept[ed] requests for access to such resources from **client machines**." (Herrmann, ¶60, FIG. 3.)

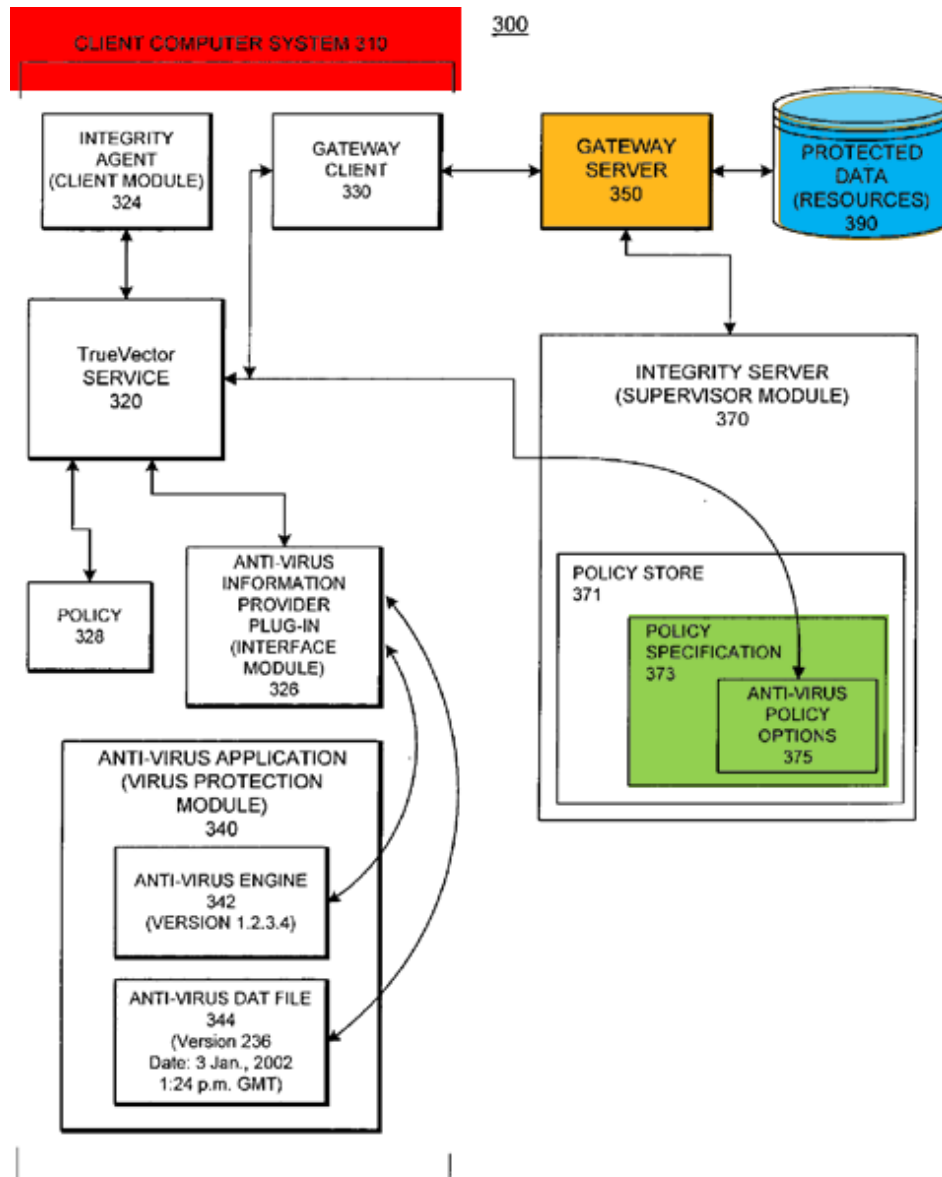
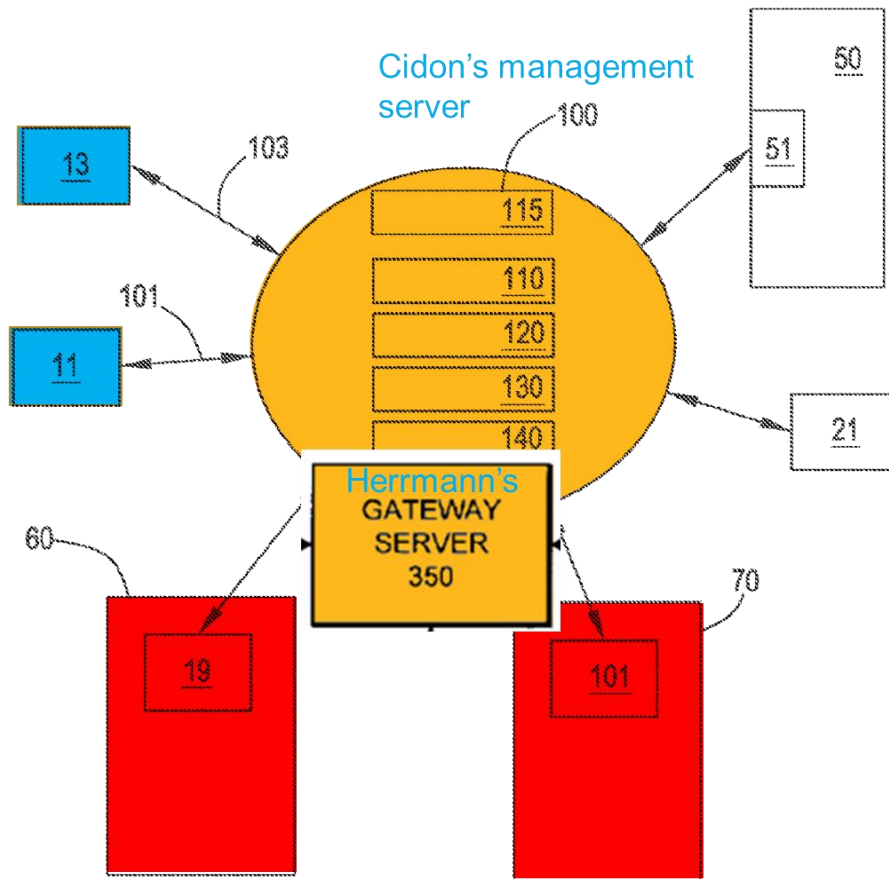


FIG. 3 (Annotated)

79. Herrmann’s “gateway server 350 perform[ed] various steps to verify that the client computer 310 [was] authorized to access the protected data 390.” (Herrmann, ¶60, FIG. 3.) Herrmann’s gateway server 350 also “assign[ed] an **access policy** to the connection/session between the gateway server 350 and the client computer 310).” (Herrmann, ¶77; *see also* Herrmann, claim 16 (“assigning

an access policy to said device, said access policy specifying protected resources that can be accessed by said device”).) Herrmann’s policy store 371 also stored the “appropriate **policy specification 373** applicable to the client computer 310” that could be retrieved “in a number of different ways, including...by **user**, by **group**, or by computer.” (Herrmann, ¶67.)

80. A POSITA would have been motivated by the teachings of Herrmann to modify Cidon’s system. Specifically, in the Cidon-Herrmann combination, a POSITA would have added Herrmann’s gateway server (gateway device) and its functionality of assigning the access policy to Cidon’s system (as illustrated in a modified Figure 2 of Cidon with Herrmann’s gateway server below) as part of the Cidon’s management server. Such a modification would have allowed the management server to “act[] as a gateway...to...the protected data” (e.g., data stored in Cidon’s enterprise file system 50 and consumer-based file repositories, such as Dropbox 13) and to perform the various functions disclosed in Cidon and Herrmann, including Herrmann’s functionalities for “assign[ing] an access policy” and “accept[ing] requests for access to such resources from client machines” (e.g., user devices 60, 70 in Cidon). (Cidon, FIG. 2; Herrmann, ¶¶77, 60.)



Modified FIG. 2

81. A POSITA would have been motivated to make this combination because instead of waiting for an administrator to manually define and assign a policy (as disclosed in Cidon), the modified system would have allowed automatic assignment and implementation of policies to users or groups of users, thereby improving system efficiencies. (Herrmann, claim 21 (“[U]pdated policy [was] automatically applied to a device.”).) Moreover, the modified system would have allowed better real-time protection by automatically applying (and assigning) an updated policy when such an update occurred. (*Id.*)

82. A POSITA would also have had reasonably expected success in making this modification because gateway devices had been well-known devices used for network security-related functionalities such as controlling network traffic of client devices and managing access policies. (See, e.g., Herrmann, ¶¶18, 77; Chambers, ¶¶66, 82 (describing network access devices, operated as gateway devices, that “identifie[d] and select[ed]” policies for user devices and “interface[ed] a LAN to WAN” for the user devices).) Additionally, Herrmann contemplated that its gateway server was sophisticated enough to implement other functions—such as those performed by its integrity server. (Herrmann, ¶¶65.) Accordingly, Herrmann’s gateway server could have been able to accommodate Cidon’s management server’s operations and would have yielded predictable results.

c. Claim 1[a][ii]: “wherein the policy defines access rights of the one or more users,”

83. Cidon disclosed that its policy defined access rights of the user to data stored in cloud storage. For example, Cidon’s management server was “responsible for encrypting files in the cloud and **enforcing secure access** to these cloud files by providing the right decryption keys to the right users,” including “provid[ing] access policy configuration for secure file access...[and]...enforc[ing] access policy for **secure file access**.” (Cidon, ¶102; see also *id.*, ¶108 (explaining that different users had different policies to access “file repositories[,] e.g.,[] Dropbox”), ¶¶112,

115.) As I explained above for Element 1[a][i] (§ I.A.3.b), Herrmann also corroborated the teachings of Cidon by explaining that its policy could be assigned “by user or by group.” (Herrmann, ¶67.)

d. Claim 1[b]: “encrypting, by the gateway device, using cryptographic key information defined by the policy, content of the file to produce a searchable encrypted file by:”

84. As I explained above in connection with Element 1[a][i] (§ I.A.3.b), a POSITA would have found it obvious and would have had a reasonable expectation of success in combining Herrmann’s gateway server with Cidon’s management server, such that the claimed operations would be carried out by the gateway. (See *supra* §I.A.3.b.)

“encrypting...content of the file to produce a searchable encrypted file”:

85. When combining Cidon with Herrmann, the gateway device included the functionality of Cidon’s management server that encrypted the content of the file to produce a searchable encrypted file.

86. Specifically, Cidon described that its management server was “responsible for encrypting files in the cloud.” (Cidon, ¶102; *see also id.*, ¶222 (“encrypting the file to provide an encrypted file”).) In an example in connection with Figure 13, the management server encrypted the content of **file 1410** to produce an **encrypted file 1460**. (Cidon, ¶¶303-305, 337 (explaining that the “segmenting, the calculating and the encrypting may be executed by a management

server”), FIG. 13.)

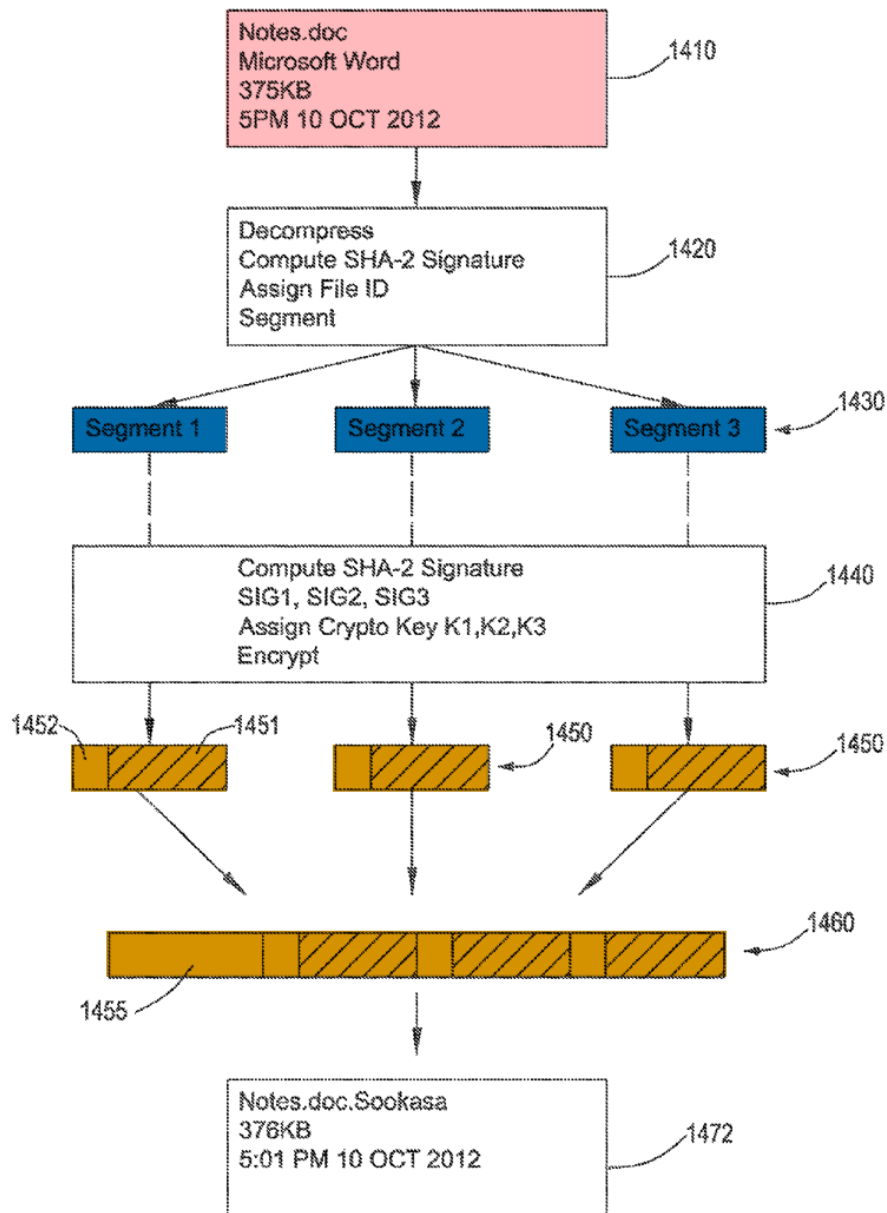


FIG. 13 (Annotated)

87. Cidon also described that its method included “encrypting each of the multiple file segments to provide multiple encrypted file segments ..., wherein the multiple encrypted file segments form an encrypted file 1640....” (Cidon, FIG. 15,

¶309.)

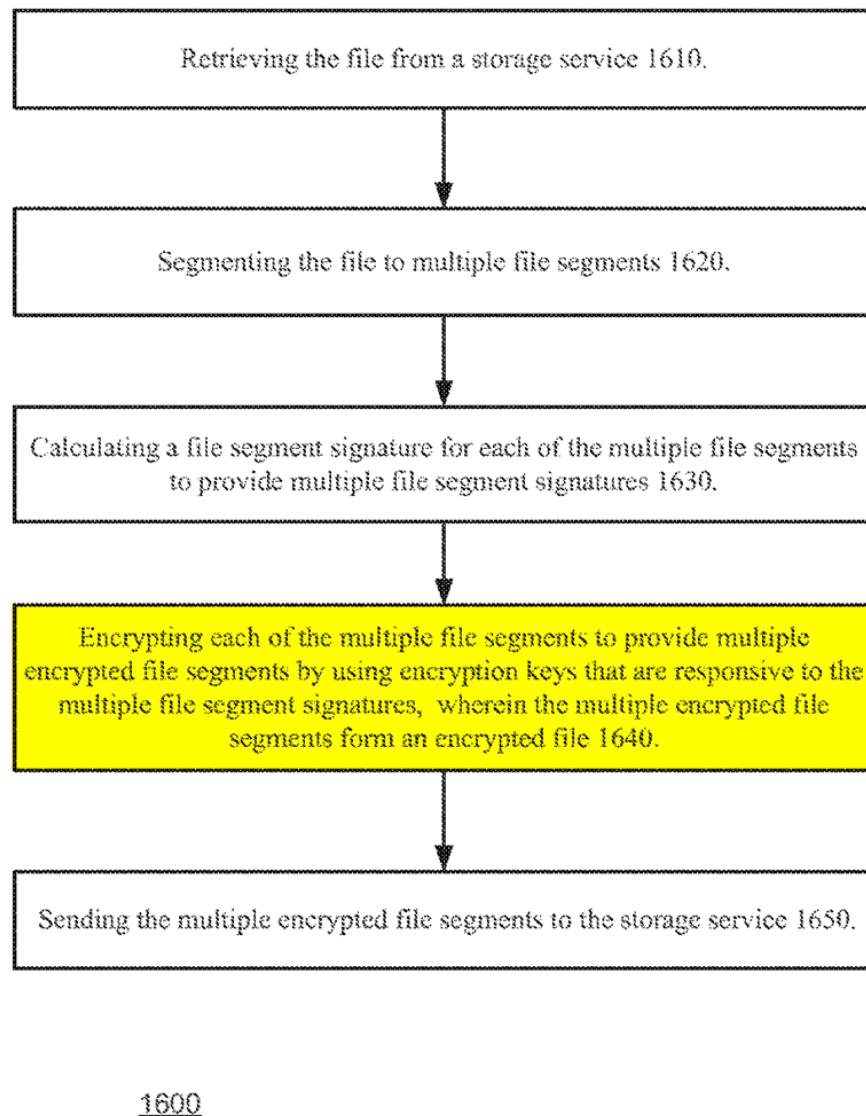


FIG. 15 (Annotated)

88. Moreover, Cidon taught or suggested that the encrypted files were searchable because Cidon's system allowed a user to "access[] the complete user file and folder structure by browsing the web service account using...an Application Interface (API)...[which] also allow[ed] the IT to program additional

functionalities such as **document search.**” (Cidon, ¶91.) Cidon also described that its management server extracted folder structure and file metadata from files uploaded to the management server for processing and these metadata “may help in **file search operation, using text/tags based search.**” (Cidon, ¶¶166, 171.)

“encrypting ... using cryptographic key information defined by the policy, content of the file”:

89. While the '678 Patent does not explicitly define the claimed “cryptographic key information,” it describes that “a selected policy can be used to define encryption keys, decryption keys, and encryption type, among **other attributes for carrying out the encryption** of data.” ('678 Patent, 4:4-7.) Consistent with this description, Cidon thought or suggested this limitation.

90. Specifically, Cidon’s policy specified whether or not to encrypt certain files. For example, Cidon described that its management server encrypted “a specific file was found to be under the encryption policy...[that] dictates that all descendent files must be encrypted too.” (Cidon, ¶¶224, 227; *see also* ¶232 (“If the policy dictates that file must be always encrypted, it will be re-encrypted after a time out as described before in the encryption functions.”).) Cidon also disclosed **using different encryption parameters** for encrypting another file that is associated with another user that differs from the specific user.” (Cidon, ¶324; *see also* ¶326.) A POSITA would have found it obvious that Cidon’s encryption parameters would include the “cryptographic key information,” such as an

encryption key because an encryption key was a common and well-known encryption parameter that was necessary for encrypting a file. (Compare to '678 Patent, 4:4-7, 10:1-3 (“[E]ncryption module 210 can encrypt each file and/or part thereof using cryptographic key information....”).)

91. A POSITA would have also found it obvious that Cidon’s policy would have defined encryption parameters (cryptographic key information) because Cidon’s policy already included encryption information (Cidon, ¶¶207-208, 212), Cidon described that different users/user groups were assigned different policies (Cidon, ¶108), and consistent with those descriptions, Cidon further explained that encryption parameters could differ based on the specific user or groups of users. (Cidon, ¶¶324, 326)

e. Claim 1[b][i]: “dividing the file into a plurality of chunks;”

92. Cidon disclosed this limitation because it described “segmenting the file into multiple file segments” (chunks). (Cidon, Abstract.) For example, as illustrated in Figure 13, Cidon disclosed dividing **file 1410** into three **file segments** 1430. (Cidon, FIG. 13, ¶305 (“The file 1410 is processed by...segmentation ...to provide three file segments (1430)....”).)

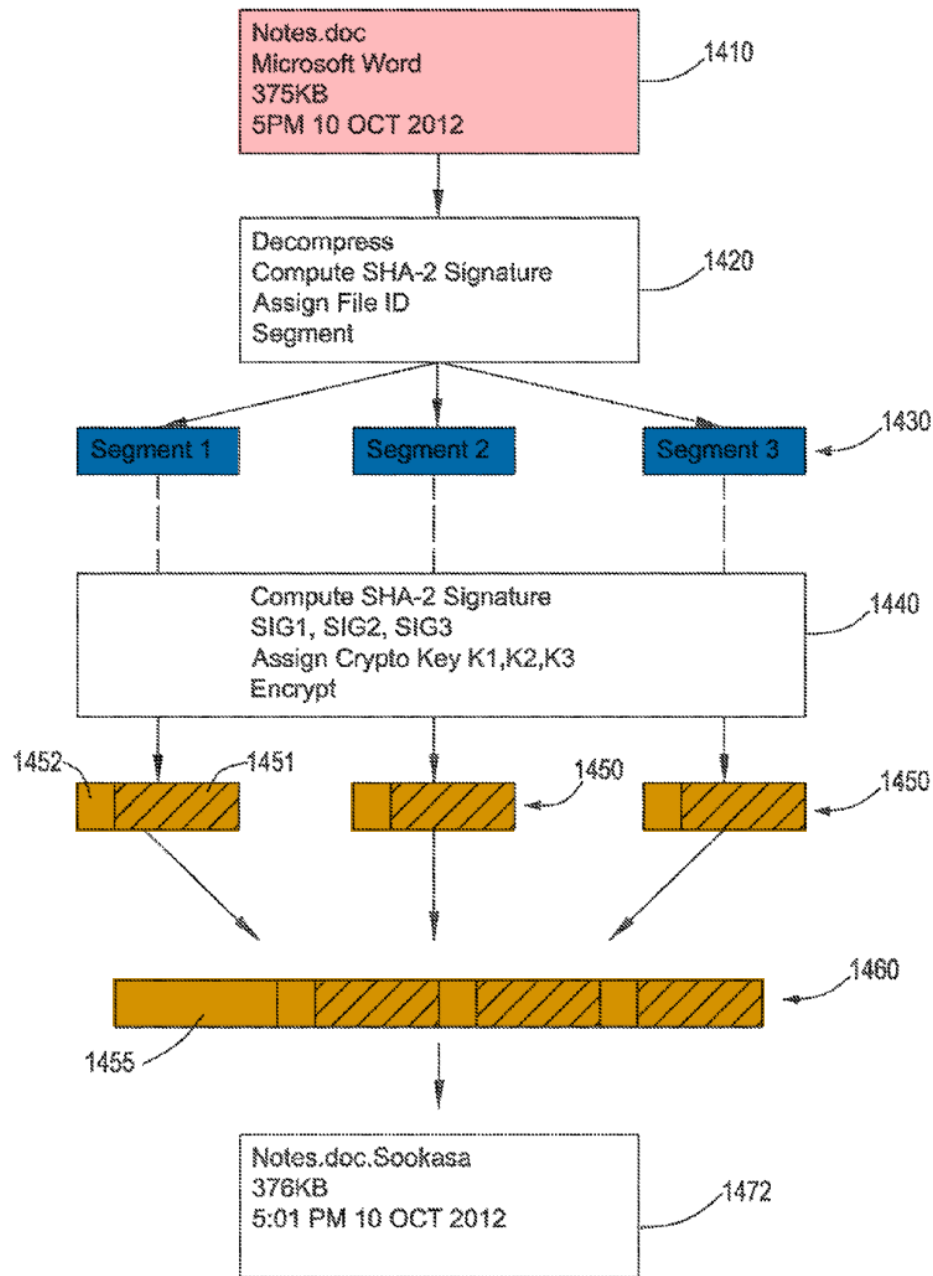
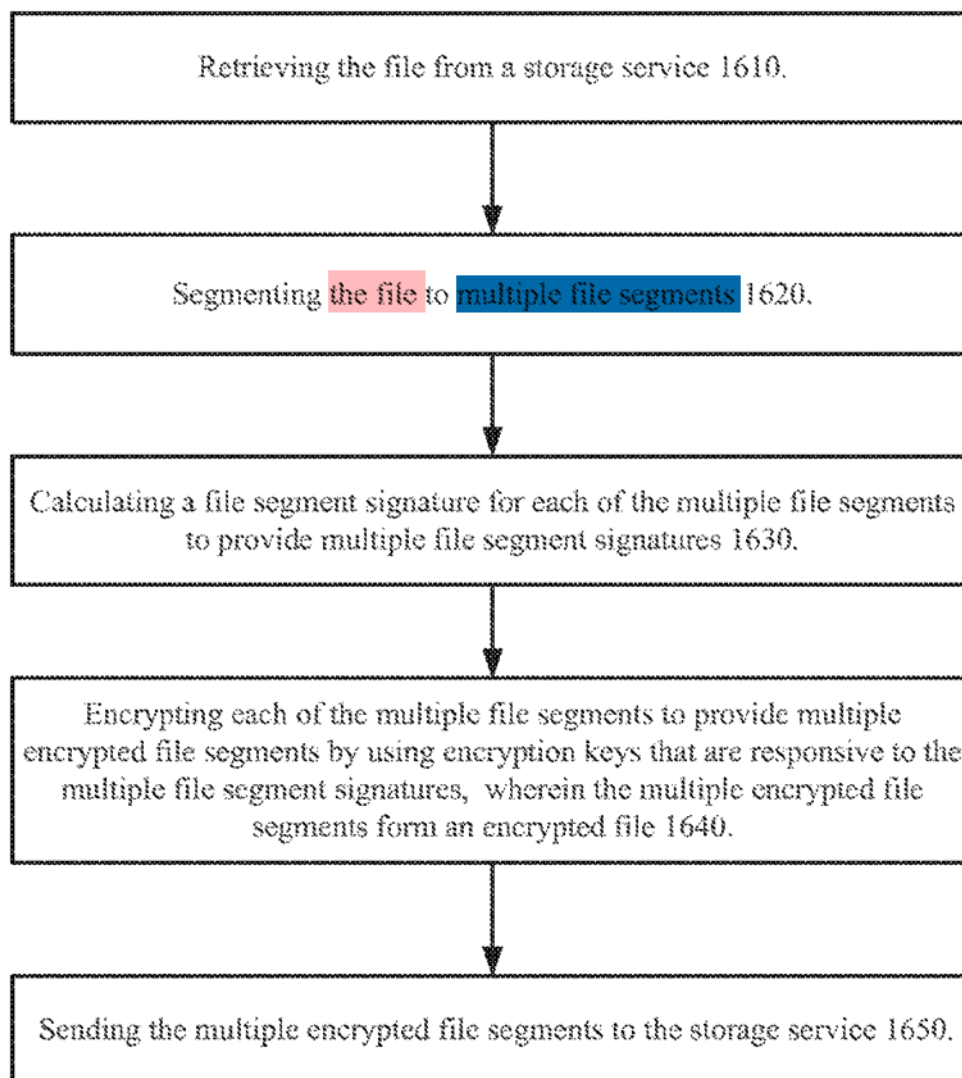


FIG. 13 (Annotated)

93. (See also Cidon, ¶35 (“[S]egmenting of the file by the file management server...”), FIG. 15, ¶309 (“Method 1600 may include:...segmenting the file into multiple file segments 1620....”).)



1600

FIG. 15 (Annotated)

f. 1[b][ii]: “creating namespaces for one or more of the plurality of chunks; and”

94. Cidon disclosed calculating (and creating) a “unique” file segment

identifier (or file segment signature) for each of **file segments** (chunks) based on the content of a file segment using, e.g., SHA2 signature computation algorithm. (Cidon, ¶¶305, 383, FIG. 13.)

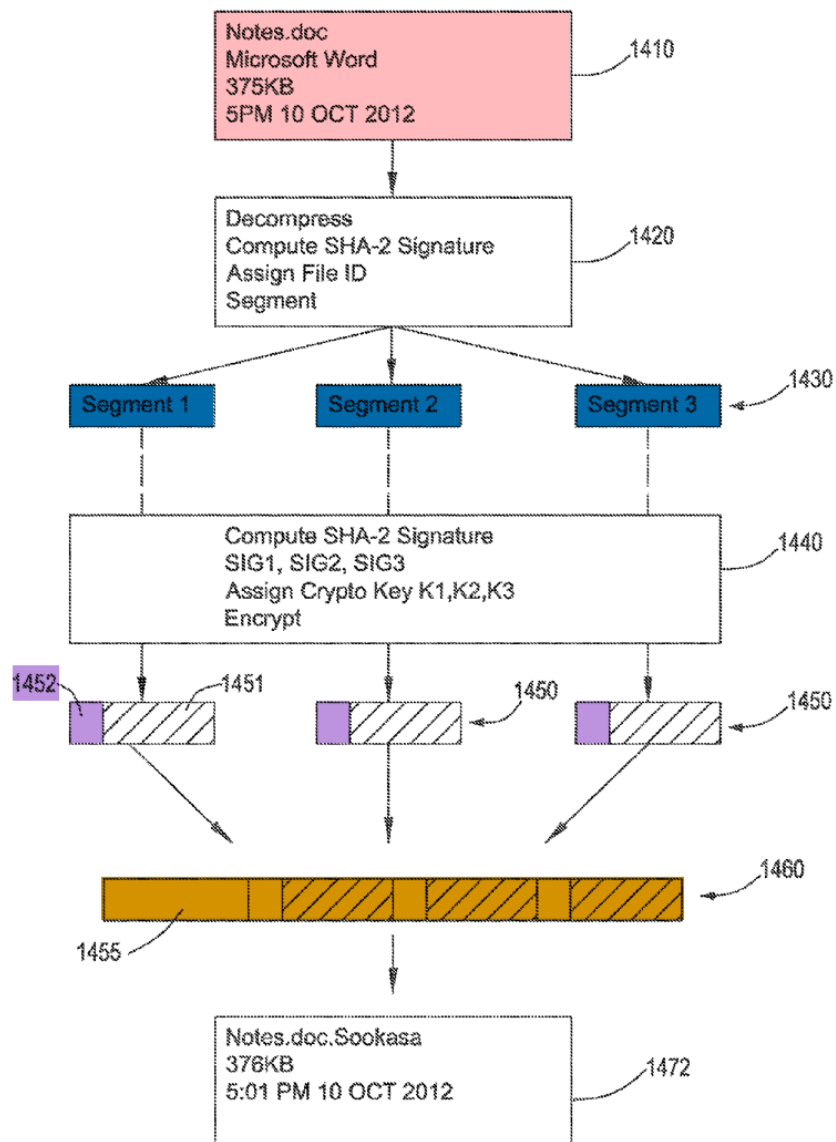


FIG. 13(Annotated)

95. (See also FIG. 15, ¶309 (“Method 1600 may include:...calculating a file segment signature for each of the file segments to provide multiple file

segment signatures 1630....”).)

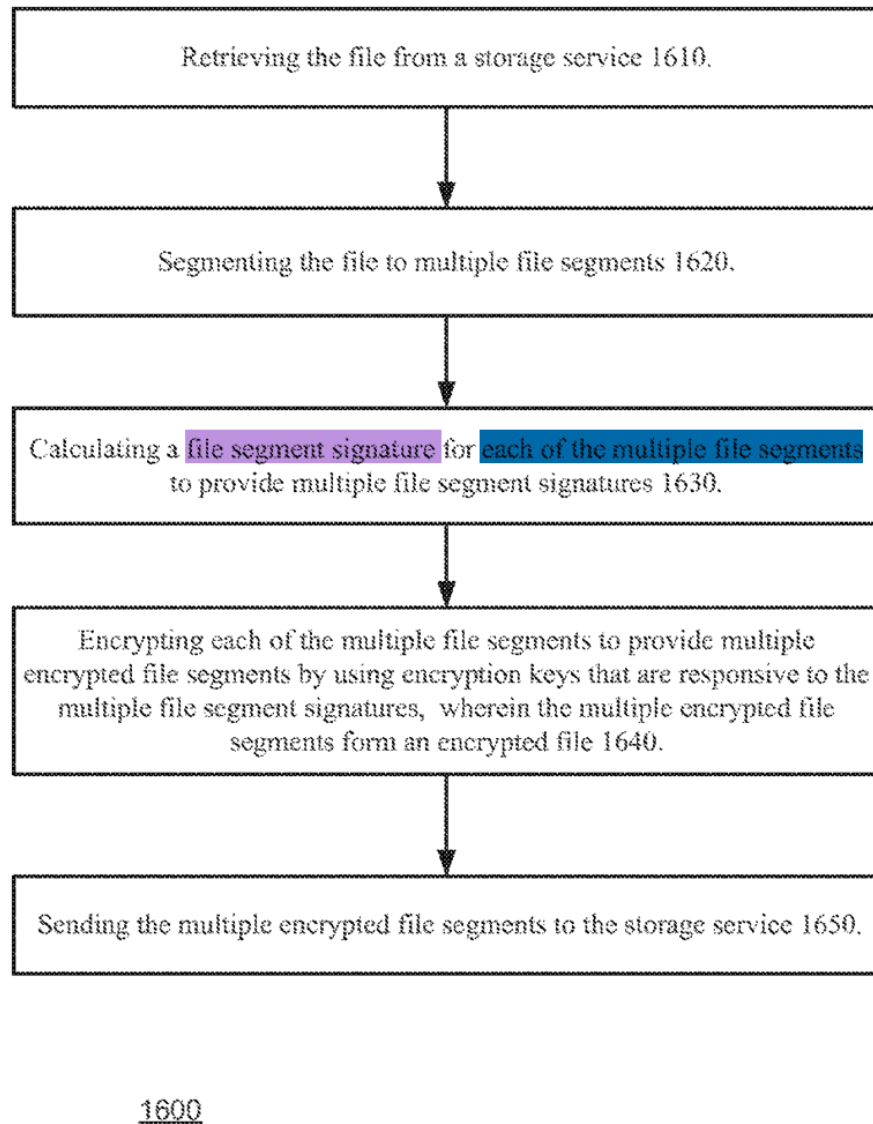


FIG. 15 (Annotated)

96. A POSITA would have understood that these segment identifiers, which were calculated using SHA2 signature computation algorithm, identified the corresponding segments. (Cidon, ¶106.)

97. Cidon also described assigning the calculated segment identifiers to the file segments. (Cidon, ¶305, FIG. 13 (illustrating that a segment identifier 1452 was assigned to each of the segments), ¶309, FIG. 15.)

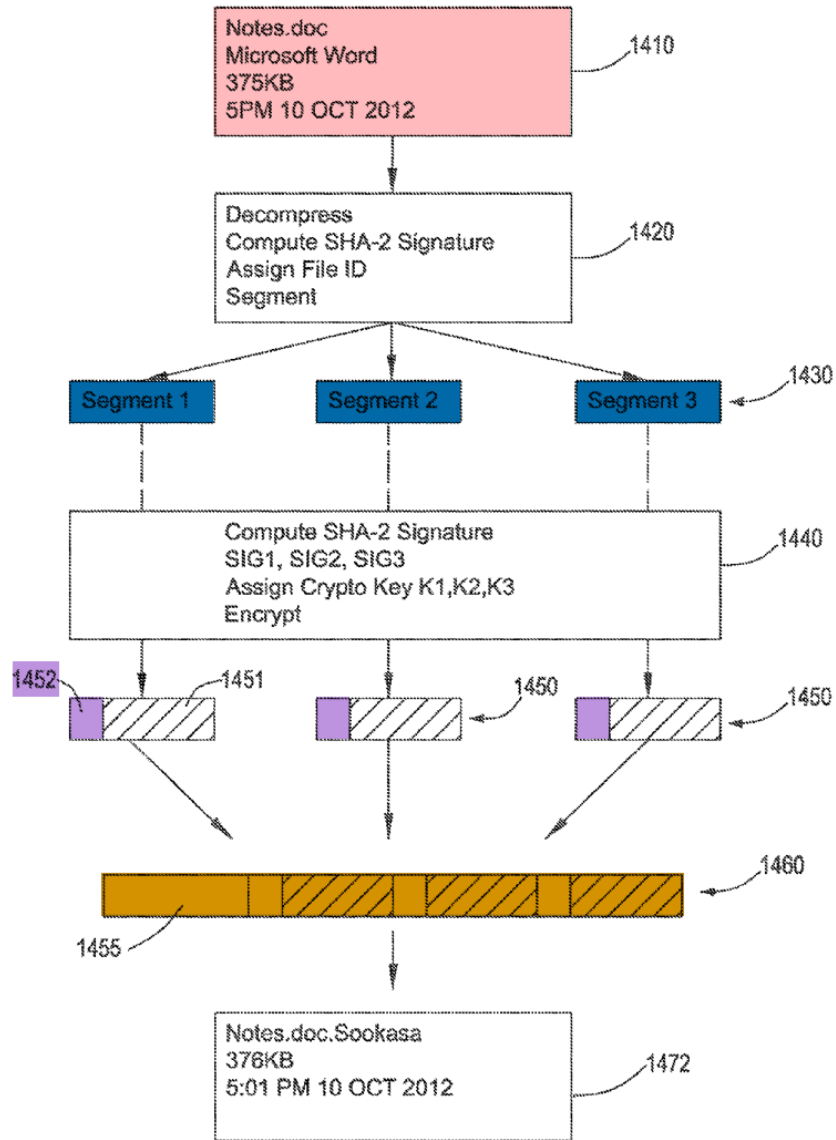


FIG. 13(Annotated)

98. Therefore, a POSITA would have found it obvious that Cidon's segment identifiers for the file segments constituted the claimed "namespaces"

because these identifiers were “the set of names available for naming”
corresponding file segments consistent with the plain and ordinary meaning of this
term. (Ex. 1014.)

- g. 1[b][iii]: “configuring the namespaces of the one or more chunks such that content of the file is encrypted in a manner that makes it searchable;”**

99. While Cidon taught or suggested encrypting file content to produce a searchable encrypted file (*see supra* § I.A.3.d) and producing segment identifiers, it did not explicitly disclose configuring segment identifiers (namespaces), so the file content was encrypted in a manner that made it searchable. Shikfa, however, taught or suggested this limitation by describing pointers/identifiers configured to allow the search and retrieval of encrypted files.

100. Specifically, Shikfa disclosed “generat[ing] one or more indexes 24 for one or more documents 26 that [were] to be encrypted as **encrypted documents 14** and stored on the **storage providers 18**.” (Shikfa, ¶29, FIG. 2.)

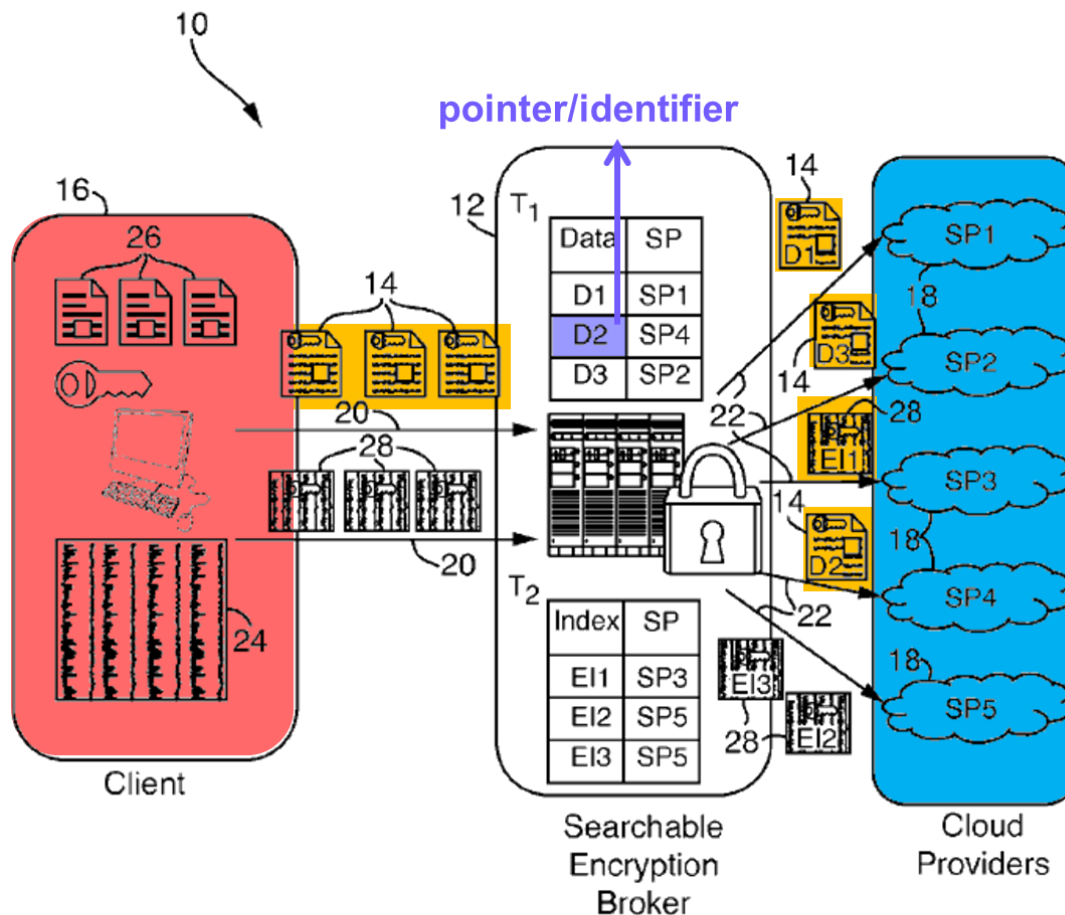


FIG. 2 (Annotated)

101. Shikfa's "index 24 include[d] one or more keywords for the encrypted documents 14 and **pointers** to the encrypted documents 14 containing the keywords." (Shikfa, ¶29.) Indexes 24 were then encrypted using a searchable encryption algorithm to produce encrypted indexes 28. (Shikfa, ¶29.) It would have been obvious to a POSITA that the **pointers** in paragraph 29 of Shikfa were the same as "**identifiers**" (e.g., **D1**, **D2**, **D3**) that were described in later sections of Shikfa. Notably, the pointers/identifiers both identified the encrypted documents

(Shikfa, ¶¶29, 31), and a search of the encrypted indexes 28 “return[ed] identifiers (e.g. D1, D2, D3, etc.) of the one or more encrypted documents 14.” (Shikfa, ¶36.) A POSITA would have also found it obvious that Shikfa’s pointers/identifiers were namespaces because they allowed identification of encrypted documents, consistent with the plain and ordinary meaning of this term: “the set of names available for naming things,” such as documents. (Ex. 1014.)

102. Referring back to Figure 2, Shikfa described the pointers/identifiers (e.g., **D2**) were stored in a translation table T1. (Shikfa, ¶31, FIG. 2.) These identifiers were configured to allow the encrypted documents to be searched, as shown in Figure 3 (below). Specifically, upon receiving an encrypted search query 30 from the client, the broker server 12 used encrypted indexes 28 to conduct the search, which “return[ed] **identifiers** (e.g. [,] **D1, D2, D3**, etc.)...that satisfy[ied the] encrypted query 30” (Shikfa, ¶36, FIG. 2.) Translation table (T1) was then used to “look up the **identifier(s)** returned...(e.g.[,] **D2**) and then [to] retrieve[] the corresponding encrypted document(s) 14 from the storage providers 18.” (Shikfa, ¶36, FIG. 3.)

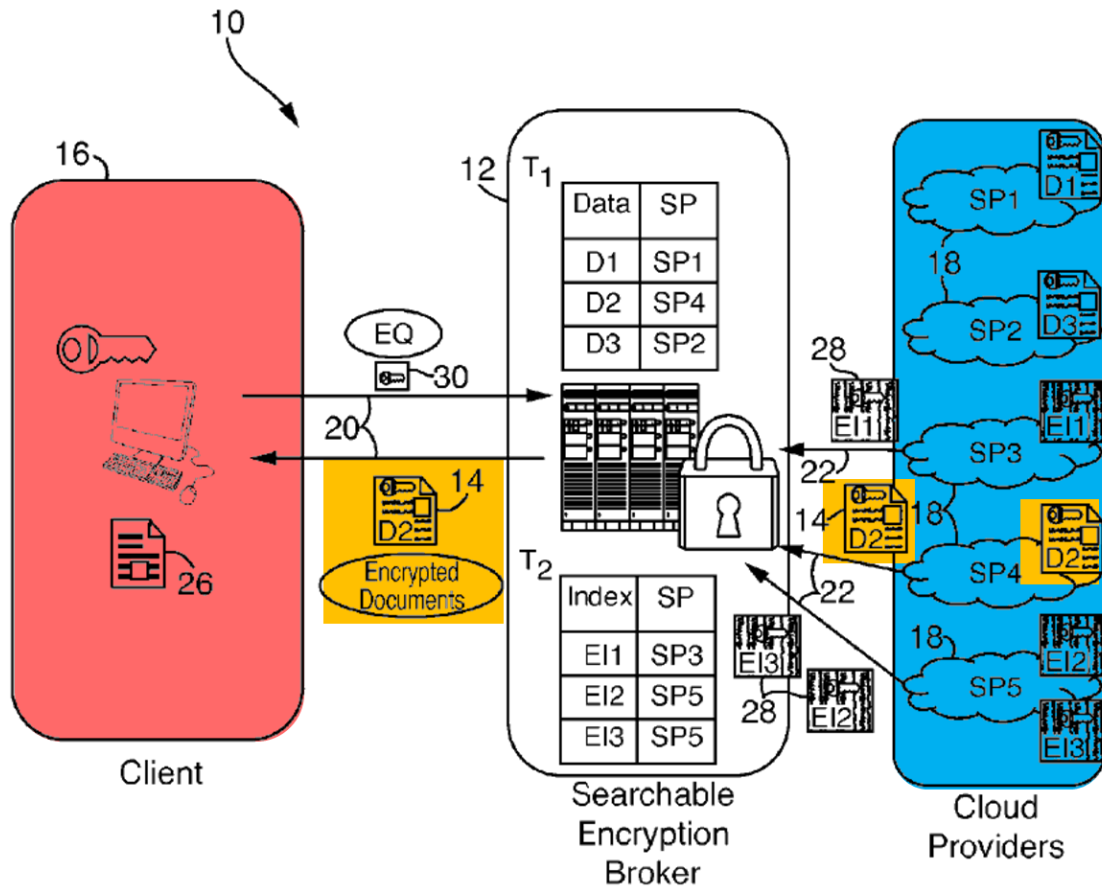


FIG. 3 (Annotated)

103. A POSITA would have understood that the operations in Figures 2 and 3 of Shikfa described configuring identifiers (namespaces) because they constituted a mapping among the identifiers, the encrypted documents, and the search keywords, and this mapping allowed the encrypted documents to be searched and identified.

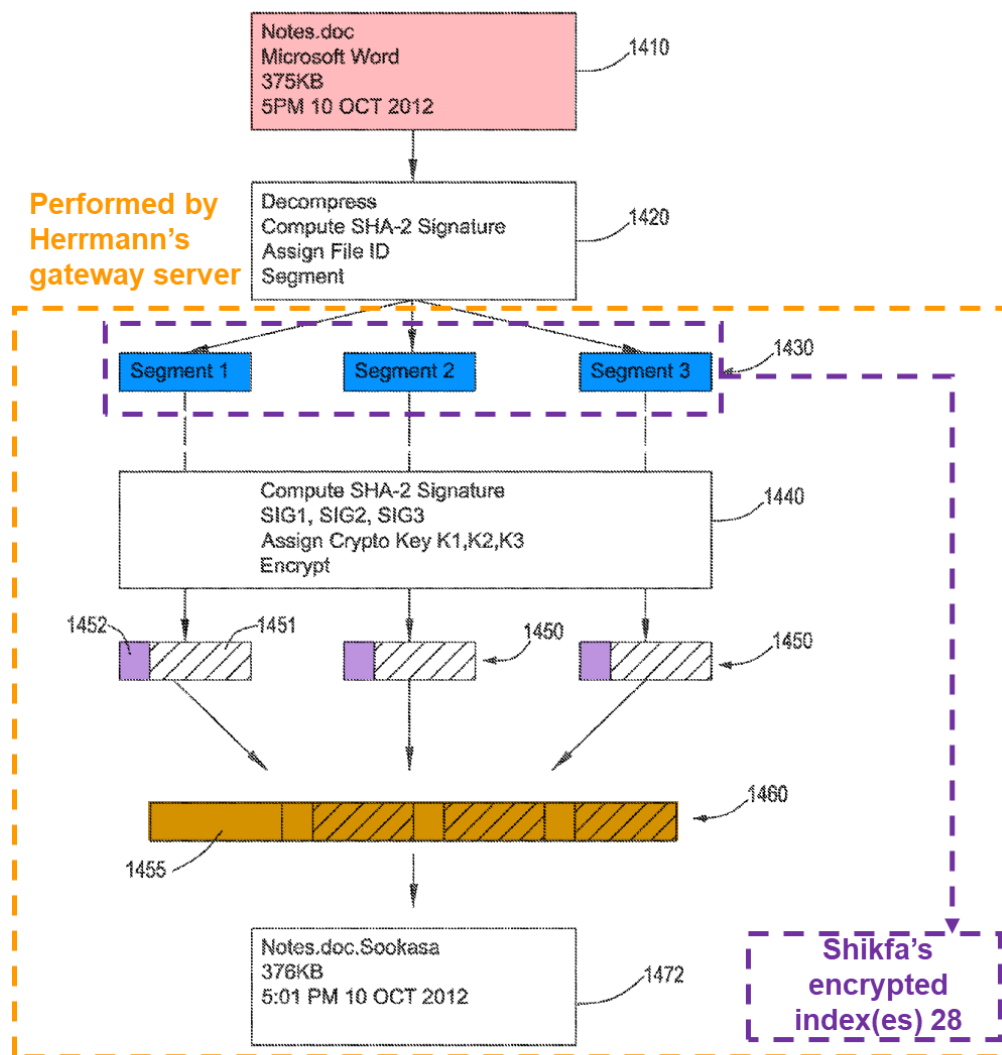
Combination of Shikfa with Cidon and Herrmann:

104. While Shikfa primarily dealt with searching and retrieving full documents, it also explained that a “**part of a particular encrypted document**” may

be stored to cloud storage providers to “guarantee availability of the stored data.” (Shikfa, ¶39.) A POSITA would have been motivated by the above teachings of Shikfa to combine Shikfa’s searchable encryption mechanism with Cidon’s encryption technique that divided the content into segments (chunks) and created a segment identifier for each chunk to enable search of encrypted chunks.

105. In doing so, the modified system would have allowed content of the file to be encrypted and searched “without requiring any such capability” at a storage provider, and thus would have provided more cloud storage options (e.g., allowing files stored on cloud storages that did not have any search capabilities while maintaining the searchability). (Shikfa, ¶50.) Additionally, the modified system would have “accommodate[d] document searchability” without “compromis[ing the]...privacy needs,” thereby preventing confidential information from being revealed to a service provider. (Shikfa, ¶50.)

106. In the combination, the **identifiers** (e.g. [,] **D1, D2, D3**) of Shikfa, which identified the full document, would have been replaced with segment identifiers of Cidon, and implemented in the gateway server as described in Herrmann (illustrative diagram of a modified Figure 13 of Cidon is shown below).



Modified FIG. 13

107. A POSITA would have reasonably expected success in combining Cidon with Shikfa's searchable encryption mechanism (and Herrmann's gateway server) because searchable encryption mechanisms had been well-known and would have been compatible with Cidon and Herrmann's systems, and the combination would have yielded expected results. (See Chiueh, Exs. 1015-1016.)

- h. 1[c]: “storing, by the gateway device, the searchable encrypted file on the one or more**

cloud platforms based on the policy; and”

108. In the Cidon-Herrmann-Shifka combination, the gateway device included the functionality of Cidon’s management server would store the encrypted files on a cloud platform based on the policy. Specifically, Cidon disclosed storing the searchable encrypted file on one or more cloud storages (cloud platforms) by sending the file to these cloud storages. For example, Cidon’s management server was “connected to **consumer-based file repositories** (e.g., Dropbox 13, Gmail 11, or others such as Evernote) through Application Interfaces (API) 101 and 103 or through web access.” (Cidon, ¶¶99-100, FIG. 2.)

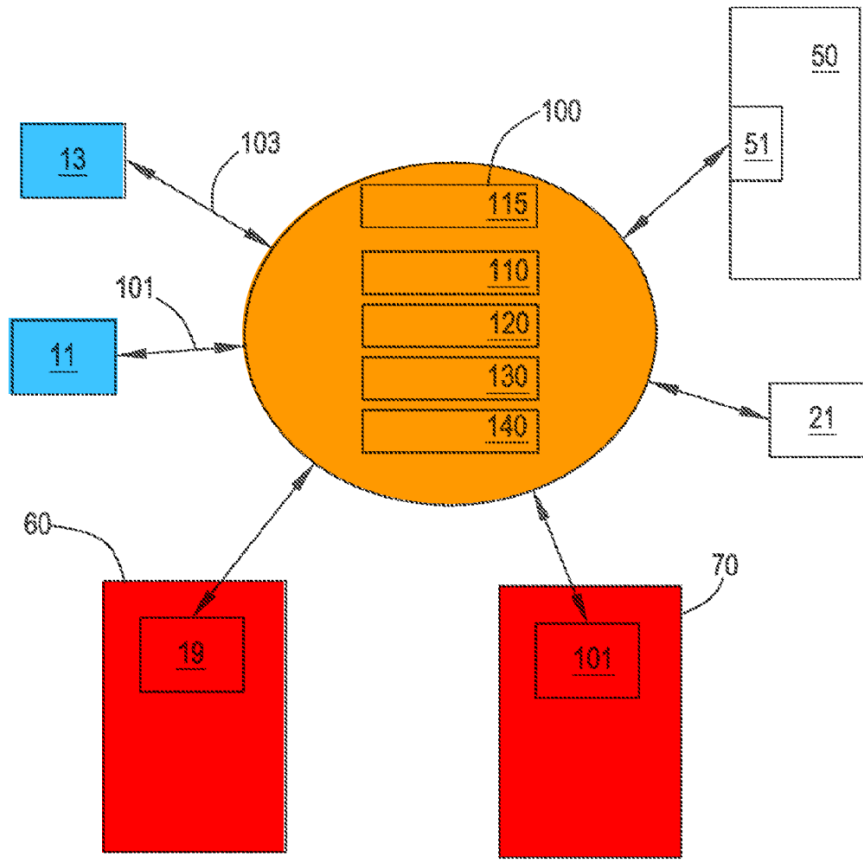
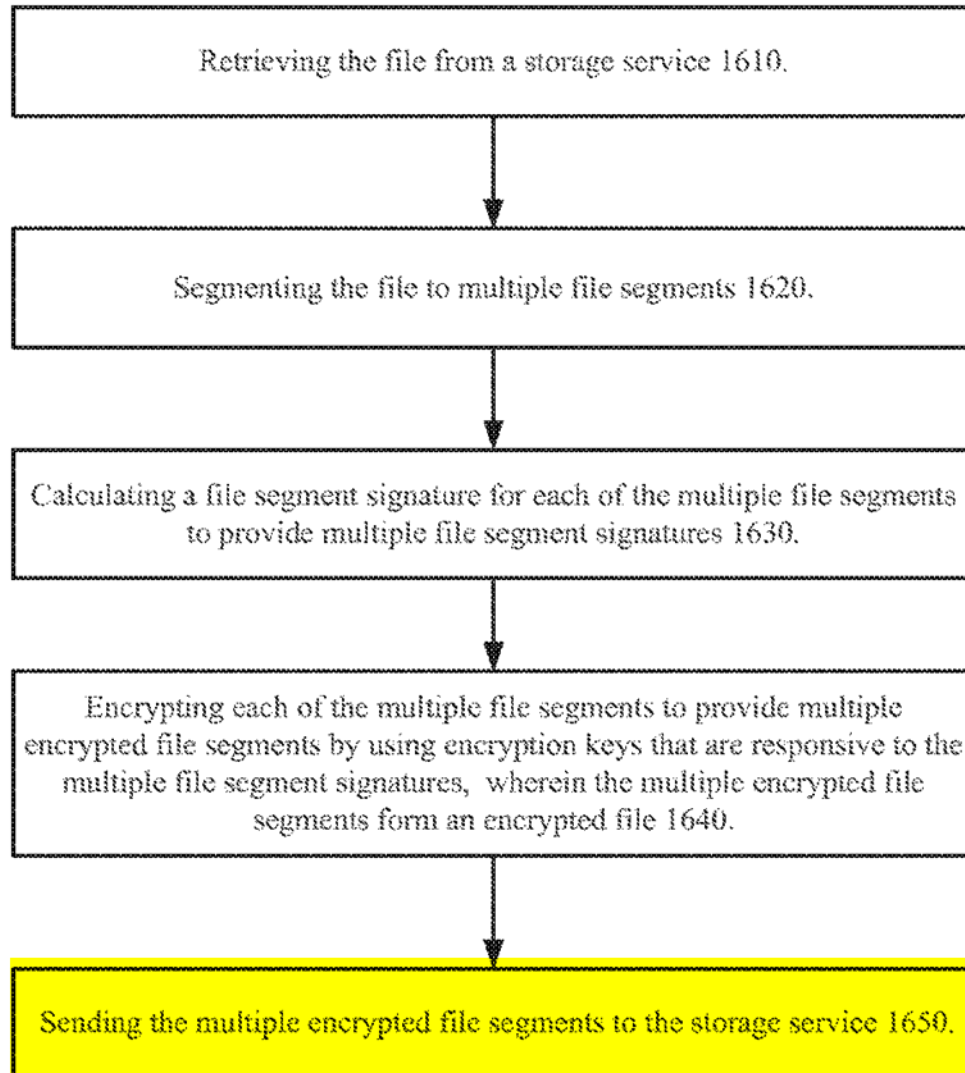


FIG. 2 (Annotated)

109. Cidon's management server sent encrypted file segments (which formed an encrypted file) to the cloud storage. (Cidon, ¶309, FIG. 15; *see also id.*, Abstract, ¶¶10, 43, 229, claims 1.)



1600

FIG. 15 (Annotated)

110. A POSITA would have further understood that storing the searchable encrypted file on a cloud storage was based on the policy because Cidon's described that its policies included "[p]lacement [of files] in specific directories or

shares” (Cidon, ¶¶208-209), and placement policy violations could have resulted in moving the file to reverse the violation (Cidon, ¶¶214-215).

i. 1[d]: “managing access to the searchable encrypted file by the one or more users based on the policy.”

111. Cidon contemplated that different users would have different policies to access the files stored on cloud storages, such as Dropbox. (Cidon, ¶108.) Cidon’s management server was responsible for “enforcing secure access to these cloud files,” including “provid[ing] access policy configuration for secure file access” and “enforc[ing] access policy for secure file access.” (Cidon, ¶102.) For example, the management server “provide[d] user management, API access (as well as API feed) to cloud storage services such as Dropbox and Box.” (Cidon, ¶113.) Also, the management server’s IT Policy Enforcement Module 120 provided “an admin interface [that] allow[ed] for an administrator to define access policies...through a web browser interface to the module,” and “[o]nce this policy [was] defined it [would] be enforced through application interfaces (API).” (Cidon, ¶108; Herrmann, ¶67 (described assigning policies by “user [or] group”).)

112. A POSITA would have understood that Cidon’s management server managed access to the searchable encrypted file by the users based on the policy by, e.g., “provid[ing] access policy configuration for secure file access” and “enforc[ing] access policy for secure file access.” (Cidon, ¶102.)

4. Claim 16

113. In my opinion, Cidon, Herrmann, and Shifka rendered obvious claim 16.

114. Claim 16 does not require (or prohibit) that the recited functions or modules be performed by a single gateway device. As I explained above for claim 1, as an example, a single gateway device (e.g., Herrmann's gateway device) could be used to perform similarly recited functions. (*See supra* § **Error! Reference source not found.**)

a. Claim 16[pre]: “A secure cloud storage system comprising:”

115. Cidon disclosed a **management server** (secure cloud storage system) for “encrypting a file,” including “segmenting the file into multiple file segments;...encrypting each of the multiple file segments to provide multiple encrypted file segments...;...and sending the multiple encrypted file segments to the **storage service**.” (Cidon, FIG. 2, ¶57; *see also* Cidon, ¶¶99-100; *supra* § I.A.3.a.)

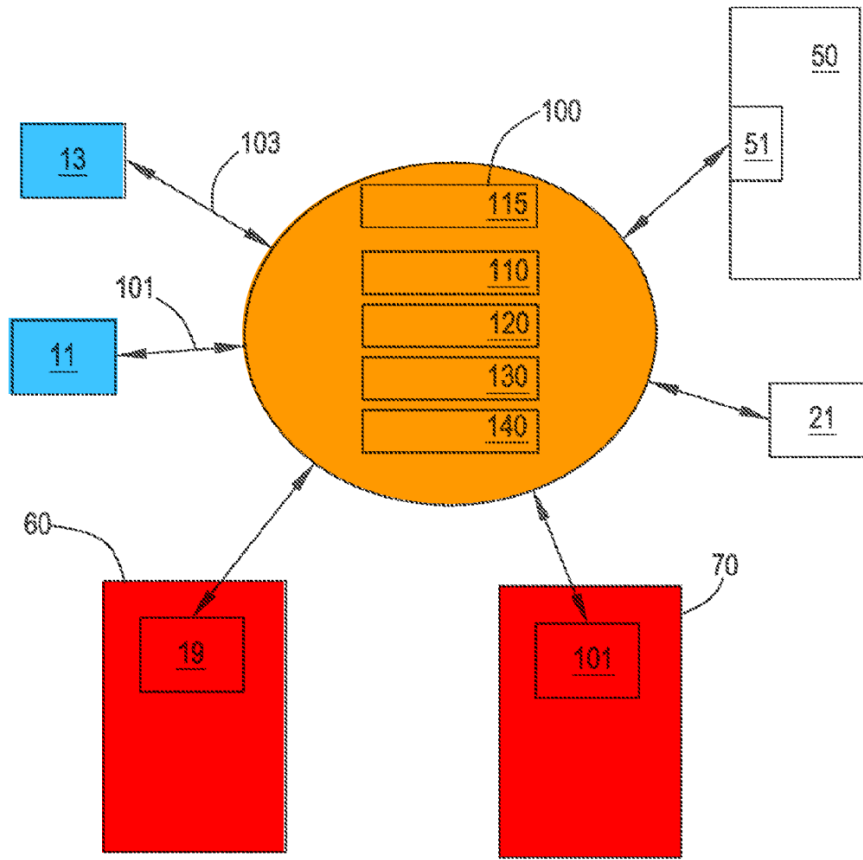


FIG. 2 (Annotated)

b. Claim 16[a]: “one or more processors;”

116. Cidon described that its invention “may also be implemented in a computer program for running on a computer system,” like management server 100, and such computer system included one “processing unit” (processor) and associated memory to execute software instructions to perform the functions described therein. (Cidon, ¶¶307, 407, 416.)

c. Claim 16[b]: “a communication interface mechanism;”

117. Cidon disclosed that its management server “communicate[d] with

user devices 60 and 70,” e.g., receiving requests for decrypting files or sending encryption keys to user devices. (Cidon, ¶¶101, 340, 342.) Cidon also described that the management server was connected to “file repositories (e.g., Dropbox 13, Gmail 11, or others such as Evernote) through Application Interfaces (API) 101 and 103 or through web access, similar to the way that a browser interacts with a web server, accessing stored files through interfaces such as FTP, Secured FTP (SFTP) or WebDAV” and sent encrypted file segments to cloud storage service. (Cidon, ¶¶100, 309.)

118. At least from the above teachings of Cidon, a POSITA would have understood that Cidon’s management server would include a communication interface mechanism for communicating with user devices and cloud storage services described in Cidon.

d. Claim 16[c]: “one or more internal data storage devices operatively coupled to the one or more processors and storing:”

119. As I explained above, Cidon disclosed that a computer system, such as management server 100, included memory (internal data storage) associated with (operatively coupled to) a processing unit (processor) that stored software instructions or computer program that, when executed by the processing unit, caused the computer system to “execute any or all of the methods described” therein, including various modules to carry out certain functions. (Cidon, ¶¶50,

101-102, 307, 407, 412, 416.)

- e. **Claim 16[c][i]: “a policy assignment module, which, when executed by the one or more processors, implements a policy selected from a group of policies for a user,”**

120. Cidon’s computer system, such as management server 100, included a processing unit (processor) and, when executing software instructions or computer program, “execute[d] any or all of the methods described” therein. (Cidon, ¶50.) Such computer programs included an IT policy enforcement module that provided “an admin interface allow[ing] for an administrator to define access *policies*” for users and implemented these policies. (Cidon, ¶¶101, 108.)

121. Cidon described that different users/user groups were assigned different policies. (Cidon, ¶108). Therefore, a POSITA would have found it obvious that Cidon’s IT policy enforcement module would select a policy for the appropriate user or user group to be implemented.

- f. **Claim 16[c][i][A]: “wherein the policy defines and manages access to and processing of a file to be uploaded on a cloud,”**

122. Cidon’s policy defined and managed access to and processing of a file to be uploaded on a cloud. (See § I.A.3.b (limitation 1[a][i]).)

- g. **Claim 16[c][i][B]: “wherein the policy defines access rights of the user;”**

123. Cidon’s policy defined access rights of the user. (See § I.A.3.c (limitation 1[a][ii]).)

h. Claim 16[c][ii]: “an encryption module, which, when executed by the one or more processors:”

124. Cidon described “any or all of the methods described above, including any stages—and any combinations” could be implemented using executable instruction or computer program. (Cidon, ¶50.) Therefore, a POSITA would have found it obvious that Cidon’s encryption would be performed by an encryption module using instructions that are execute by processor. (*See also* Cidon 102 (describing its management server was “regarded as a broad term to describe the complete software system which [was] responsible for encrypting files in the cloud and enforcing secure access to these cloud files by providing the right decryption keys to the right users” and “include[d] various software and/or hardware components” to carry out these functions.)

i. Claim 16[c][ii][A]: “using cryptographic key information defined by the policy, produc[ing] a searchable encrypted file by encrypting content of the file to be stored across one or more cloud platforms;”

125. As I explained above, Cidon disclosed “using cryptographic key information defined by the policy, produc[ing] a searchable encrypted file by encrypting content of the file.” (*See supra* I.A.3.d.) Cidon also disclosed that the encrypted content of the file was stored across more than one cloud platform because it described that its system “archiv[ed] and backup[ed a file] to other storage services, and [performed] file synchronization between services_g” (**two or**

more cloud platforms). (Cidon, ¶174.)

j. Claim 16[c][ii][B]: “divid[ing] the file into a plurality of chunks;”

126. Cidon disclosed this limitation. (*See supra* § I.A.3.e.)

k. Claim 16[c][ii][C]: “creat[ing] namespaces for one or more of the plurality of chunks; and”

127. Cidon disclosed this limitation. (*See supra* § I.A.3.f.)

l. Claim 16[c][ii][D]: “configure[ing] the namespaces of the one or more chunks such that content of the file is encrypted in a manner that makes it searchable;”

128. As I explained above, Cidon and Shikfa rendered this limitation obvious. (*See supra* §§ I.A.3.g, I.A.3.f.) A POSITA would have been motivated to add Shikfa’s searchable encryption mechanism (*see supra* I.A.3.g, I.A.3.f), including configuring the namespaces, into Cidon’s software or module so the modified module would have allowed content of the file to be encrypted and searchable “without requiring any such capability” at a storage provider. (Shikfa, ¶50.) Additionally, the modified system would have “accommodate[d] document searchability” without “compromis[ing the]...privacy needs.” (Shikfa, ¶50.)

129. A POSITA would have reasonably expected success in combining Cidon with Shikfa’s searchable encryption mechanism because searchable encryption mechanisms had been well-known and would have been compatible with Cidon’s module, and the combination would have yielded expected results.

(See Chiueh, Exs. 1015-1016.)

- m. Claim 16[c][iii]: “a storage module, which, when executed by the one or more processors, stores the searchable encrypted file on the one or more cloud platforms based on the policy; and”**

130. As I explained above, Cidon disclosed that its management server sent (and stored) the searchable encrypted file on the cloud storage (cloud platform) and a POSITA would have found it obvious that Cidon’s system sent and stored the searchable encrypted file based on the policy. (*See supra* § I.A.3.h.) As explained above, Cidon described “any or all of the methods described above, including any stages—and any combinations” can be implemented using executable instruction or computer program. (*See supra* §§ I.A.4.b, I.A.4.d; Cidon, ¶¶50, 102.)

- n. Claim 16[c][iv]: “a management module, which, when executed by the one or more processors manages and controls access to the searchable encrypted file by the user based on the policy.”**

131. As I explained above, Cidon disclosed that its management server managed access to the searchable encrypted file by the users based on the policy, (*see supra* § I.A.3.h), and implementing “any or all of the methods described above, including any stages—and any combinations” using executable instruction or or computer program. (*See supra* §§ I.A.4.b, I.A.4.d; Cidon, ¶¶50, 102.)

5. Claims 2/17: “...wherein the step of managing comprises allowing[/the management module further allows] the one or more users to download the searchable encrypted file to a local device”

132. Cidon disclosed claims 2/17 because its management server allowed the user to download the searchable encrypted file to a user device (local device) by “enabling user devices to connect to their web storage services” (cloud platforms) “through a standard web file access such as WebDAV, FTP and secure FTP supported on...[the] server...[,]...serv[ing] as a proxy to these file services.” (Cidon, ¶221; *see also id.*, ¶231 (“[T]he file may be...downloaded by some other application.”), ¶299 (“The app may...allow the download of the file.”).)

6. Claim 3: “wherein the local device comprises a mobile computing device.”

133. Cidon disclosed claim 3 because Cidon’s user devices (local devices) to which user could download encrypted files could be mobile computing devices, such as “mobile devices” (Cidon, ¶116), “iPhone/iPad or Windows [l]aptop” (*id.*, ¶¶108-109, 129). (*See also id.*, ¶95 (“When these files are accessed by the user they will be encrypted and read by the user using a decryption mechanism available on his device (PC, **laptop, notebook, tablet, smart phone** etc.”).)

7. Claim 8: “wherein the step of encrypting the file is carried out independent of the service provider of the one or more cloud platforms.”

134. In my opinion, Cidon disclosed claim 8 because, rather than relying on encryption services provided by the service provider of the one or more cloud

platforms, such as Dropbox, Box (Cidon, ¶99), *Cidon’s management server itself* “provide[d] file and folder encryption services to the user.” (Cidon, ¶¶113, 102 (the management server provided “encryption of...files”).) Indeed, Cidon’s management server allowed uploading **encrypted** files to different web storage services and its single connection “served **multiple** web storage instances from the same or **multiple vendors**.”) (Cidon, ¶221.)

8. Claims 9/23: “...wherein the policy is implemented for a group of users.”

135. In my opinion, Cidon disclosed claim 9/23 because it disclosed implanting and enforcing a policy for a group of users. For example, Cidon explained that “[d]ifferent IT **departments** [would] want different policies for access to these secure files” and “[u]sers in a company may be organized in further groups and **each group may have different policies**.” (Cidon, ¶108.) Cidon’s system also allowed an administrator to define these policies, which could be for a user group (Cidon, ¶108), and “[o]nce [a] policy was defined it [would] be enforced.” (Cidon, ¶108; *see also* Herrmann, ¶67 (describing that a policy could be assigned by “group”).)

9. Claims 10/24: “...wherein the policy defines the manner in which the file is stored, accessed, and processed.”

136. Cidon’s policy defined the manner in which the file was stored (e.g., “[p]lacement in specific directories or shares”), accessed (e.g., “[a]ccess and

sharing permissions,” “[c]opy control”), and processed (e.g., “[e]ncryption,” “[c]opy control”). (Cidon, ¶¶207-212; *see also* Cidon, ¶¶102, 108, 401-402, FIG. 23.)

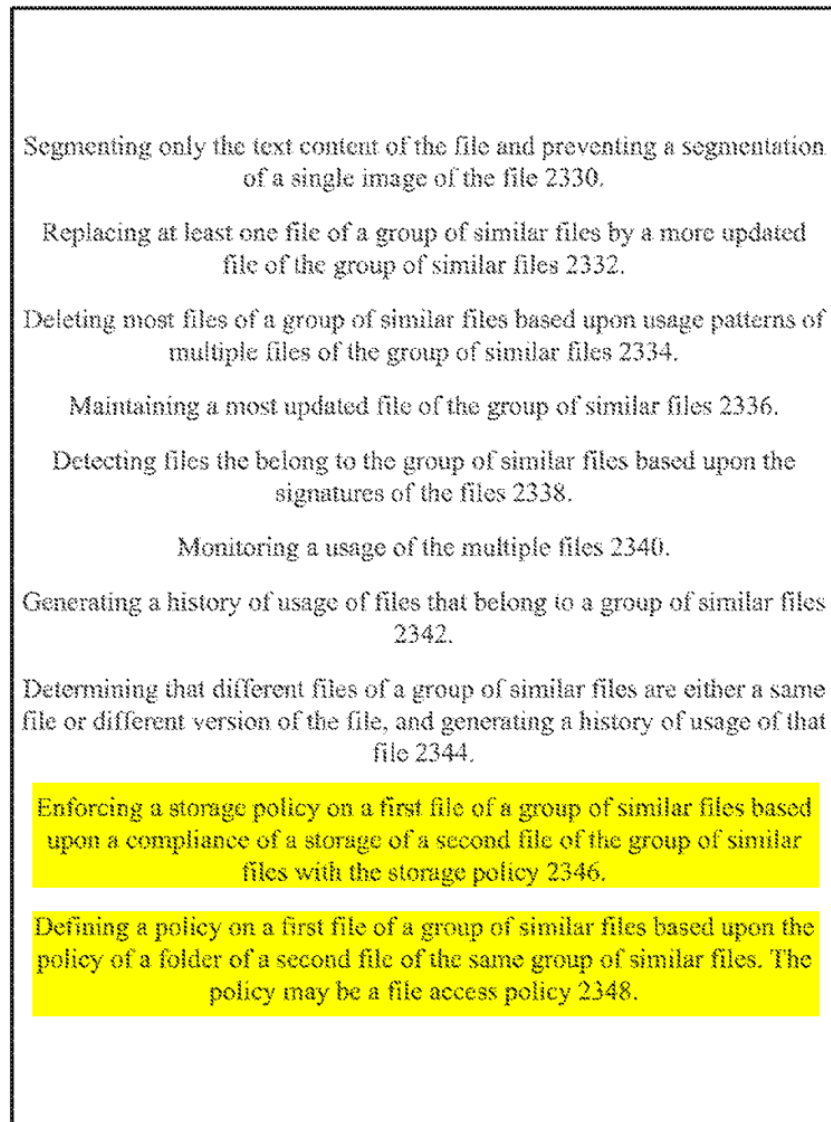


FIG. 23 (Annotated)

10. Claims 11/25: “...wherein the file is stored across two or more cloud platforms.”

137. In my opinion, Cidon disclosed claims 11 and 25 because it disclosed that its system performed operations on files that included “archiving and backup to other storage services_s, and file synchronization between services_s” (emphasis on plural, illustrating that the files could be stored in more than one cloud service or platform). (Cidon, ¶174.) A POSITA would have understood that Cidon’s backup and/or synchronization of the file on or between storage services constituted storing the file across two or more cloud platforms. (*See, e.g.*, Cidon, ¶221 (describing that Cidon’s system allowed uploading encrypted files to different web storage services).)

11. Claim 12: “...wherein said storing the searchable encrypted file on the one or more cloud platforms is carried out independent of the service provider of the one or more cloud platforms.”

138. In my opinion, Cidon rendered obvious claim 12 because, consistent with the ’678 Patent, Cidon’s searchable encrypted file stored on storage services (cloud platforms) could “be moved to any cloud platform at any moment, thereby avoiding vendor lock-in.” (’678 Patent, 13:54-59.)

139. Specifically, Cidon’s management server allowed “archiving and backup[ing the file] to other storage services_s, and file synchronization between services_s” (Cidon, ¶174), such as “consumer-based file repositories (e.g., Dropbox

13, Gmail 11, or...Evernote)” (Cidon, ¶100, FIG. 2), “BOX 24 service and in an enterprise file system 50” (Cidon, ¶173). Cidon’s management server also allowed storing encrypted files to different web storage services, and its single connection “served **multiple** web storage instances from the same or **multiple vendors**.” (Cidon, ¶221, FIG. 2.)

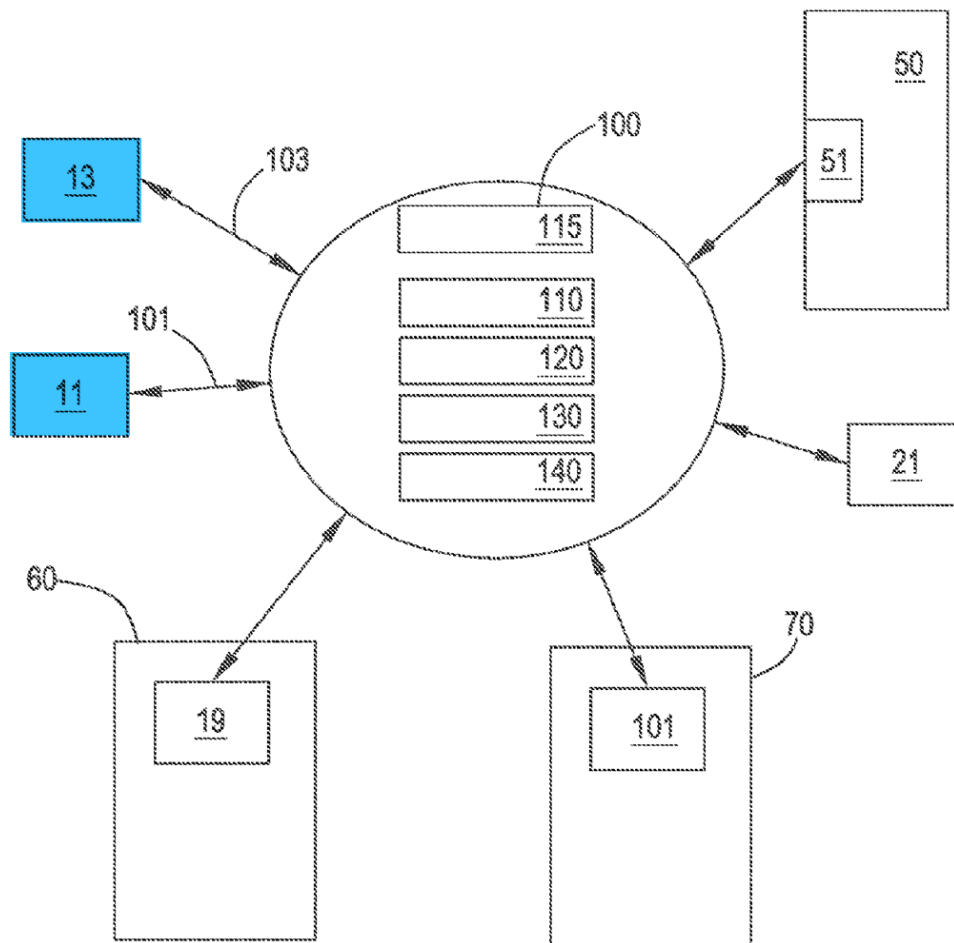


FIG. 2 (Annotated)

140. Further, as I explained above in connection with claim 8, the files processed using a searchable encryption that was independent of the storage

service provider. (*See supra* §I.A.7.) Therefore, a POSITA would have found it obvious that Cidon's technique was not "lock[ed]-in" ('678 Patent, 13:54-59) to a specific service, and Cidon's searchable encrypted file could "be moved to any cloud platform."

12. Claims 13/26: "...wherein the policy is selected from a plurality of policies stored in a database, wherein the database is operatively coupled with the gateway device."

141. In my opinion, Cidon and Herrmann disclosed these limitations.

142. Cidon explained that "[d]ifferent IT departments will want different policies for access to these secure files...[and u]sers in a company may be organized in further groups and each group may have different policies." (Cidon, ¶108.)

143. Cidon's IT Policy Enforcement Module (part of the management server) also provided an administrator with "an admin interface allow[ing] for an administrator to define access *policies*" for users and implemented these policies. (Cidon, ¶¶101, 108.) Cidon further described a "management web application" on user's devices communicated with the management server to "manage team *policies* and permissions, and to track team members' usage of the cloud storage services." (Cidon, ¶115.)

144. Herrmann similarly recognized the need for assigning different policies to different users and disclosed a policy store (database) for storing policy

specification. (Herrmann, ¶¶50, 67.)

145. While Cidon was not explicit about storing the policies in a “database,” a POSITA would have been motivated to store Cidon’s policies that were defined and managed by an administrator in a database coupled to the management server (with Herrmann’s gateway). The modified system would have enabled selecting a policy from the stored policies for the same user and applying the selected policy, instead of repeatedly creating a new policy, thereby improving system efficiency and policy implementation consistency.

146. A POSITA would have reasonably expected success in making this modification because Cidon already disclosed storing data, such as “file metadata, keys, access tables and user profiles,” “the user and team information,” in a database (Cidon, ¶¶103, 110, 119) and storing policies into a database had been well-known. (Chambers, claim 1 (“an access control policy (ACP) database to store ACPs”); Herrmann, ¶67 (“retriev[ing] the appropriate policy specification 373 applicable to the client computer 310 from the policy store 371”).)

13. Claims 14/27: “...wherein the policy defines [/and manages] one or more of a set of encryption keys, a set of decryption keys, search indices, a type of encryption and one or more authorized locations to which the file may be stored.”

147. In my opinion, the combination of Cidon, Shikfa, and Herrmann disclosed claims 14/27.

148. Specifically, the recited features would have been obvious features of a security policy, because Herrmann described that “[i]n general terms, a security policy [was] an organization's Statement defining the rules and practices that regulate[d] how it [would] provide Security, handle intrusions, and recover from damage caused by Security breaches. An explicit and well-defined Security policy include[d] a set of rules that [were] used to determine whether a given Subject [could] be permitted to gain access to a specific object.” (Herrmann, ¶24.)

the policy defines [/and manages] one or more of a set of encryption keys, a set of decryption keys...[and] a type of encryption:

149. More specifically, Cidon disclosed using a certain type of encryption (AES encryption) and encryption key (256-bit key) to encrypt a file. (Cidon, ¶110; Shikfa, ¶29 (describing encrypting documents and indexes using different encryption algorithms); Herrmann, ¶21 (describing two types of encryption—transport and tunnel modes).)

150. Cidon also disclosed generating, storing, and retrieving encryption keys. (Cidon, ¶¶102, 110.) Cidon’s encryption keys were used for decrypting the encrypted files and were, thus, also decryption keys. (Cidon, ¶43.) Cidon also explained that different users could have different policies and a policy was configured based on an administrator’s input. (Cidon, ¶108.)

151. A POSITA would have been motivated to have a policy defined (and managed) an encryption type, and encryption and decryption keys as taught or

suggested by Cidon. For example, when encrypting a file associated with a user, instead of waiting for the administrator's input, the modified system could have retrieved a corresponding policy to obtain the keys and encryption type defined in the policy that was previously configured by the administrator. (Cidon, ¶108.) Doing so would have improved system efficiency and ensured that the file was encrypted in a manner defined by the administrator. For example, a policy for a user who likely had more sensitive information (e.g., employees in the financial department) could have defined stronger encryption and decryption keys, and when encrypting a file associated with an employee from the financial department, the system would have encrypted the file using the encryption key(s) defined in the policy for the employee.

the policy defines [/and manages]...one or more authorized locations to which the file may be stored:

152. Cidon disclosed storing a modified file to a location other than the location that stored the original file to prevent the user from directly accessing the location that stored the original file. (Cidon, ¶¶364, 367.) A POSITA would have found it obvious that the locations where the file could and could not be accessed would be part of Cidon's security policy that could also define (and manage) such location information. To this end, when encrypting and storing a file (e.g., a modified file), Cidon's system would obtain the authorized location for storing the encrypted modified file from the policy, and accordingly store the encrypted

modified file at that authorized location. Doing so would have “prevented [the user] from directly accessing” other unauthorized storages, such as the location where the original file was stored, thereby improving data security. (Cidon, ¶367.)

the policy defines [/and manages]...search indices:

153. Shikfa disclosed generating a single encrypted search index or multiple encrypted indexes for an encrypted document. (Shikfa, ¶29.) Shikfa also described that the number of the encrypted index(es) generated depended on the specific searchable encryption mechanism that was used to encrypt the index. (Shikfa, ¶29.) In implementing the searchable encryption based on the combination of Cidon and Shikfa, a POSITA would have found it obvious to also define (and manage) whether to generate multiple indexes or one index for an encrypted document (as disclosed by Shikfa) in Cidon’s policy, so when encrypting files associated with the user, the modified system would have determined the number of encrypted index(es) to be generated based on the relevant information defined in the policy. Doing so would have ensured the number of the generated index(es) would be consistent with the searchable encryption mechanism as intended, instead of having the system to make arbitrary decisions or wait for input from the administrator.

14. Claim 21: “...wherein one or more of the policy assignment module, the encryption module, the storage module, and the management module are implemented on different devices, wherein the devices are selected from one or more of access management devices, proxy devices, gateway devices, and network controllers.”

154. Claim 21 requires that various recited modules (or functions) are implemented on different devices. In my opinion, Herrmann suggested claim 21.

155. Specifically, as explained above for claim 1, a POSITA would have found it obvious to use Herrmann’s gateway server to perform the recited functions in claim 1 (and similarly recited in claim 16). (*Supra* § **Error! Reference source not found..**)

156. Herrmann disclosed an **integrity server** that performed similar functions, such as managing policies and access requests based on policies. (Herrmann, ¶¶60, 65, FIG. 3.)

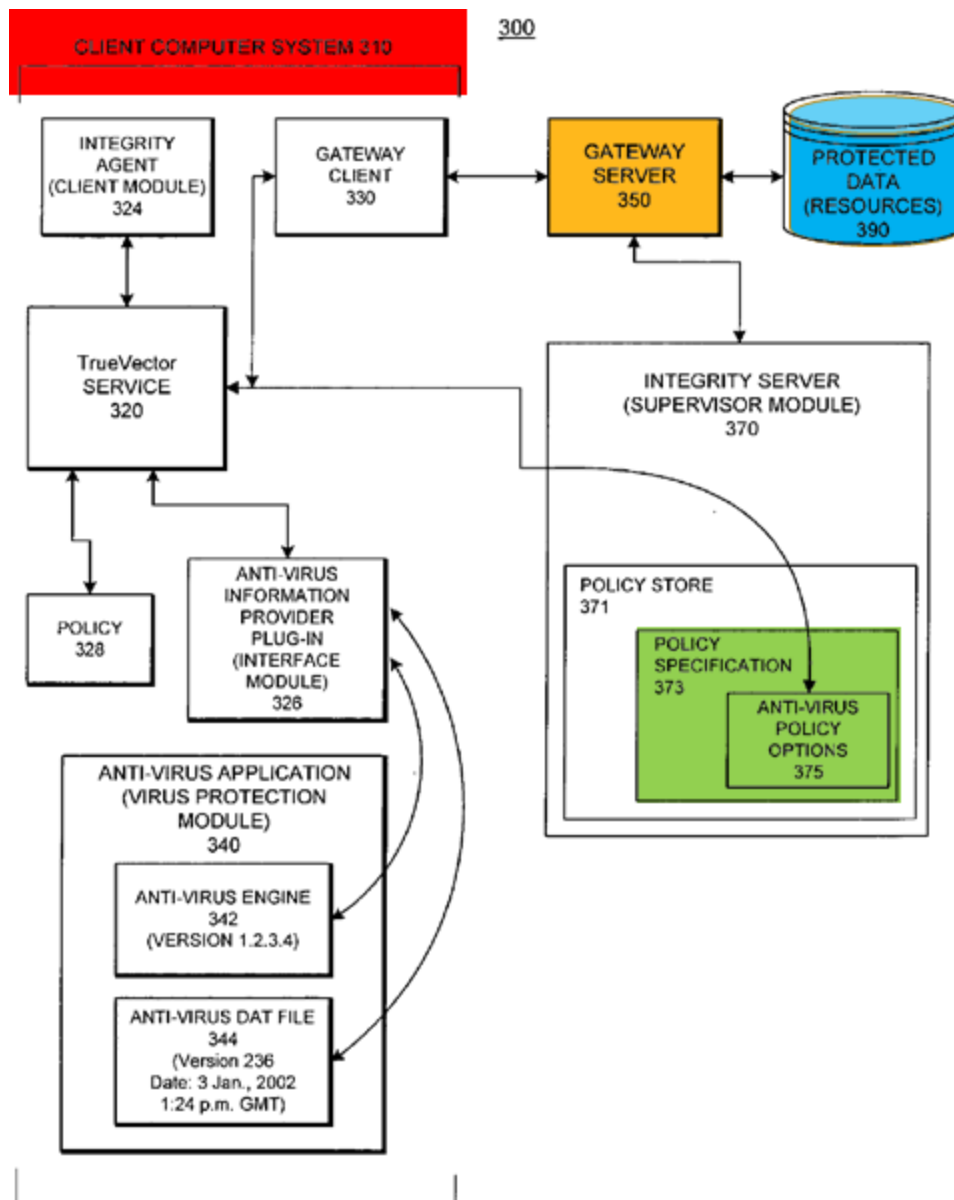


FIG. 3 (Annotated)

157. Herrmann also described that its integrity server and gateway server could be “installed on the same machine” or different machines. (Herrmann, ¶65.) Thus, based on these teachings, a POSITA would have been motivated to implement these recited modules into different machines. For example, the

encryption and storage modules could have been implemented on Herrmann's gateway service, and the policy assignment and management modules could have been implemented on Herrmann's integrity server (or Cidon's management server). Doing so would have allow these functions to be performed parallelly, thereby increasing processing power and efficiency, and helping balance task load.

158. A POSITA would have had a reasonable expectation of success in making such modification because distributed computing, i.e., using different devices to implement related tasks, had been well-known, Herrman and Cidon already contemplated that the functions described therein could be configured as different modules, and the combination would have yielded expected results.

(Herrmann, ¶65; Cidon, ¶¶101-102).)

15. Claim 22: "...further comprising a mediation module, which, when executed by the one or more processors makes implementation of one or more of the policy assignment module, the encryption module, the storage module, and the policy management module independent of the service provider of the one or more cloud platforms."

159. In my opinion, Cidon rendered claim 22 obvious because, consistent with the description of the "mediation module" in the specification of the '678 Patent, Cidon's management server "mediate[d] vendor specific protocol/APIs thereby facilitating geographical and provider diversity in data placement, making the system agnostic with respect to specific cloud vendor APIs." ('678 Patent,

4:46-49.)

160. Cidon described its management server similarly as being “connected to consumer-based file repositories (e.g., Dropbox 13, Gmail 11, or others such as Evernote) through Application Interfaces (API) 101 and 103” and “provide[ing] user management, API access (as well as API feed) to cloud storage services such as Dropbox and Box.” (Cidon, ¶¶100, 113, FIG. 2.)

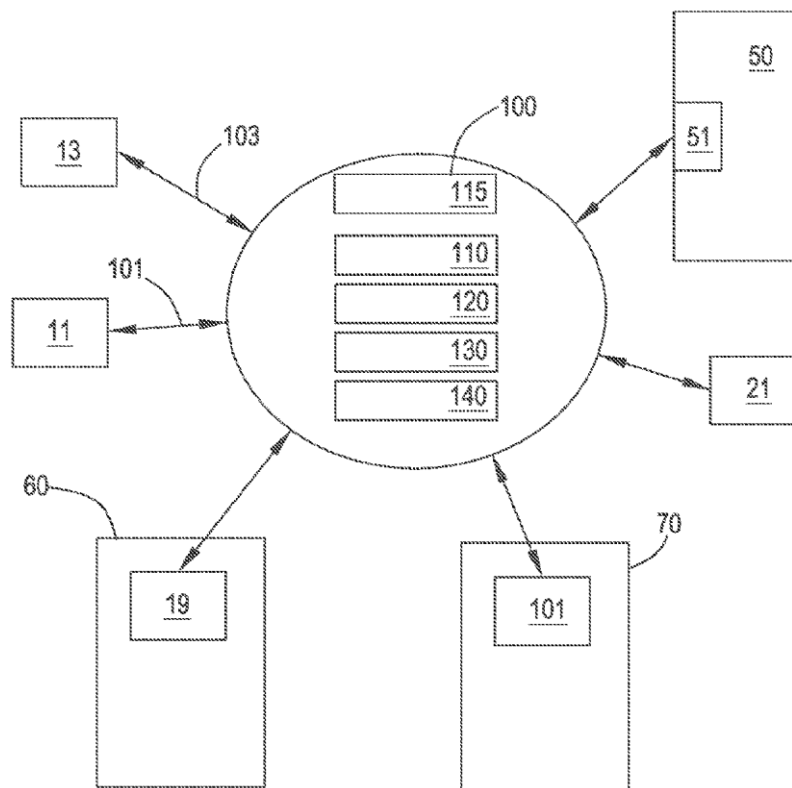


FIG. 2

161. Additionally, as I explained above, Cidon described that its encryption and storage were carried out independent of the service provider of the one or more cloud platforms. (See *supra* §§ I.A.7, I.A.11.) Cidon’s management server also

“serve[d as] a proxy” between the cloud services (connected to which via vendor-specific APIs, e.g., APIs 101, 103) and a user device. (Cidon, ¶221; FIG. 2.)

162. Further, Cidon described its management server as “a broad term to describe the complete software system which [was] responsible for encrypting files in the cloud and enforcing secure access to these cloud files” and “include[d] various software...components” that when executed by its processing unit (processor), to execute the functions described therein, including the function performed by this so-called “mediation module.” (Cidon, ¶102; *see also supra* §§ I.A.4.b, I.A.4.d.)

163. From these disclosures of Cidon, a POSITA would have found it obvious to have included the so-called mediation module in Cidon’s management server 100 to perform the aforementioned functionalities for serving as a “proxy” because the user device and various cloud platforms. As noted earlier, Cidon also described implementing “any or all of the methods described above, including any stages—and any combinations” using executable instructions or a computer program. (Cidon, ¶50; *see also supra* §§ I.A.4.b, I.A.4.d; Cidon, ¶102.)

16. Claim 26: “...wherein the group of policies are stored in a database operatively coupled with the system.”

164. In my opinion, Cidon rendered claim 26 obvious. (*See supra* § I.A.4.e.)

B. Ground 2: Claim 6 was obvious over Cidon, Shikfa, Herrmann, and Fellenstei

1. Claim 6: “wherein the step of configuring the namespaces comprises storing content of the file as filenames.”

165. In my opinion, Cidon and Fellenstei rendered claim 6 obvious.

166. Fellenstein explained claim 6 in its background section by describing key words (content of the file) that were added to filenames, which could then be used to retrieve the file by specifying those keywords. (Fellenstein, ¶8 (“Another **widely used** method [was] one in which one or more keywords for retrieval [were] added to each file name in advance and the file name having the keywords one of which matches an inputted retrieval word [was] regarded as a result of retrieval.”))

167. A POSITA would have been motivated to add this well-known functionality into the Cidon-Herrmann-Shikfa combination because, in doing so, in addition to searching all the contents of a file segment (e.g., parsing the file and extracting content), the modified system would have provided another option for only searching the filenames, which could be done with a simple string matching operation, and would have resulted in a quicker search. (*Id.*) In the modified system, one or more keywords (disclosed in Fellenstei) and file segment identifier (disclosed in Cidon; *see supra* § I.A.3.f) would have been included as part of the filename of a file segment.

168. A POSITA would have reasonably expected success in combining

Cidon with Fellenstei because assigning keywords to a filename had been well-known, Fellenstei's technique would have been compatible with Cidon's system, and the combination would have yielded expected results.

C. Ground 3: Claim 15 was obvious over Cidon, Shikfa, Herrmann, and Hirai

1. Claim 15: "...wherein said encrypting further comprises encrypting one or more of filename and search indices."

169. In my opinion, Cidon, Shikfa, and Hirai rendered claim 15 obvious.

170. Shikfa disclosed generating search indices that contained "one or more keywords for the encrypted documents" and were used for searching the encrypted documents. (Shikfa, ¶29.). Shikfa also disclosed encrypting those indices to generate encrypted indices 28. (Shikfa, ¶29, FIG. 2.)

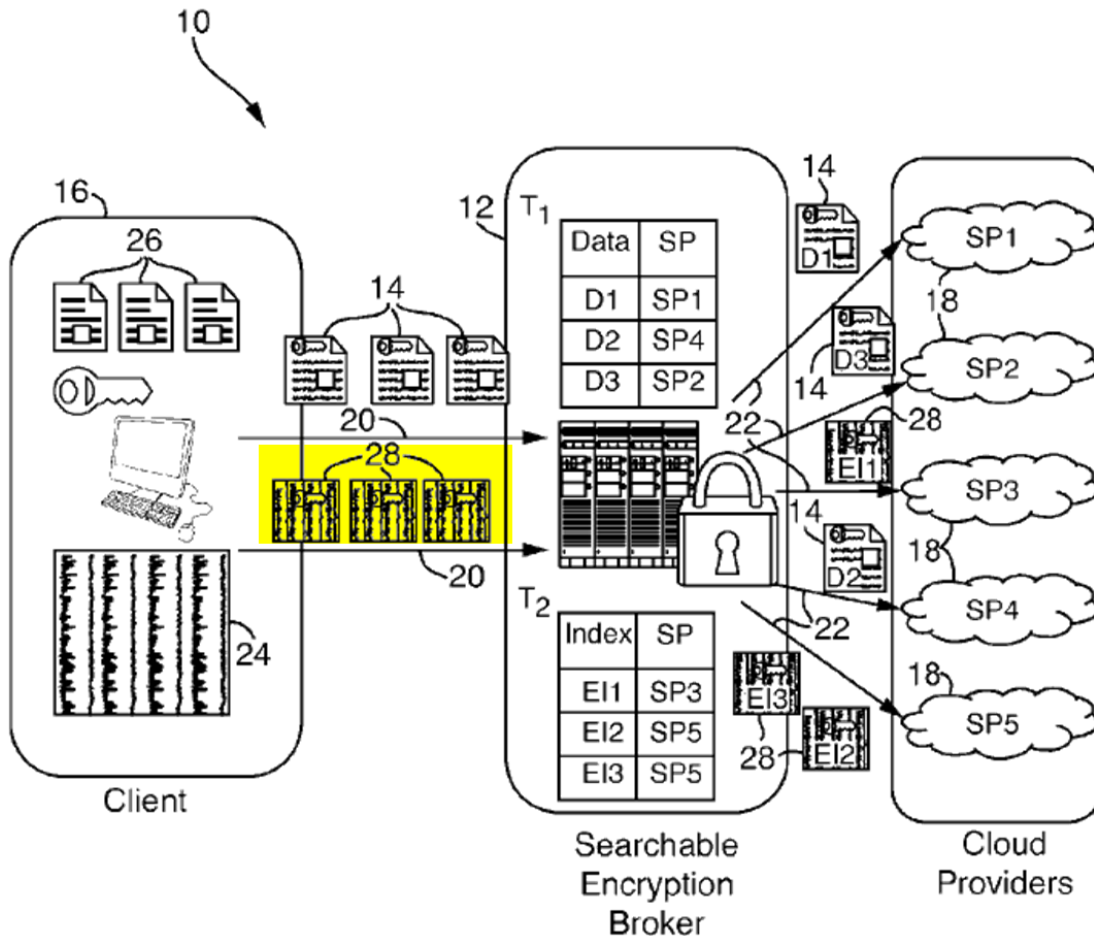


FIG. 2 (Annotated)

171. Shikfa did not explicitly disclose encrypting filenames, but Hirai described a file data retrieving system that generated “a file name and a plurality of key words related to the file data” and cryptographically encoded “the key words and the file name” using, for example, PGP encryption. (Hirai, 5:64-6:13.)

172. Filenames and search indexes were often descriptive of the file contents and could reveal confidential information. Thus, a POSITA would have been motivated to add Shikfa’s and Hirai’s functionalities for encrypting filenames

and indexes to Cidon's system. Doing so would have prevented any confidential information in the filenames and search indexes from being revealed to the provider of the cloud storage that stored the files. (Shikfa, ¶50 (the user "does not have to compromise its privacy needs to accommodate document searchability.").)

173. A POSITA would have reasonably expected success in combining Cidon with Shikfa and Hirai, because techniques for encrypting data like search indices and filenames had been well-known, Shikfa and Hirai's techniques would have been compatible with Cidon's system (given that Cidon's system could already encrypt data), and the combination would have yielded expected results.

D. Ground 4: Claims 1-3, 8-19, 21-27 were obvious over Auradkar, Chiueh, and Chambers

1. Overview of Prior Art

Auradkar:

174. Like the '678 Patent, Auradkar concerned storing encrypted data in cloud storages and managing access to the encrypted data based on policies assigned to users, as shown in Figure 1. (See, e.g., Auradkar, Abstract, FIG. 1.)

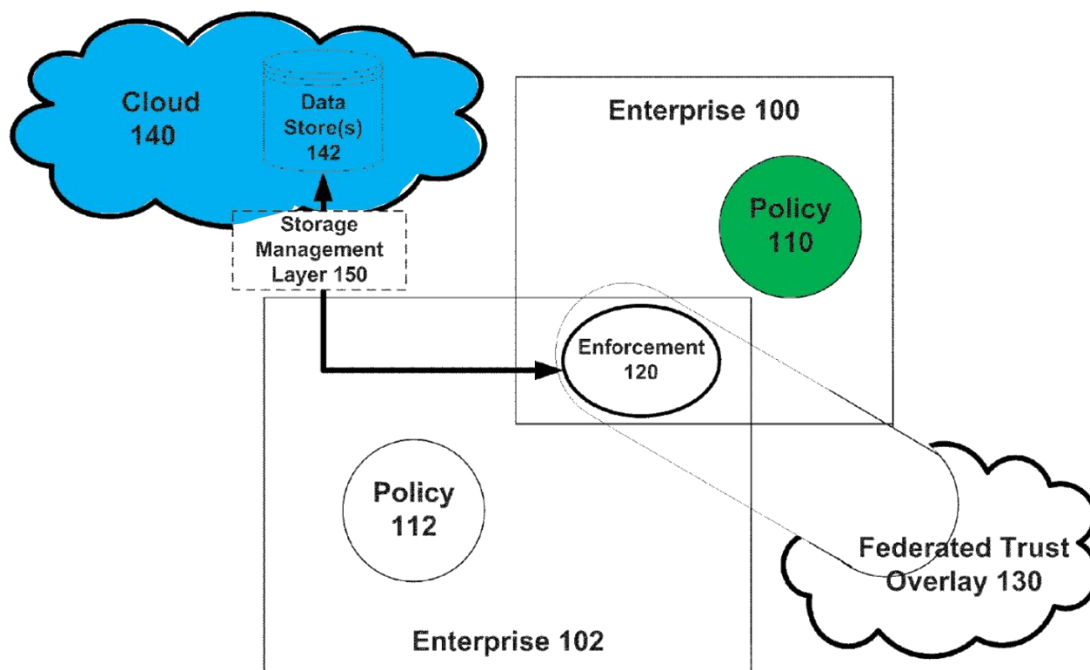
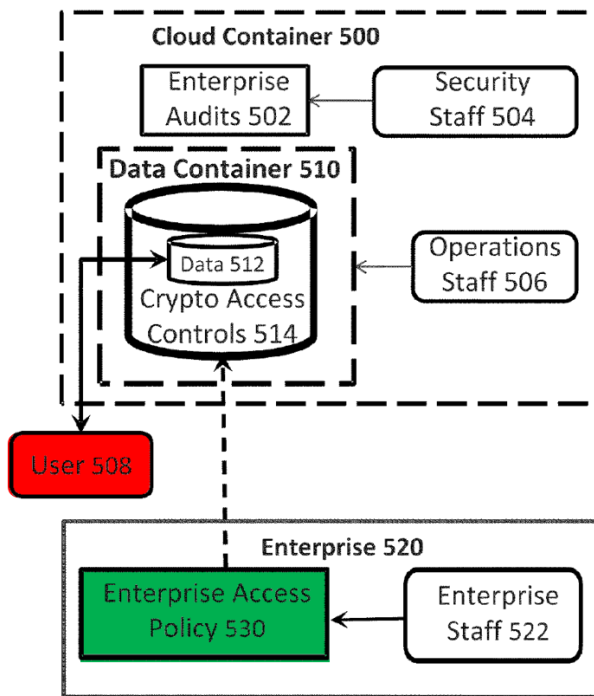


FIG. 1 (Annotated)

175. For example, Auradkar disclosed assigning to **enterprise staff** a **policy** for managing access to and processing a file to be stored on **cloud container 500**. (Auradkar, ¶¶112-113, 116-117, FIGS. 1-2, 5-6.)



Data as Custodian

- Containers are Redundant for Security
- Access is Enforced by Crypto Wrapper
- Policy is Set by Owner and guaranteed by Crypto Wrapper

FIG. 5 (Annotated)

176. Like the '678 Patent's policy, Auradkar's policies defined how the data stored in the cloud "[could] be accessed, manipulated, retrieved, searched" and whether to be encrypted and the type of encryption (e.g., AES 256 encryption) to be used. (Auradkar, ¶¶113, 96.)

177. As shown in Figure 15, Auradkar also described encrypting documents (e.g., **document 1500**) and associated keywords (1510), and uploading the **encrypted document** and encrypted keywords to "the **cloud**." (Auradkar, ¶130, FIG. 15.)

Technology for Confidentiality: Searchable Encryption

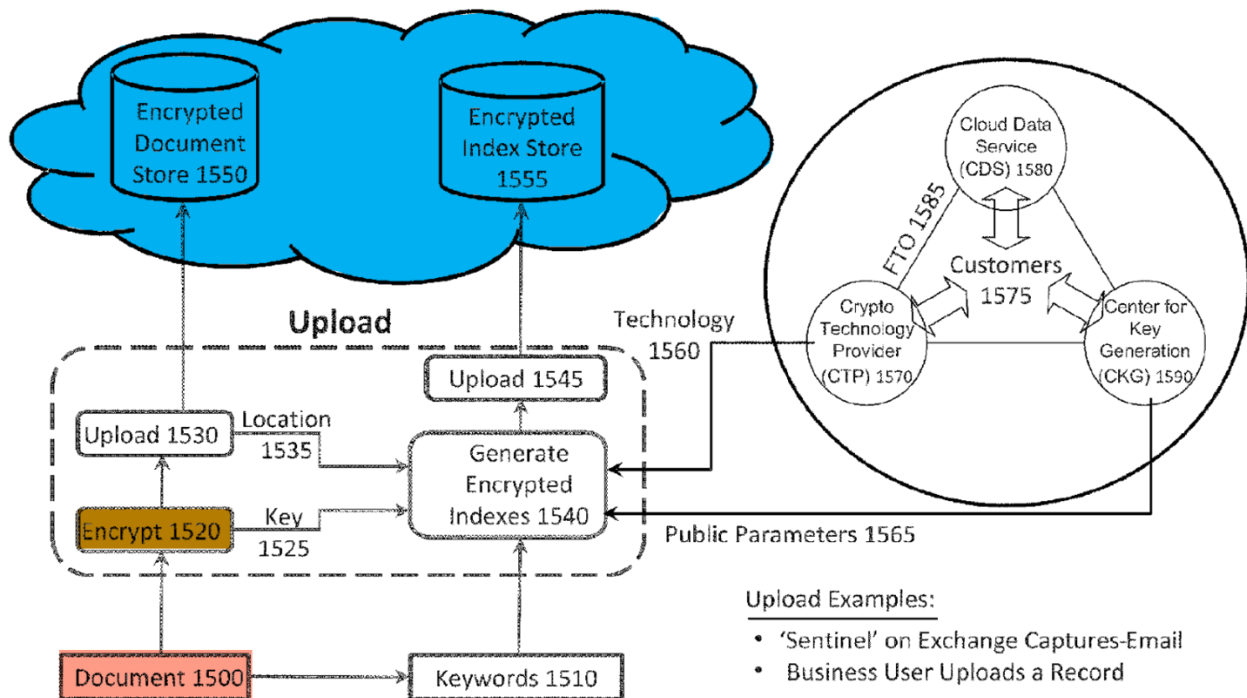


FIG. 15 (Annotated)

178. As shown in Figure 15, when searching encrypted documents, “customer 1675 form[ed] a query 1600, and then acquire[d] a trapdoor 1610,” which was “sent along with query 1600 to the cloud.” (Auradkar, ¶133, FIG. 16.)

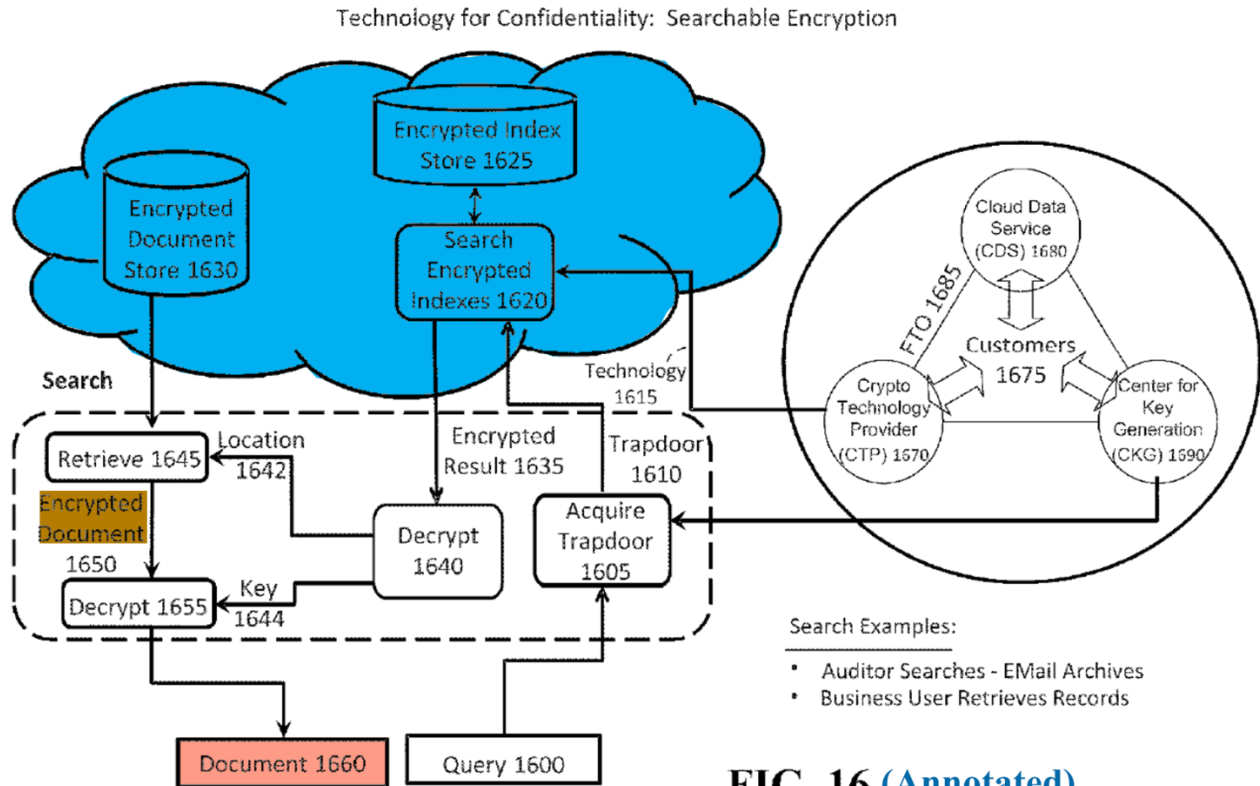


FIG. 16 (Annotated)

179. “[B]ased on a search of the indexes, various content or records [could] be identified matching a given index or indexes and then the user [could] either access the matching content or records based on privileges or not.” (Auradkar, ¶127.) Specifically, “[i]n the cloud, the encrypted indexes in encrypted index store 1625 [were] searched...[, and]results 1635 [were] then returned still encrypted and decrypted at 1640.” (Auradkar, ¶133.)

Chiueh:

180. Chiueh was also concerned with storing searchable encrypted documents on a remote site. Chiueh disclosed dividing a document into individual pages, encrypting individual pages, assigning a document identification (ID) to

each of the encrypted pages, and uploading encrypted pages to a remote server.

(Chiueh, ¶¶20, 40, FIG. 2.)

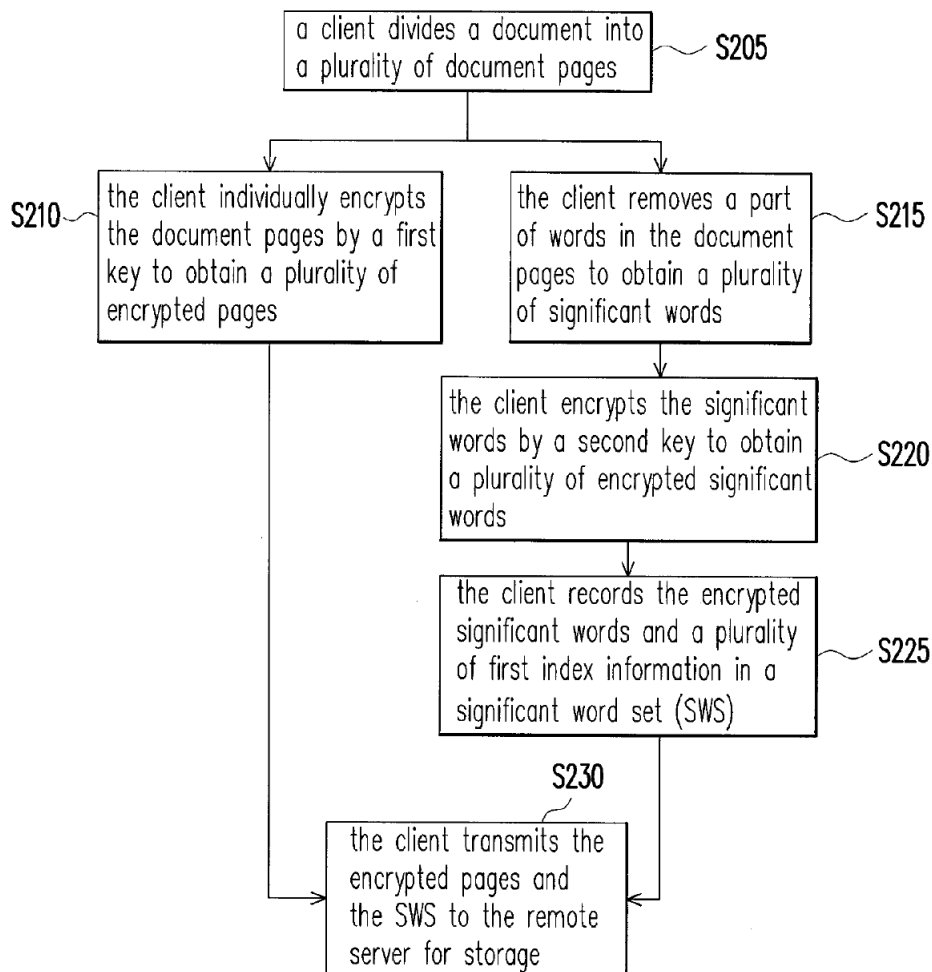


FIG. 2

181. As shown in Figure 2 and similar to the '678 Patent and Auradkar, Chiueh disclosed creating an index (e.g., a significant word set (SWS)) mapping encrypted keywords to corresponding document ID(s) of the encrypted pages, so that the content of the encrypted pages could be searched using keywords in the future. (Chiueh, ¶¶21-23, FIG. 2.)

182. Chiueh's Figure 3 described a search process similar to that described in the '678 Patent and Auradkar. Specifically, a search keyword from the user was encrypted, and Chiueh's system searched a global search index (GSI) that contained previously uploaded encrypted keywords to identify any document ID (for individual encrypted pages) that matched the encrypted search keyword. (Chiueh, ¶¶23-25, 40, FIG. 3.)

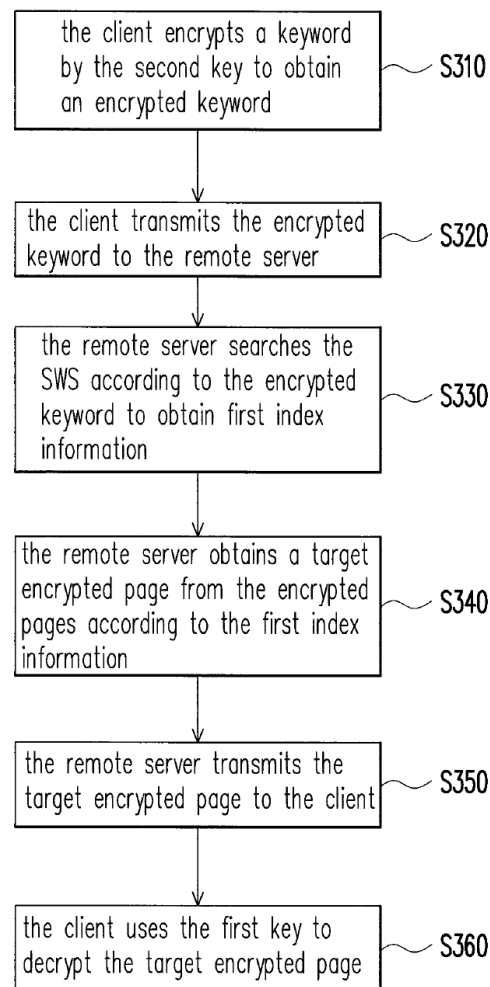
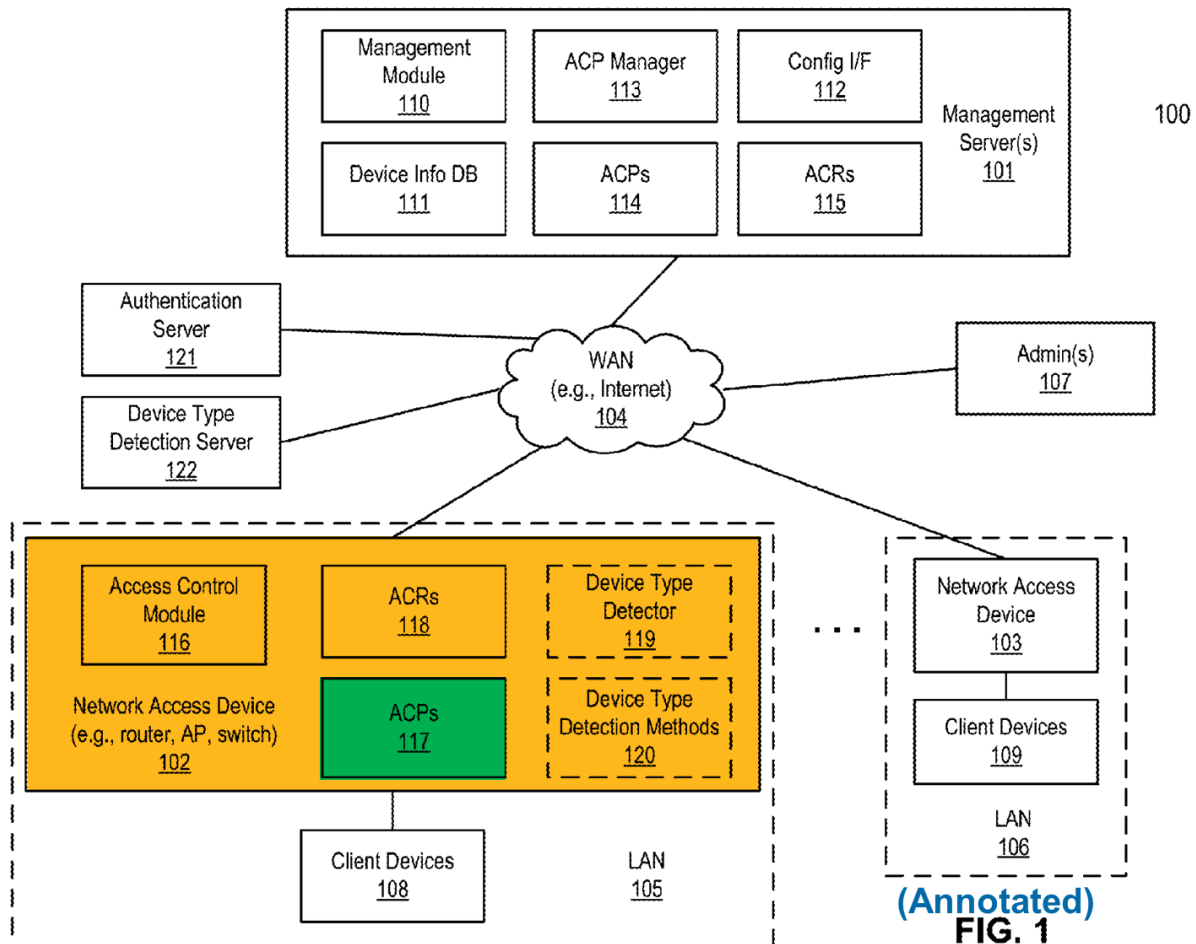


FIG. 3

Chambers:

183. Chambers disclosed a network access device (**NAD**) 102 that “operate[d] as a gateway device...to LAN[] 105...where various client devices 108[]...[could] be communicatively coupled to LAN[] 105.” (Chambers, ¶24, FIG. 1.)



184. Chambers’ NAD 102 included an access control module (ACM) 116 that “select[ed] and applie[d] appropriate...[access control policies (]**ACPs**[]) 117” to client devices 108. (Chambers, ¶46; *see also id.*, ¶21.) In the grounds below, Chambers is used to illustrate using a gateway device to implement the claim limitations was an obvious endeavor.

2. Motivations to Combine

185. A POSITA would have been motivated to combine Auradkar, Chiueh, and Chambers with a reasonable expectation of success.

186. First, all these references were concerned with improving computer networking and security for accessing files stored in a remote site from client devices. (Auradka, Abstract; Chiueh, ¶3; Chambers, Abstract.) Thus, a POSITA would have found it obvious that Auradkar's system could have benefited from Chiueh and Chambers' complementary techniques with the same or similar goal.

187. Second, the Auradkar-Chiueh-Chambers combination would have also produced a better system. For example, while Auradkar's system allowed searching encrypted data using searchable encryption techniques, a POSITA would have been motivated to modify this technique by Chiueh's searchable encryption mechanism that allowed searching individual document pages. Doing so would have enabled a more efficient search and retrieval of individual pages (chunks) instead of the whole document. (Auradkar, ¶¶127, 133, FIG. 16; Chiueh, ¶¶20, 40, FIGS. 2-3.) Additionally, both Auradkar and Chambers aimed to implement access policies for accessing protected enterprise data, and a POSITA would have understood that incorporating Chambers' dedicated gateway as an interface between the client and protected data sources would have enhanced network security by, e.g., shielding these sources from the client.

188. Additional motivations are discussed below.

3. Claim 1

189. In my opinion, Auradkar, Chambers, and Chiueh rendered obvious claim 1.

a. Claim 1[pre]

190. Auradkar disclosed a method for encrypting files and storing encrypted files. (*See, e.g.*, Auradkar, Abstract (“A digital escrow pattern and trustworthy platform is provided for data services including mathematical transformation techniques, such as searchable encryption techniques, for obscuring data stored at a remote site or in a cloud service, distributing trust across multiple entities to avoid a single point of data compromise.”), FIG. 15.)

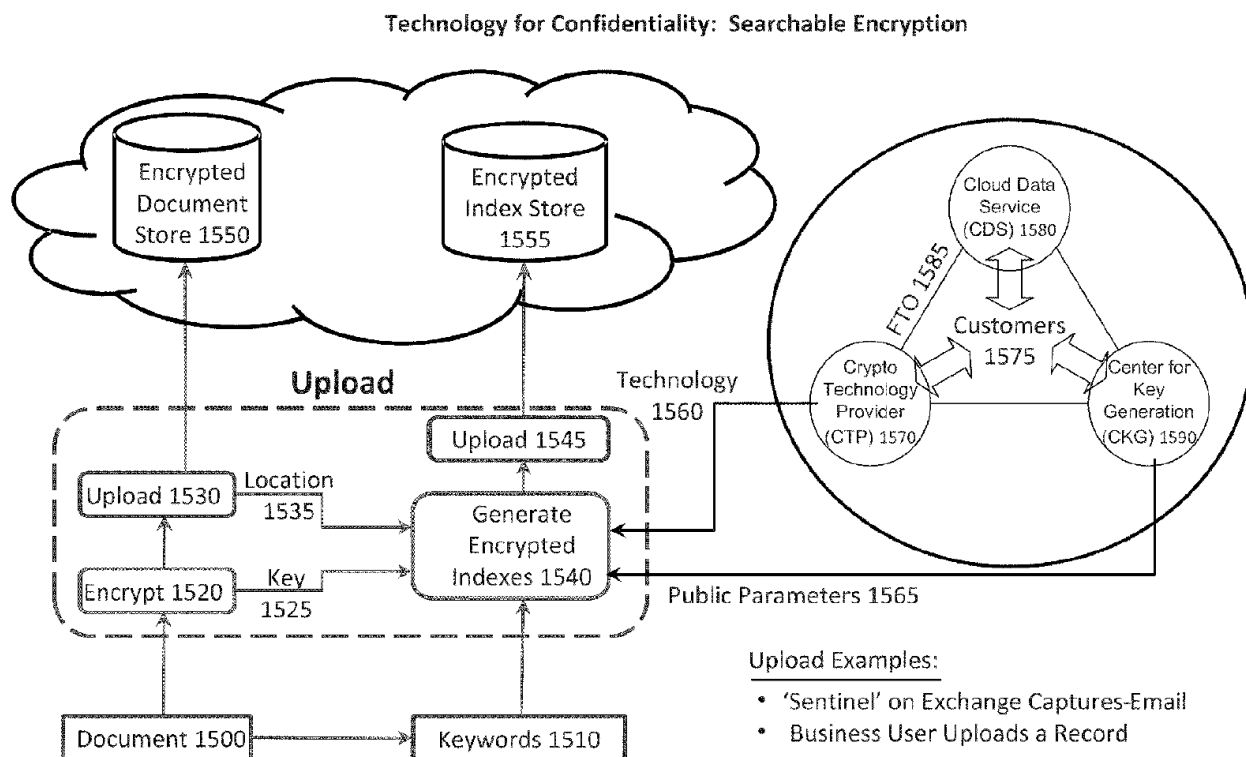


FIG. 15

b. Claim 1[a][i]

191. Auradkar disclosed assigning to **enterprise staff** (users) a **policy** for managing access to and processing a file to be stored on **cloud container 500** (cloud platform). (Auradkar, ¶¶112-113, 116-117, FIGS. 1-2, 5-6.)

192. Similar to the '678 Patent's policy, which "can be selected, by an administrator...based on the role, responsibility, enterprise practices" ('678 Patent, 9:11-19), Auradkar's policy (e.g., policy 530) was set by "some of...enterprise staff 522" and different policies were applied to "different accessing entities or tasks...to ensure that visibility is restricted to those to whom access should be enabled." (Auradkar, ¶117, FIGS. 5-6, ¶119 ("[E]nterprise 620 define[d] its

enterprise access policy 630 applicable to enterprise staff 622....”).)

193. Moreover, Auradkar’s policy was for managing access to and processing files to be stored in cloud (cloud platform). For example, as shown in Figure 5 (below), Auradkar’s policy could “affect how data 512 stored in a data container 510 of a **cloud container 500** [could] be **accessed, manipulated,** retrieved, searched,” and “when users 508 of data 512 attempt[ed] to access such data 512, various **crypto access controls 514** guided by, but separated from, **enterprise access policy 530** protect[ed] the data 512 from unwarranted access by users 508.” (Auradkar, ¶117, FIG. 5.)

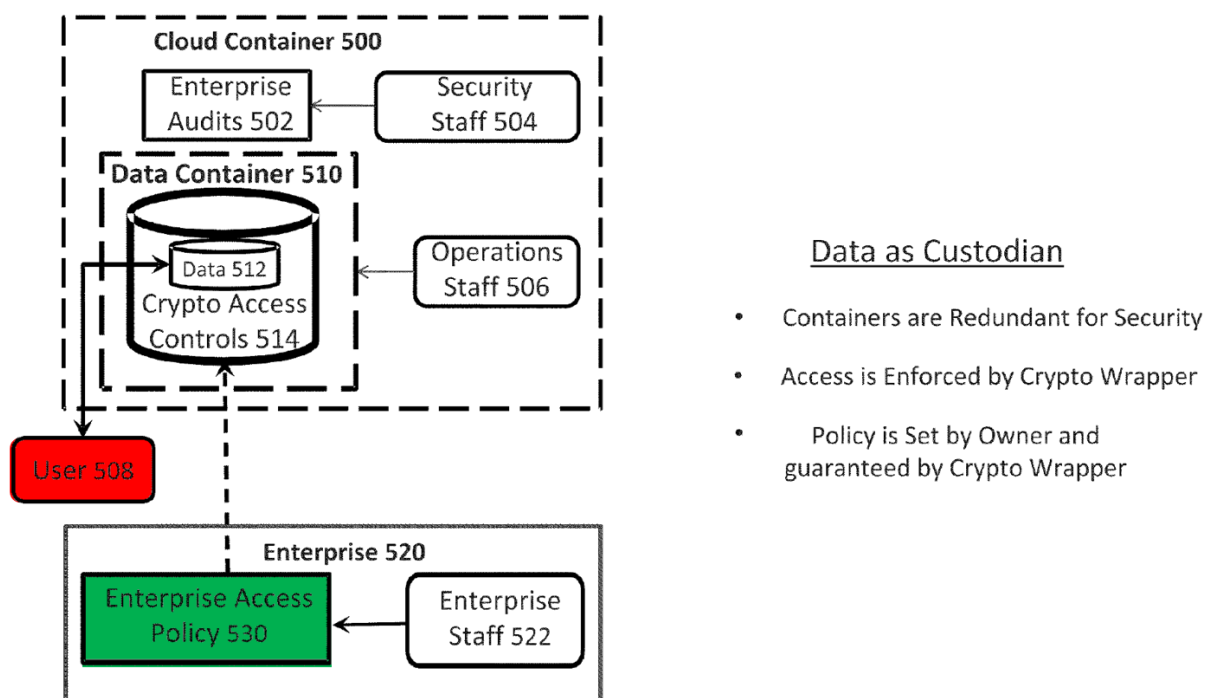
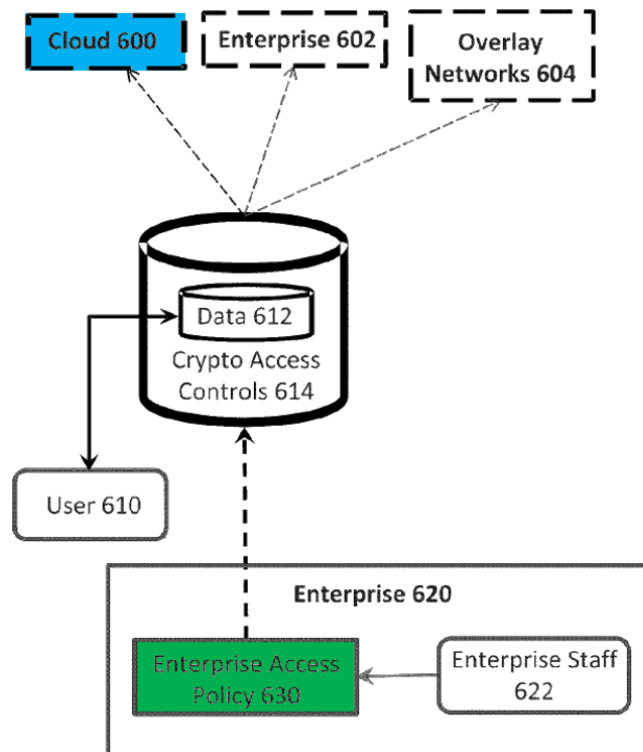


FIG. 5 (Annotated)



Data can Transcend Containers

- The Systems / Security Equation Changes
- Data can Straddle Containers
- Placement can optimize Access, Resilience, etc.
- New Challenges are Consistency, Versioning, Garbage Collection, etc.

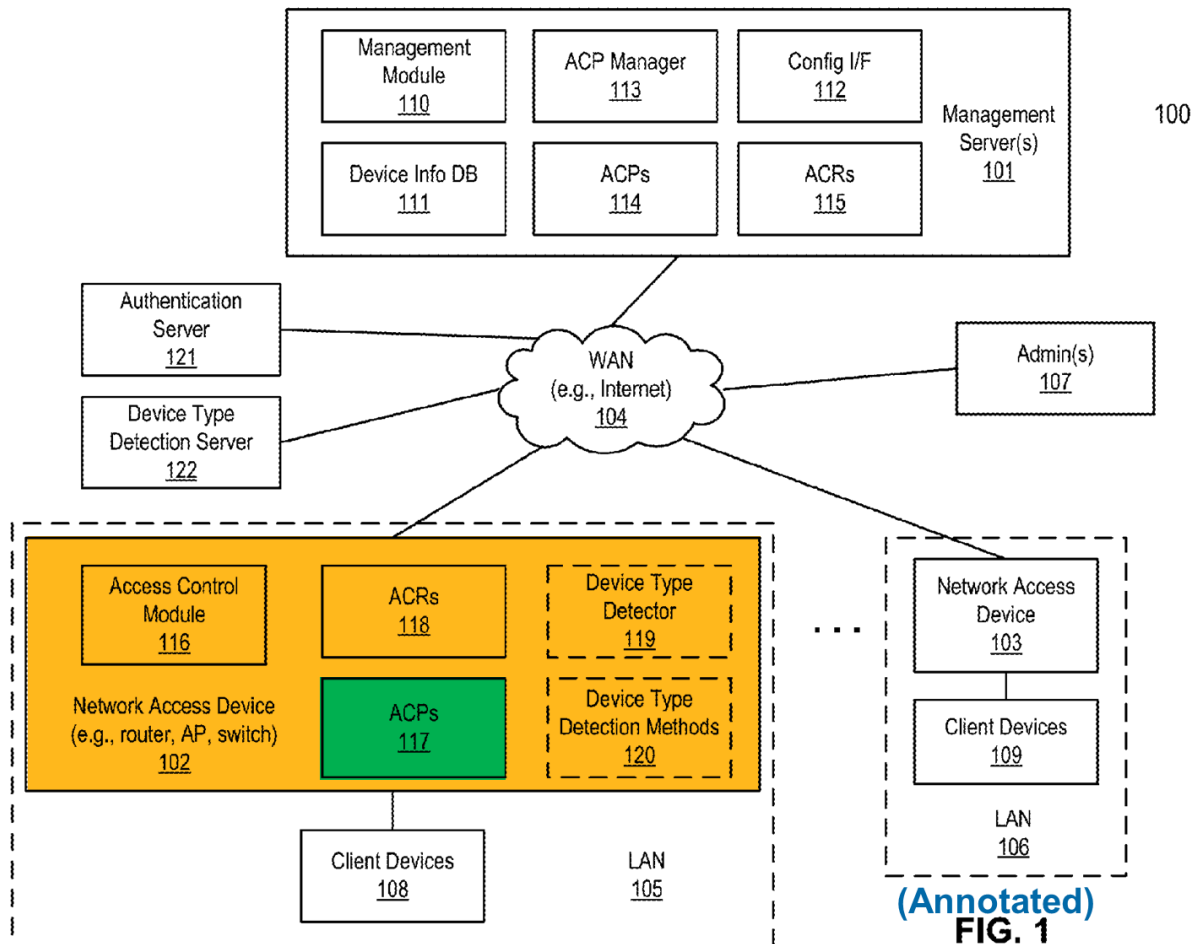
FIG. 6 (Annotated)

194. Similar operations were discussed in connection with Figure 6. (See *id.*, ¶119 (“[E]nterprise 620 define[d] its enterprise access policy 630 applicable to enterprise staff 622, while data 612 [was] stored remotely and protected by cryptographic access controls 614 applicable to users 610 wishing to access data 612. The system and users 610 are agnostic whether containers storing data 612 are stored in a cloud 600, somewhere at the enterprise 602, or stored via overlay networks 604.”), FIG. 6.)

Assigning the Policy:

195. While Auradkar disclosed assigning a policy to users, it did not explicitly disclose a **gateway device** assigning the policy. Chambers, however, disclosed this feature.

196. Specifically, Chambers disclosed a network access device (**NAD**) 102 (gateway device) that “operate[d] as a **gateway device**, an access point (AP), a network switch, *or a combination thereof*” and assigned an access policy to a client device. (Chambers, ¶24, FIG. 1.)



197. Chambers’ NAD 102 included an access control module (ACM) 116 that “select[ed] and applie[d] **appropriate...**[access control policies (**ACPs**)] 117,” stored in NAD 102, to client devices 108. (Chambers, ¶46.) After selecting and applying “an appropriate ACP,” NAD 102 (via its ACM) sent “update information” concerning the selected ACP to management server 101, which then

broadcasted the update information to other NACs. (*Id.*) A POSITA would have understood that by ***selecting and applying an appropriate*** access control policy to a client device without intervention from an administrator or the management server, Chambers' NAD 102 assigned this policy to the client device. Indeed, Chambers described its "[t]echniques for **automatically assigning**" and applying policies. (Chambers, ¶21)

198. Chambers' Figure 4A (below) also illustrated that "Group policy" assigned to "staff" and "iPad group," which a POSITA would have recognized constituted assigning policies to users or groups of users.

400

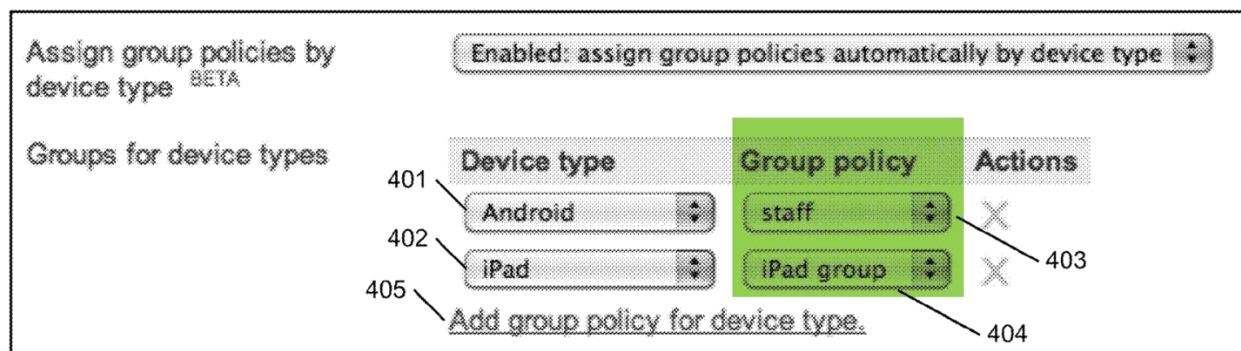


FIG. 4A (Annotated)

199. While Auradkar disclosed the claimed policy assigning function (as explained above), it did not explicitly disclose what device implemented this

function. A POSITA would have found it obvious to use Chambers' gateway to implement Auradkar's "assigning" function (and other claimed functions), and to automatically assign the pertinent security policies as described in Chambers.

Below is an illustration of this combination using Figure 15 of Auradkar.

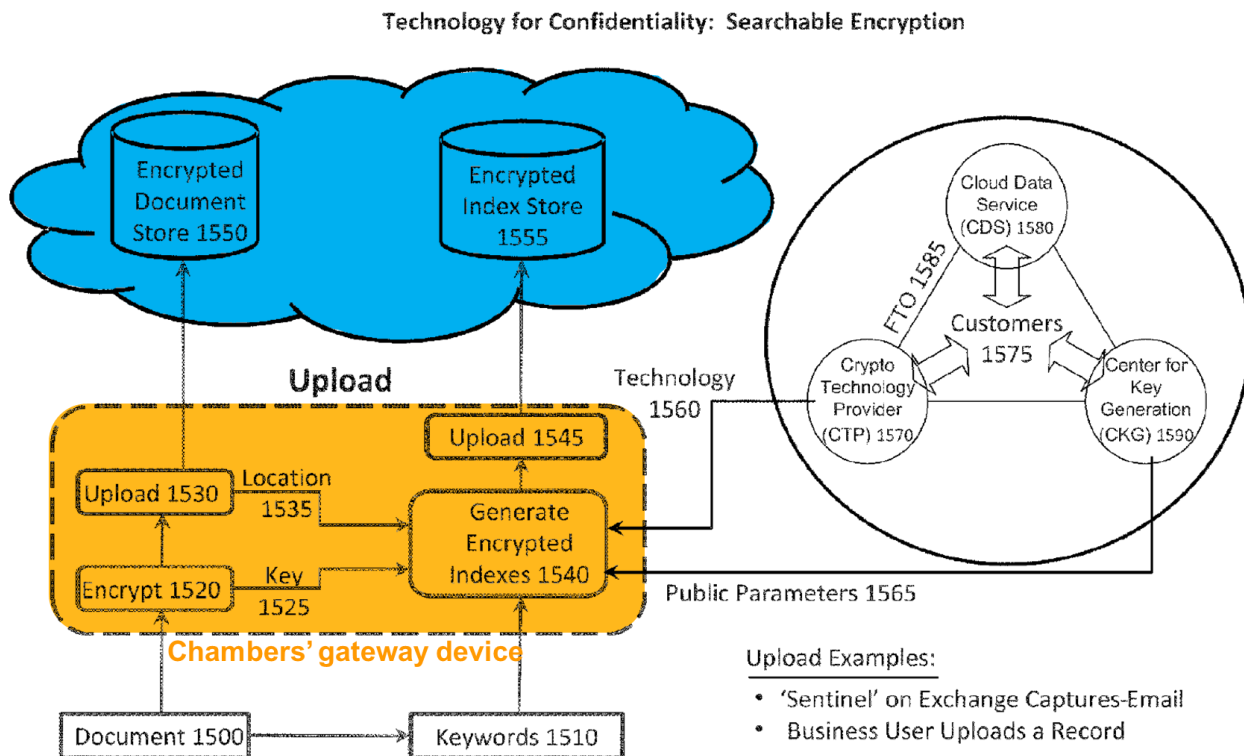


FIG. 15 (Annotated)

200. A POSITA would have recognized that this combination would be beneficial because an organization may have different types of users, and a user may use multiple devices (e.g., a laptop and smartphone) to access data stored in cloud. Incorporating the teachings of Chambers into Auradkar's system would have allowed to automatically assign a policy based on, e.g., the user's role in the organization or the device used by the user, would have resulted in consistent

application of the policy and would have avoided repeated policy configuration for the same user. Further, automatically configuring, assigning, and implementing policies (as disclosed in Chambers) would have improved system efficiency and reduced human errors. (Chambers, ¶¶5-6 (explaining that manual configuration and assignment of policies could be “cumbersome, time-consuming and error-prone”), 21.) Moreover, the modified system would have allowed better real-time protection by automatically applying (and assigning) an updated policy when such an update occurred. (Chambers, ¶29 (describing that Chambers’ NADs updated policies).)

201. As I noted earlier, using Chambers’ dedicated gateway as an interface between the client and protected data sources would have also enhanced network security by, e.g., shielding these sources from the client. (See Section VIII.D.2.)

202. A POSITA would have had a reasonable expectation of success in making this combination because gateway devices had been well-known devices used for network security-related functionalities such as controlling network traffic of client devices and managing access policies. (See, e.g., Chambers, ¶¶66, 82; Herrmann, ¶¶18, 77.) Thus, adding Chambers’ gateway and its functionalities into Auradkar’s system would have yielded predictable results.

c. Claim 1[a][ii]

203. Auradkar disclosed that its policy defined access rights of the user. For example, Auradkar disclosed that “[w]ith **policy and enforcement** within the

control of users or an enterprise, data and corresponding logs [were] encrypted and **accessible only with specific capabilities granted to a user.**” (Auradkar, ¶112.)

Auradkar also described that its policy could “affect how data 512 stored in a data container 510 of a cloud container 500 [could] be **accessed, manipulated, retrieved, searched**” and “when users 508 of data 512 attempt[ed] to access such data 512, various crypto access controls 514 guided by, but separated from, enterprise access policy 530 protect the data 512 from unwarranted access by users 508.” (Auradkar, ¶117.)

d. Claim 1[b]

204. Auradkar’s Figure 11 generally illustrated encrypting a “**record 1102** (e.g., data payload) and associated metadata and/or tags 1100” to produce encrypted metadata and tags 1110 and **encrypted record 1112**, on which “various operations 1120 can be performed, including “search of the data.” (Auradkar, ¶124, FIG. 11.)

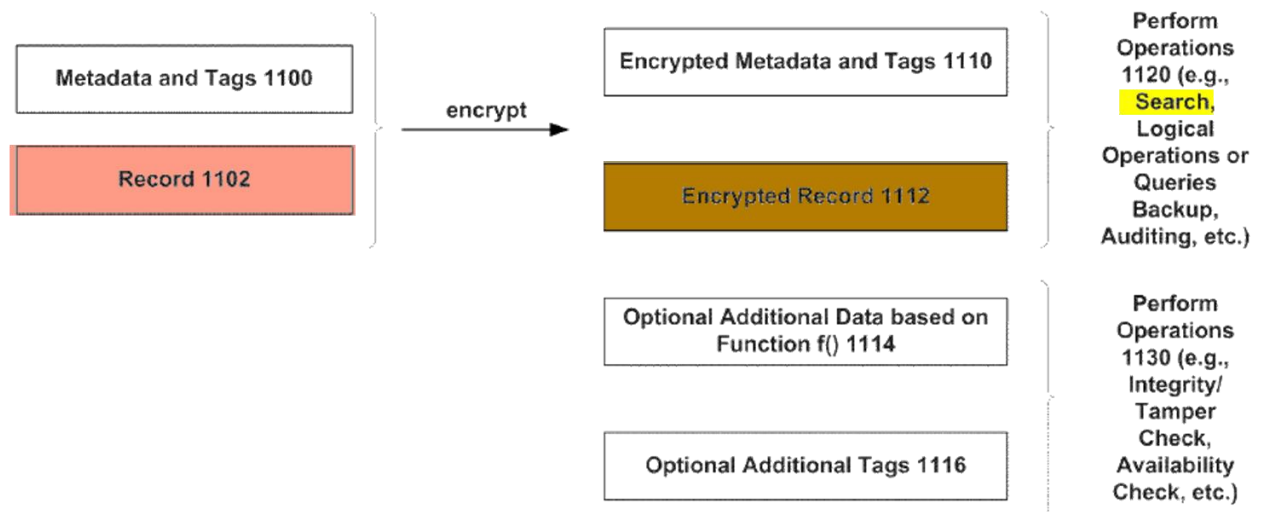


FIG. 11 (Annotated)

205. In the example of Figure 12, **payload 1202** and tags 1200, which included a filename (“FName: fox.txt”), were “encrypted to form encrypted tags 1210 and **encrypted data 1212** for operations 1220,” including search operations. (*Id.*, ¶125, FIG. 12.)

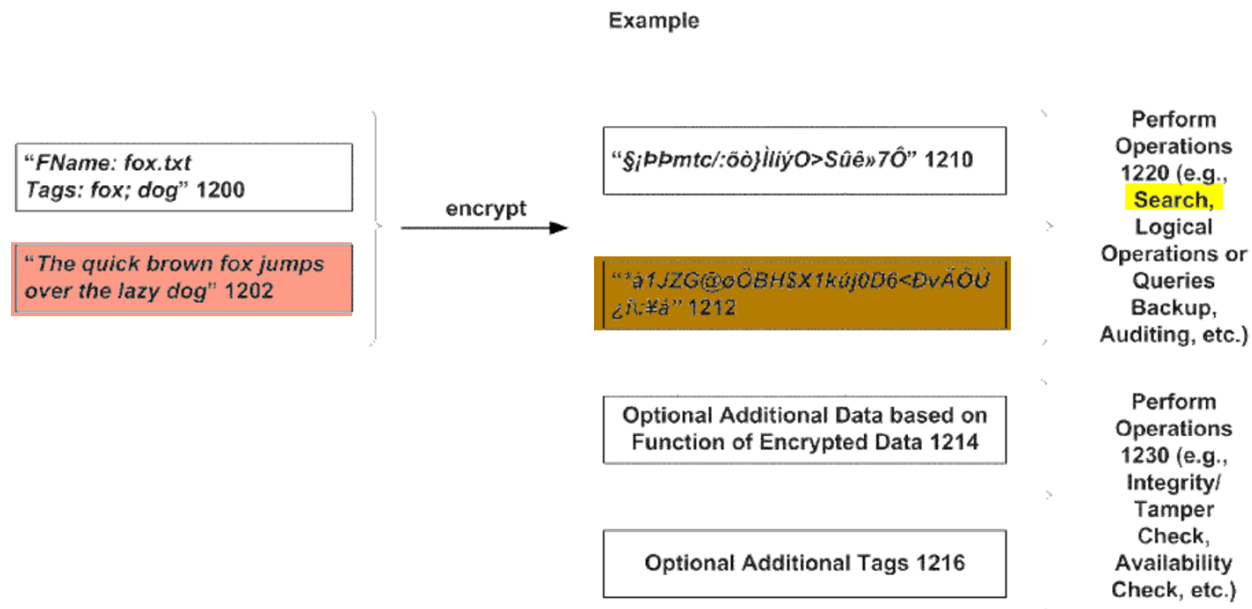


FIG. 12 (Annotated)

206. Auradkar also described encrypting a “**document 1500** with which various keywords 1510 [were] associated” and uploading the **encrypted document** to “the **cloud** into an encrypted document store 1550,” as shown in Figure 15. (Auradkar, ¶130, FIG. 15.)

Technology for Confidentiality: Searchable Encryption

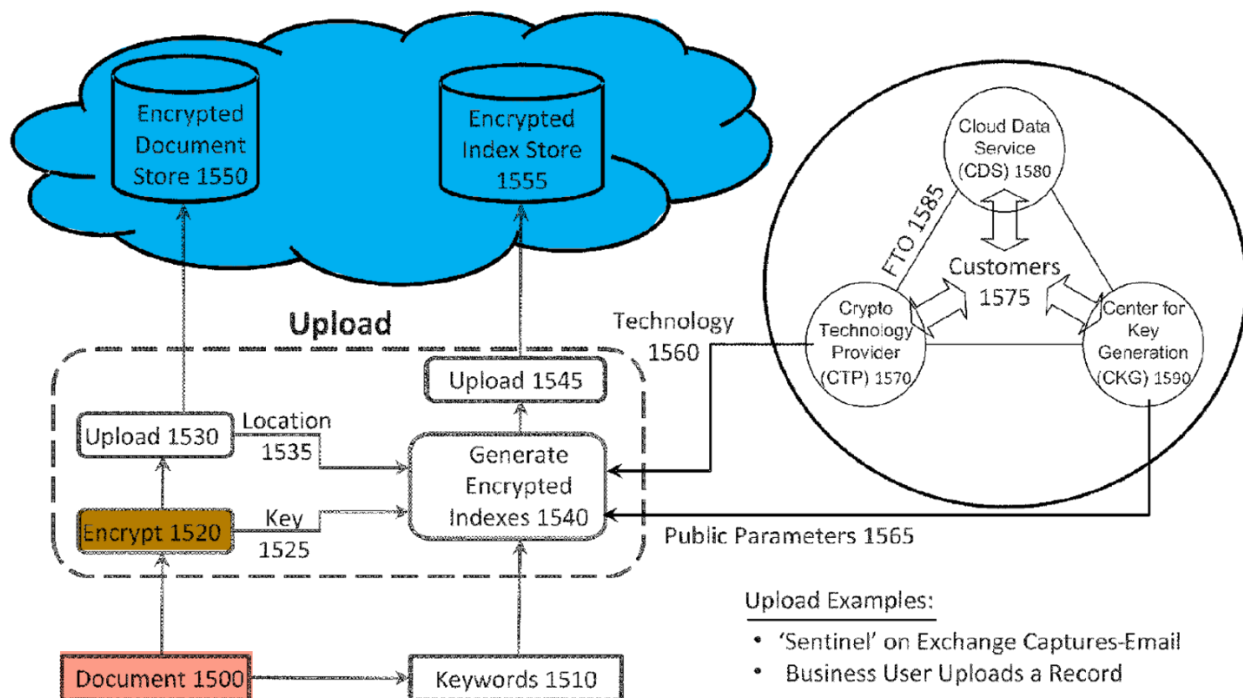
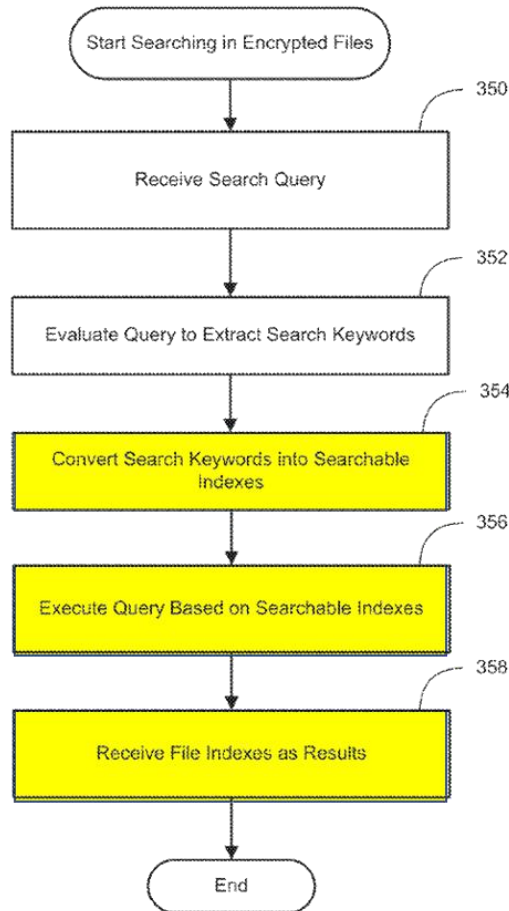


FIG. 15 (Annotated)

207. Similar to the '678 Patent's keywords (content of the file), which are indexed and encrypted (e.g., "search keyword 'Ronn' can be indexed as 'uevbwq' and Eric can be indexed as 'uastyb'"; '678 Patent, FIG. 3B, 12:44-47), Auradkar's keywords 1510 were also encrypted to generate encrypted indexes 1540, which were used for searching the encrypted document. (Auradkar, ¶130, FIG. 15.)



'678 Patent, Fig. 3B (Annotated)

208. For example, Auradkar described that to perform a search, “customer 1675 form[ed] a query 1600, and then acquire[d] a trapdoor 1610,” which was “sent along with query 1600 to the cloud.” (Auradkar, ¶133, FIG. 16.)

Technology for Confidentiality: Searchable Encryption

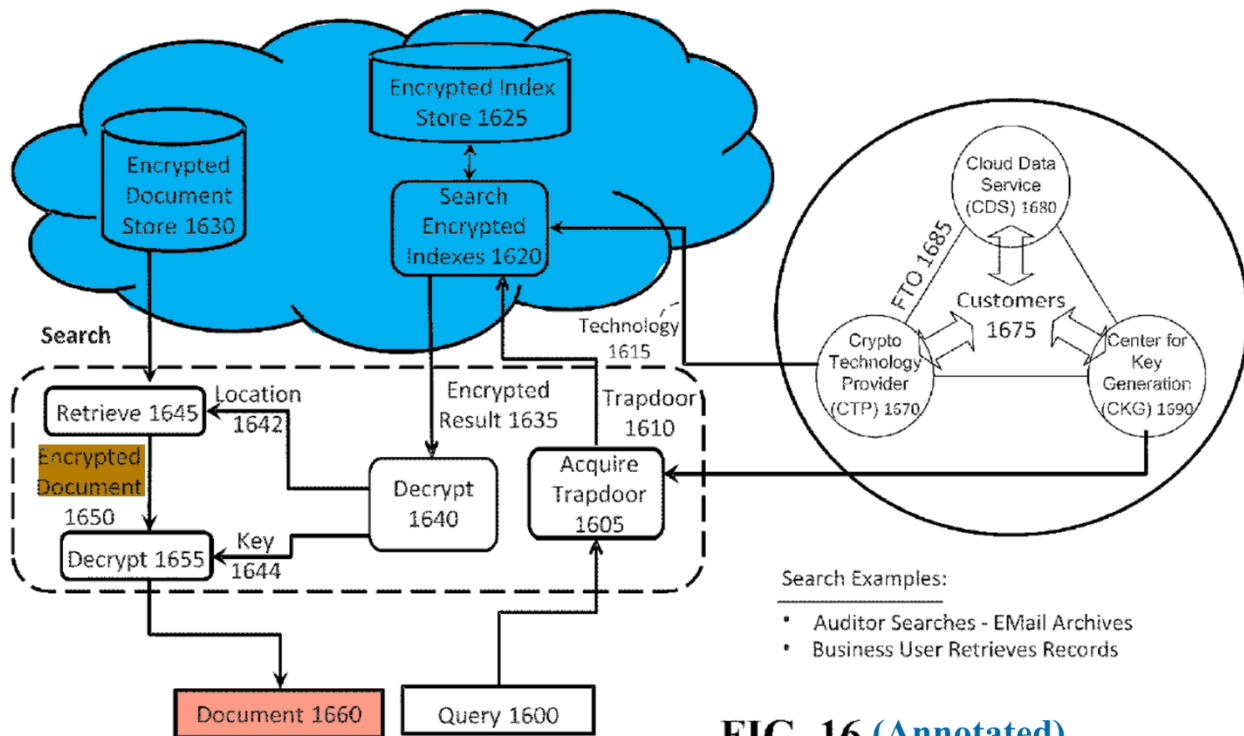


FIG. 16 (Annotated)

209. “[B]ased on a search of the indexes, various content or records [could] be identified matching a given index or indexes and then the user [could] either access the matching content or records **based on privileges or not.**” (Auradkar, ¶127.) Specifically, “[i]n the cloud, the encrypted indexes in encrypted index store 1625 [were] searched...[, and]results 1635 [were] then returned still encrypted and decrypted at 1640.” (Auradkar, ¶133.)

Cryptographic Key Information:

210. While the ’678 Patent does not explicitly define the claimed “cryptographic key information,” it describes that “a selected policy can be used to define encryption keys, decryption keys, and encryption type, among other

attributes for carrying out the encryption of data.” (’678 Patent, 4:4-7.) Consistent with this description, Auradkar’s policy defined the encryption type and the length of encryption/decryption key. Specifically, Auradkar’s policy specified that certain data, such as “sensitive[] medical data,” must be encrypted using an encryption method specified in the policy (e.g., “using **AES 256 encryption**”). (Auradkar, ¶96.) A POSITA would have understood that Auradkar is specifying the Advanced Encryption Standard (AES) encryption algorithm (encryption type) with a 256-bit encryption/decryption key. (Auradkar, ¶96.) (Other common AES key lengths were 128 and 192 bits.)

211. As I explained in connection with Limitation 1[a][i] (Ground 4), a POSITA would have found is obvious and would have had a reasonable expectation of success in combining Chambers’ gateway to Auradkar’s system such that the claimed operations would be carried out by the gateway. (*See supra* §I.D.3.b.)

e. Claim 1[b][i]

212. Auradkar disclosed this limitation because it disclosed breaking a payload (e.g., an XML document) (file) into XML fragments (chunks) and encrypting individual fragments based on corresponding “protection levels.” (Auradkar, ¶150, FIG. 19; *see also* ¶¶92, 95, 98.)

213. Specifically, Auradkar disclosed “breaking an **XML document**” (file)

into XML fragments (chunks) with “potentially different protection levels,” such as “Public, Secret and Top Secret paragraphs,” and encrypting XML fragments to generate **encrypted XML fragments**. (Auradkar, ¶¶150, 92, 95, FIG. 19; *see also* FIGS. 11-12.)

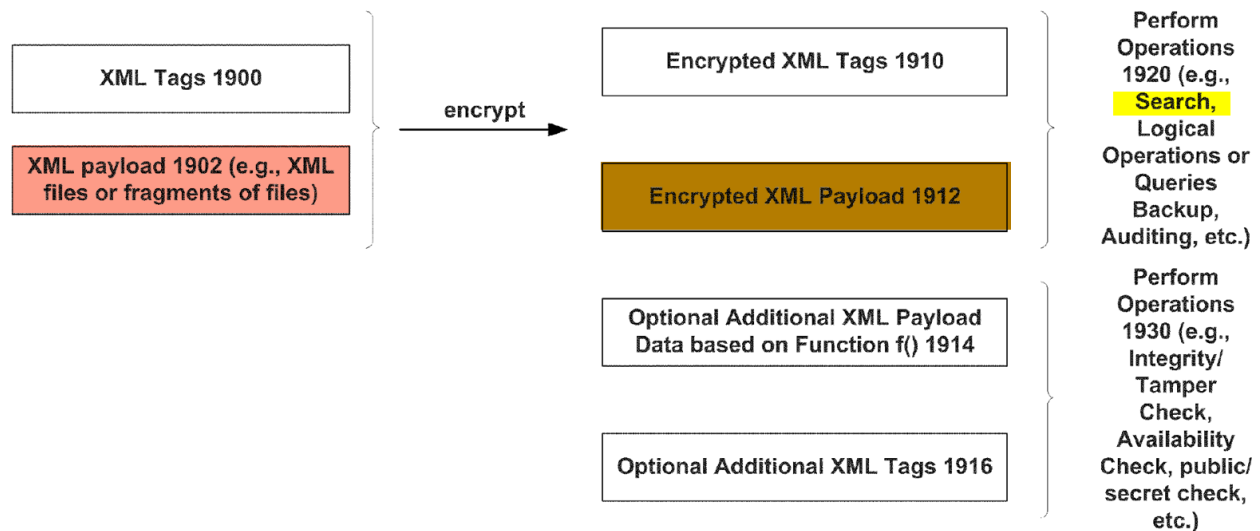


FIG. 19 (Annotated)

214. (See also Auradkar, ¶¶98 (describing that “the confidentiality level labels of public, secret or top secret [could] be applied to each payload element of the XML document **on a fragment by fragment basis**, and included in the auxiliary encrypted metadata to achieve highly granular **policy around access to parts** of the XML document”); 95 (explaining that a “person performing a search and requesting access with a Secret clearance would only get access to Public and Secret paragraphs,” and “a policy [could] be implemented that Top Secret content

[couldn't] be accessed from a wireless or remote device”).)

f. 1[b](ii)

215. Auradkar and the Auradkar-Chiueh combination taught or suggested this limitation.

216. Auradkar disclosed creating tags for payloads (e.g., payload 1200, XML fragment 1902 (chunks)), such as **tag 1200** that included the filename of the payload (FName: fox.tx) and **XML tags 1900**, and encrypting the tags into encrypted tags, which was used in operations 1230 such as searching. (Auradkar, ¶¶125, 150, FIGS. 12, 19.)

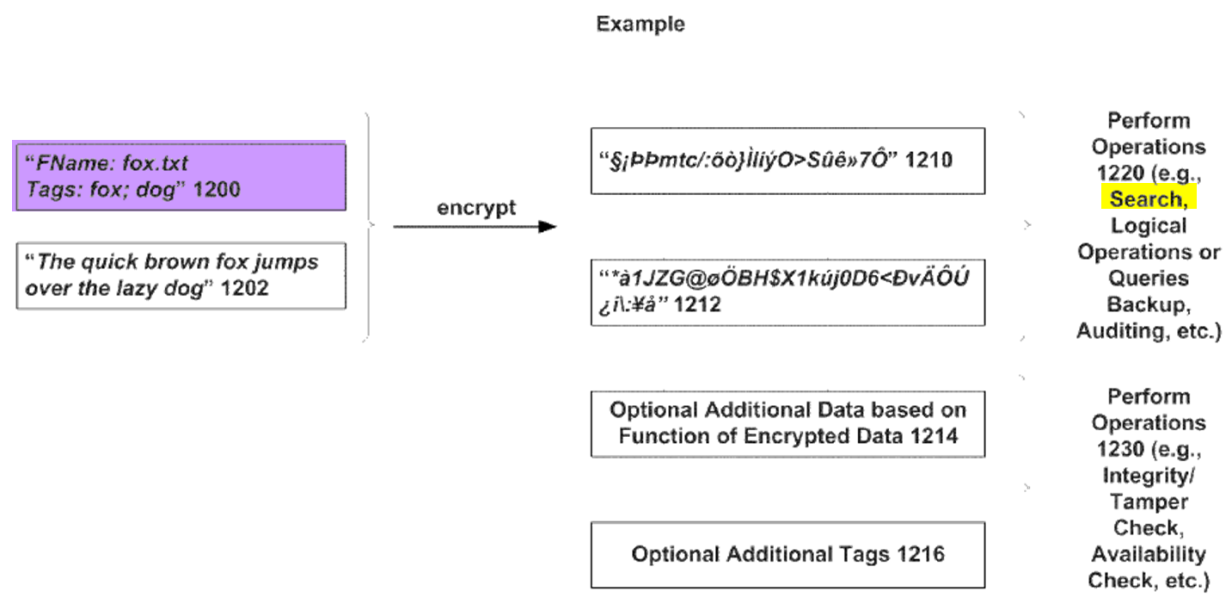


FIG. 12 (Annotated)

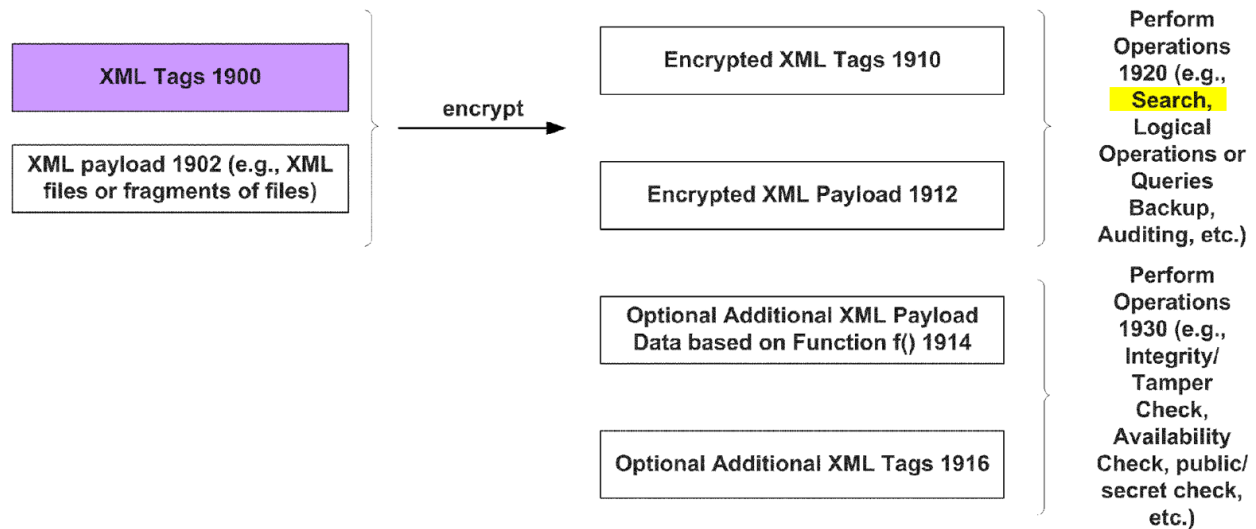


FIG. 19 (Annotated)

217. A POSITA would have found it obvious that these tags constituted namespaces for the payloads (e.g., file 1202), because, consistent with the plain and ordinary meaning of the term “namespace,” these tags were a “set of names available for naming things” such as files or segments. (Dictionary of Computer and Internet Terms, Tenth Edition (2009), Ex. 1014.)

218. To the extent that Auradkar did not disclose creating and configuring namespaces for chunks, Chiueh taught or suggested this limitation.

219. In particular, Chiueh disclosed “break[ing] a document into a plurality of pages” and “individually encrypt[ing] the document pages...[produce] to a plurality of encrypted pages.” (Chiueh, ¶20.) Chiueh’s system also created and assigned a document identification or ID (namespace) to each of the encrypted

pages. (Chiueh, ¶20.) “Each of the encrypted words in the [global search index (]GSI[)] was] mapped to a corresponding document ID, where the **document ID indicate[d] an encrypted page where a given word [could] be found.**” (Chiueh, ¶40.) Similar to Auradkar’s tag 1200, which included a filename with other tag information (Auradkar, FIG. 12), Chiueh’s document ID was “**combination of a file name and a page offset in the document**, and the file name and the page offset [were] combined and encrypted.” (Chiueh, ¶40.)

220. The Auradkar-Chiueh combination would have allowed creating document IDs that identified portions of a document (e.g., a document page, an XML fragment) and searching the encrypted files using the document IDs (as explained in Limitation 1[b][iii] below). A POSITA would have been motivated to have made this modification, because in addition to Auradkar’s existing option for searching the whole document, the modified system would also have allowed identifying and retrieving individual encrypted document portions stored in the cloud, “so that a bandwidth resource of the communication network [could] be effectively saved.” (Chiueh, ¶28.)

221. A POSITA would have reasonably expected success in incorporating Chiueh’s functionalities into Auradkar’s system because Auradkar already disclosed generating tags, which could include filenames for payloads, and using these tags to search the encrypted payloads. Additionally, creating identifiers for

portions of a document had been well-known and would have been compatible with Auradkar's system, and the combination would have yielded expected results. (See Auradkar, ¶¶125, 150, FIGS. 12, 19; Chiueh, ¶40, FIG. 2.)

g. 1[b][iii]

222. The Auradkar-Chiueh combination disclosed this limitation.

223. The '678 Patent does not describe what namespaces are or how they are **configured**, but it uses this term in the context of a search process that involves generating searchable keyword indexes ('678 Patent, 12:14-18), encrypting keywords that are used in a search query ('678 Patent, 12:42-47), and searching the encrypted index to identify any file chunks that match the encrypted search keywords ('678 Patent, 12:48-55).

224. Auradkar described a similar process for creating searchable encrypted content that included configuring tags (namespaces) that allowed searching an encrypted document based on encrypted keywords. Chiueh also described a procedure that allowed searching segments (chunks) of an encrypted document based on document IDs (namespaces). In the Auradkar-Chiueh combination, the documents IDs/tags were configured to allow search, identification, and retrieval of document segments (e.g., a document page, an XML fragment) as further described below.

225. Specifically, Auradkar disclosed creating and encrypting tags for a

payload (e.g., an XML fragment); the tags included a filename (see e.g., “FName: fox.txt” in Figure 12), were encrypted and used to perform various “operations 1220,” such as searching the encrypted files. (Auradkar, ¶¶125, 150, FIGS. 12, 19.)

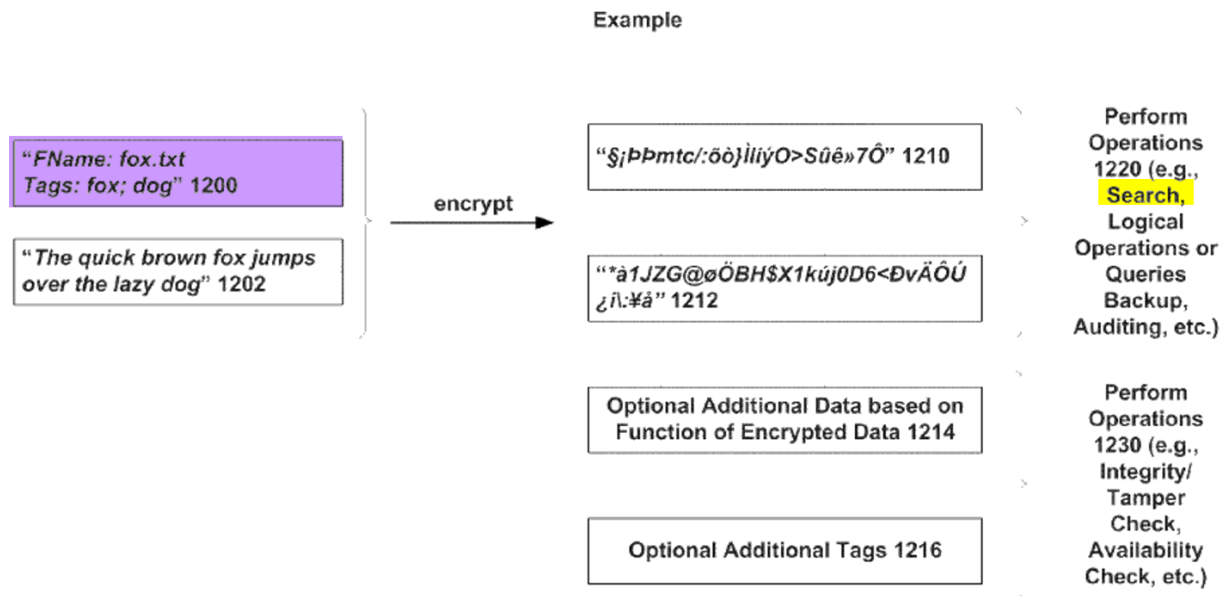


FIG. 12 (Annotated)

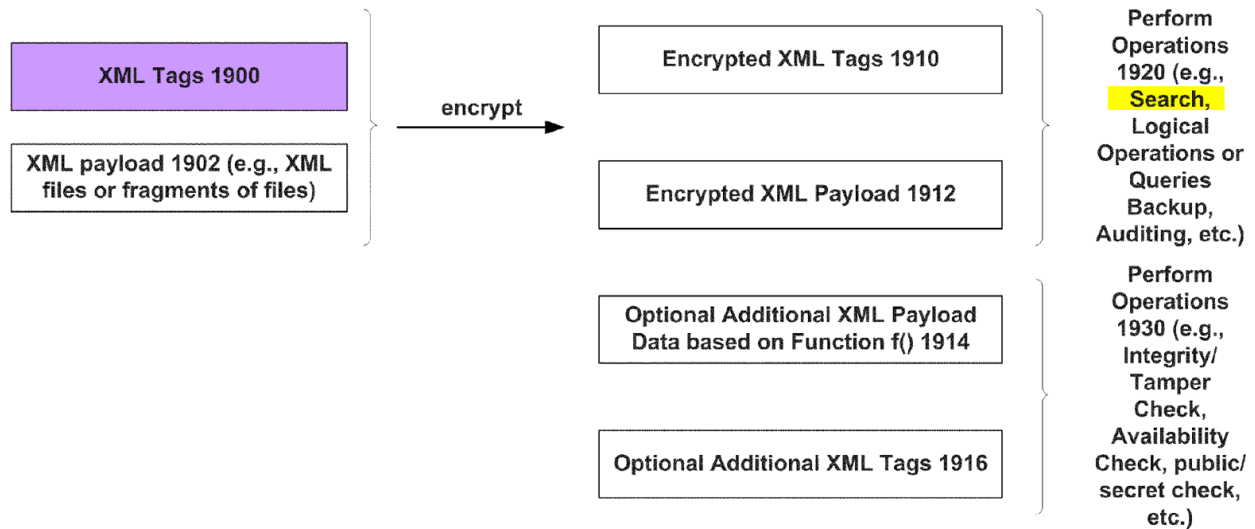


FIG. 19 (Annotated)

226. Auradkar also disclosed encrypting a **document 1500** and various keywords associated with the document to generate **encrypted document** and encrypted indexes (containing the encrypted keywords), which were uploaded to the cloud. (Auradkar, ¶130, FIG. 15.)

Technology for Confidentiality: Searchable Encryption

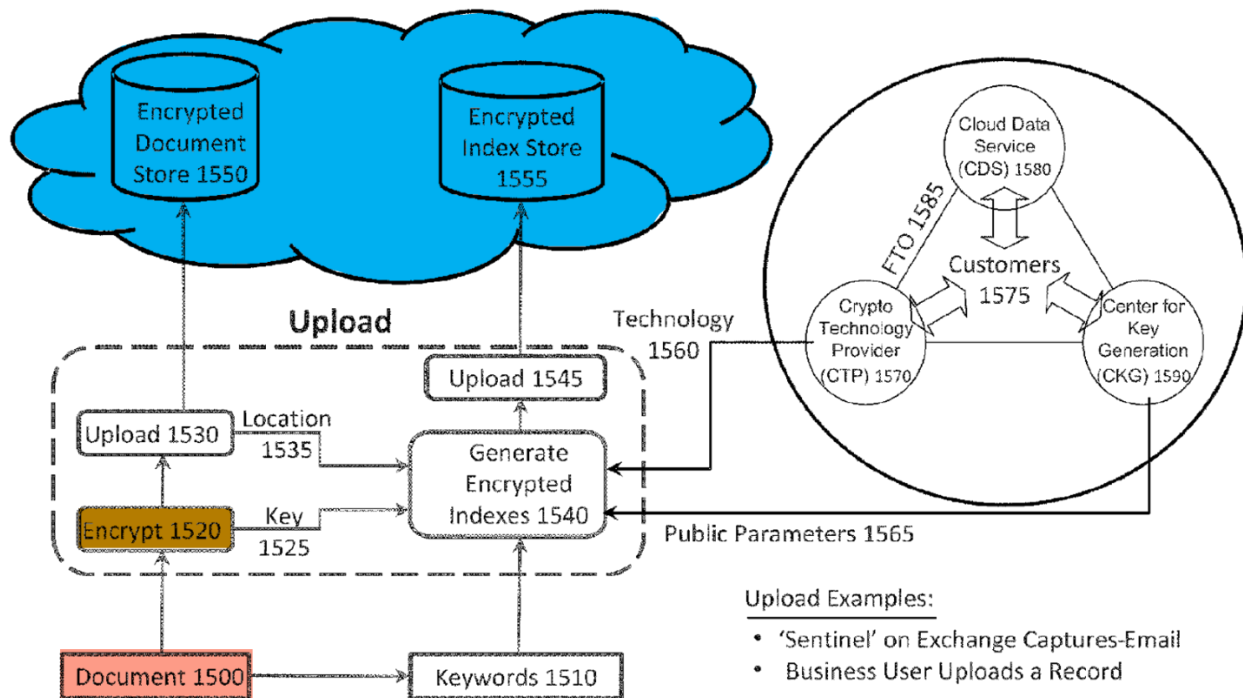


FIG. 15 (Annotated)

227. When searching the encrypted documents, Auradkar formed a search query that included encrypted search keywords (Auradkar, ¶133, FIG. 16.), and searched the encrypted indexes stored in the cloud to identify “content or records [could] be identified matching a given index” (Auradkar, ¶127).

Technology for Confidentiality: Searchable Encryption

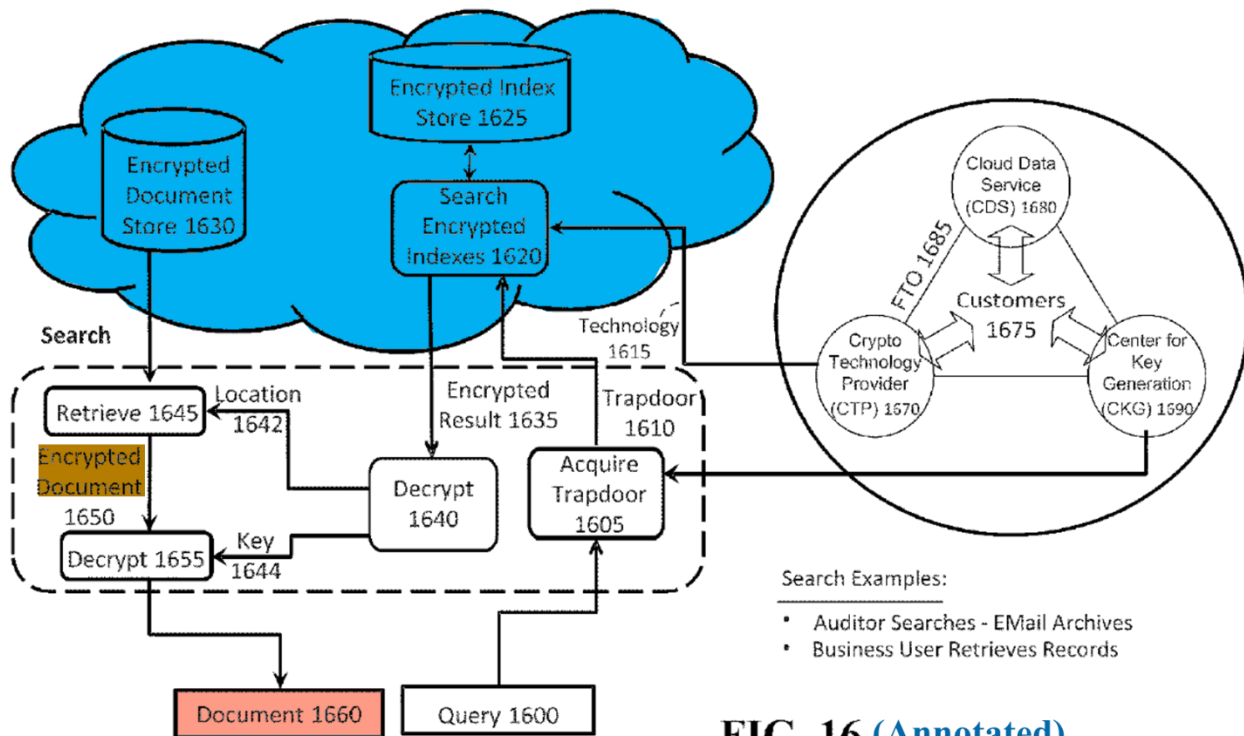


FIG. 16 (Annotated)

228. Chiueh disclosed encrypting individual document pages (chunks) to obtain corresponding encrypted pages and assigning an ID (namespace) to each of the encrypted pages. (Chiueh, ¶¶20, 40, FIG. 2; *supra* § I.D.3.f.)

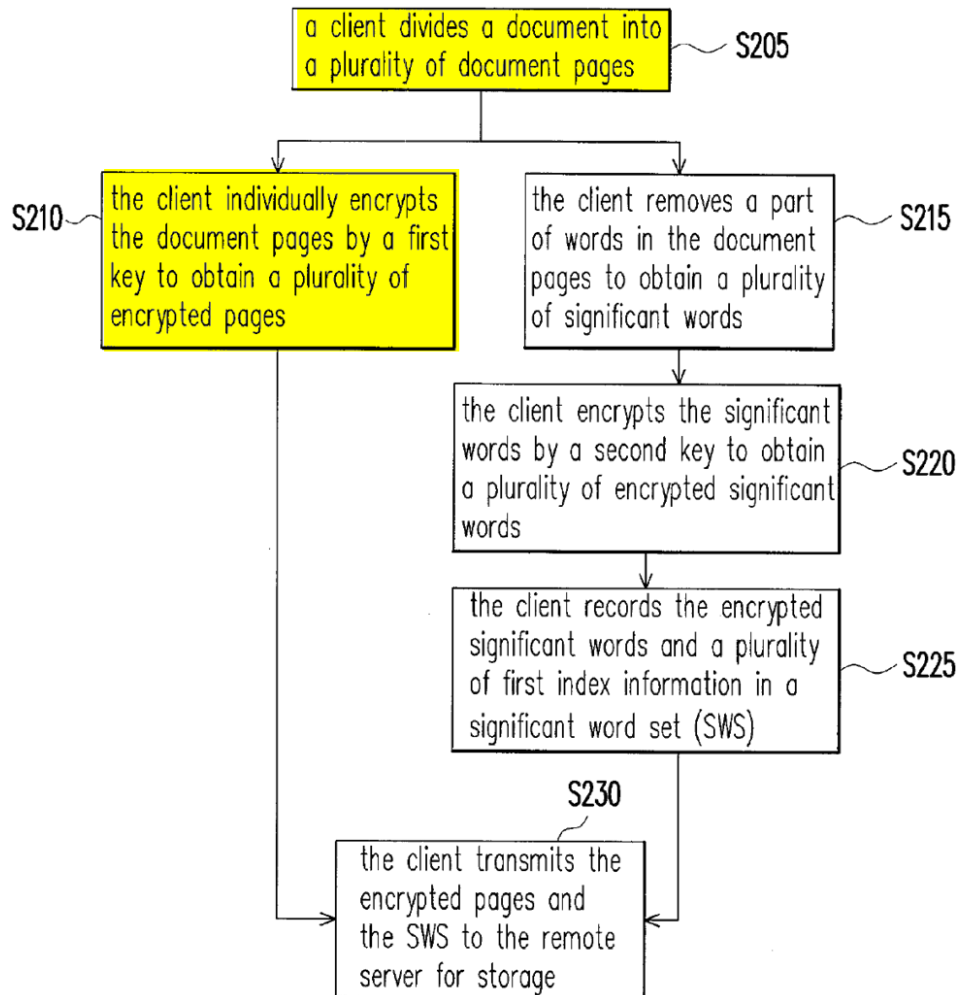


FIG. 2 (Annotated)

229. Chiueh also disclosed creating an index (e.g., a significant word set (SWS)) that mapped encrypted keywords to corresponding document ID(s) of the encrypted pages, so that the content of the encrypted pages could be searched using those keywords. (Chiueh, ¶¶21-23, FIG. 2.)

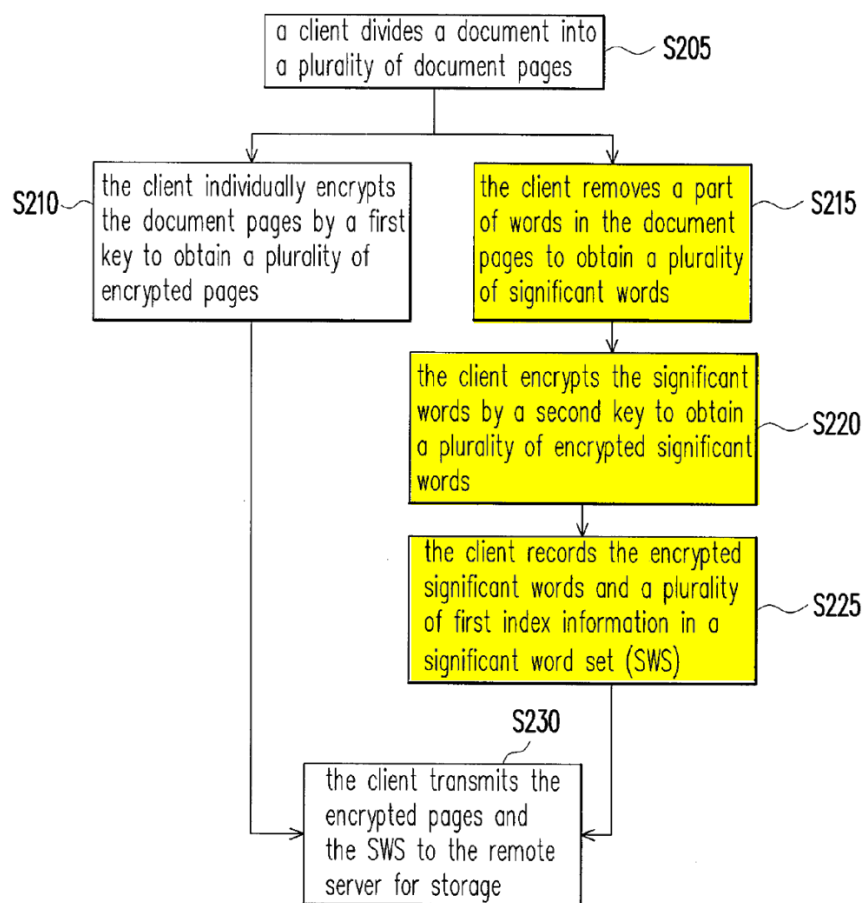


FIG. 2 (Annotated)

230. For example, the encrypted pages and index (SWS) were transmitted to a remote server 12, which then stored the SWS in a global search index (GSI) that included a key field that “record[ed] the encrypted significant words” and a value field that “record[ed] the first **index information.**” (Chiueh, ¶24.) This “index information” “indicate[d] a page in the encrypted pages where the encrypted significant word [came] from,” such as document IDs. (Chiueh, ¶¶24, 40.)

231. Moreover, similar to the '678 Patent's search process (explained

above), Chiueh's search process involved "encrypt[ing] the keyword KW...to obtain an encrypted keyword" and "search[ing] the SWS according to the encrypted keyword, i.e., search[ing] the GSI to obtain the **index information** corresponding to the encrypted keyword." (Chiueh, ¶25, FIG. 3.) And "[i]f the remote server 120 [found] the encrypted [keyword] from the SWS, the remote server 120 obtain[ed] at least one target encrypted page from the encrypted pages stored in the database according to the **index information** corresponding to the encrypted keyword...[and] transmit[ted] the target encrypted page to the client 110." (Chiueh, ¶25, FIG. 3.)

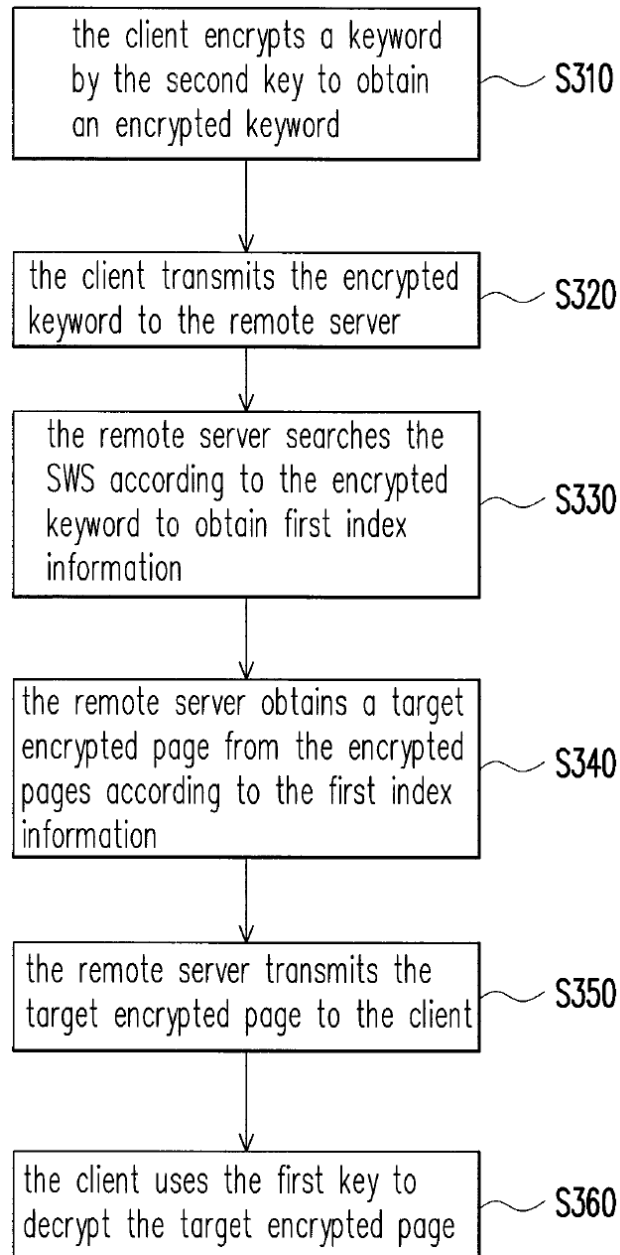


FIG. 3

232. A POSITA would have found it obvious that Chiueh's system configured document IDs (namespaces) of the encrypted pages (chunks) because it mapped these IDs (indicated in the index information) to keywords that appeared

in the corresponding pages in a global search index (GSI), such that the page content was encrypted in a manner that made it searchable through these IDs. Indeed, consistent with the description of the '678 Patent, Chiueh's system enabled users to search encrypted page content using a search keyword to identify the ID that was mapped to the keyword, and to retrieve the corresponding page based on the identified ID. (Chiueh, ¶¶23-24, 40, FIG. 3.)

233. A POSITA would have found it obvious to add Chiueh's functionality of configuring namespaces (similar to Auradkar's tags) that allowed search and retrieval of individual pages into Auradkar's system based at least on Auradkar's suggestion for searching encrypted XML fragments. In addition, doing so would have allowed another option for retrieving individual pages, instead of the whole document, "so that a bandwidth resource of the communication network [could] be effectively saved." (Chiueh, ¶28.) As explained above, a POSITA would have reasonably expected success in incorporating Chiueh's functionalities into Auradkar's system. (*See supra* § I.D.3.f.)

h. Claim 1[c]

234. Auradkar disclosed uploading and storing the **searchable encrypted file** on one or more cloud platforms (e.g., the **cloud storage** in FIG. 15) based on the policy because Auradkar described that its policy could specify "how data could be stored, **where it could be stored**, how long it could be stored...."

(Auradkar, ¶¶96, 130 (“To perform an upload, document 1500 [was] encrypted 1520 and uploaded 1530 to **the cloud** into an encrypted document store 1550.”)), FIG. 15.)

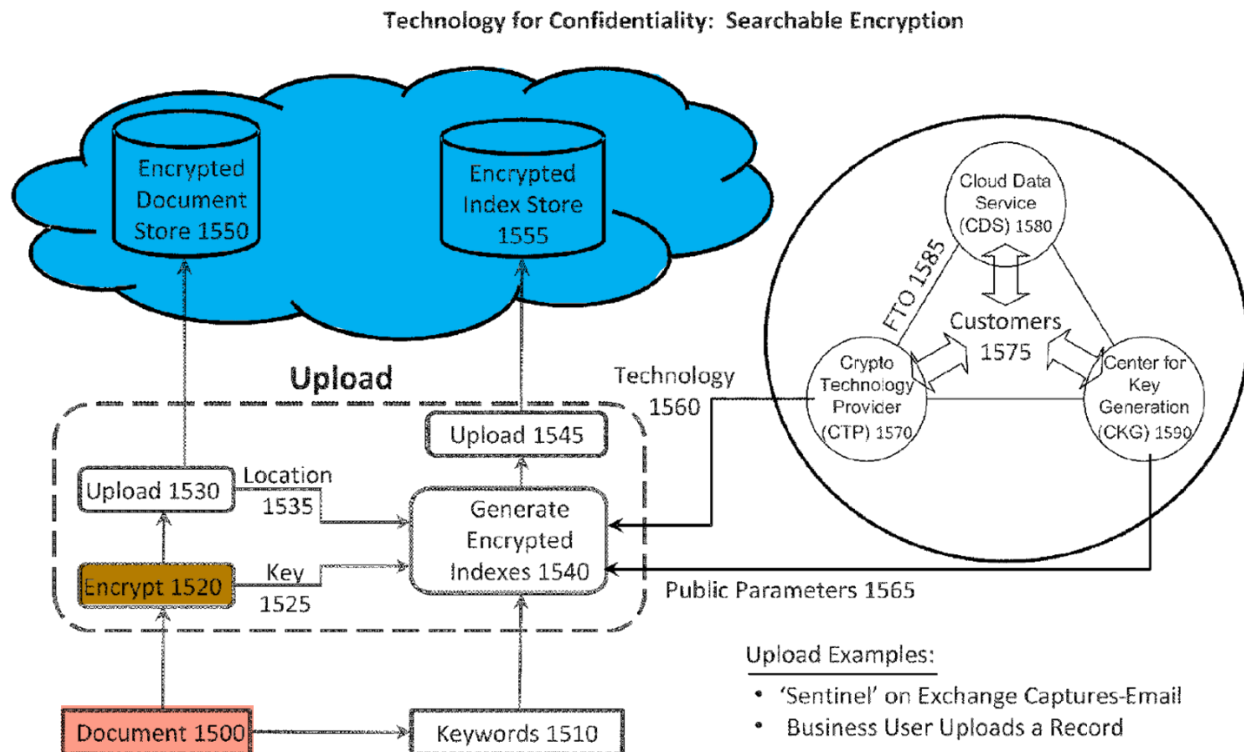


FIG. 15 (Annotated)

235. Auradkar also disclosed “storing the encrypted data by one or more CSPs 2870, such as CSPs 2872, 2874, 2876, or 2878.” (Auradkar, ¶189, FIG. 28.)

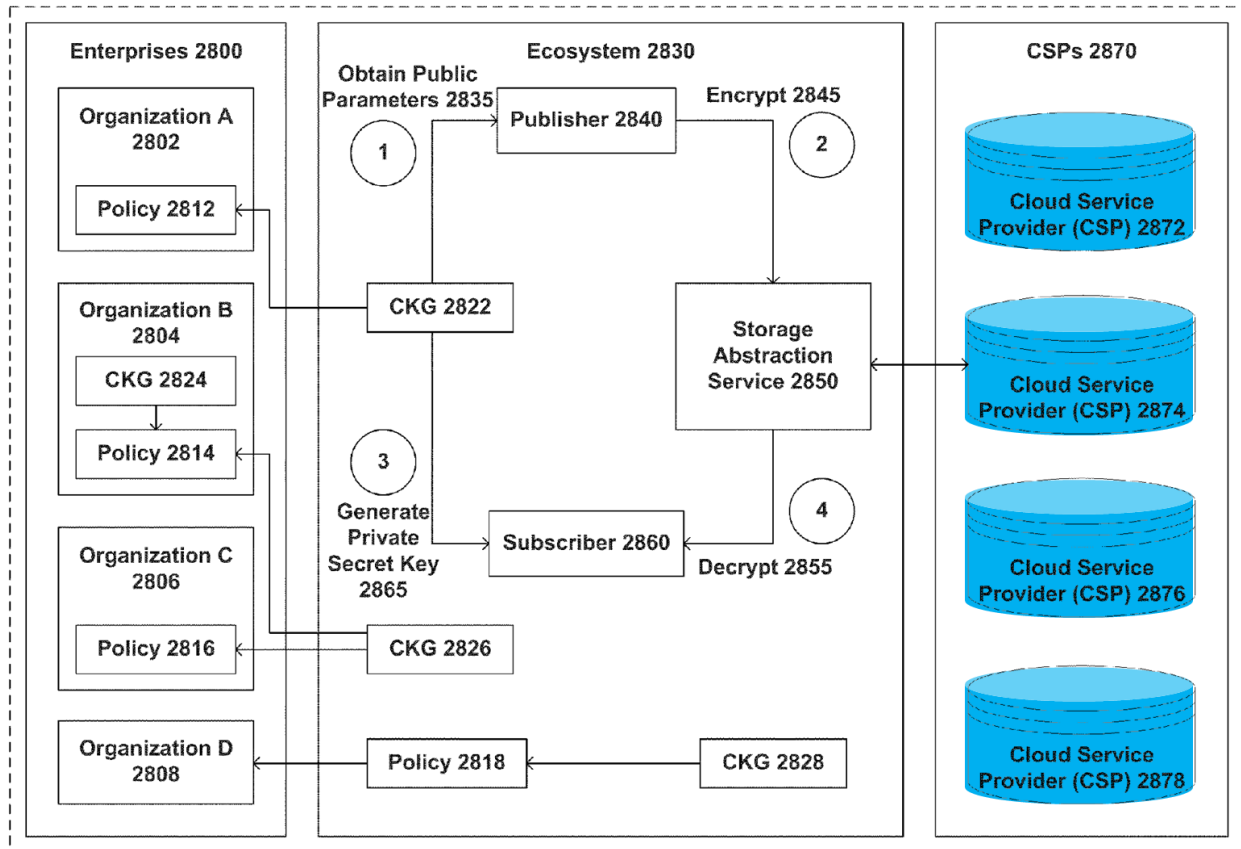


FIG. 28 (Annotated)

236. As explained above, a POSITA would have found it obvious to use Chambers’ gateway device to perform the functions described in Auradkar, such as storing the searchable encryption file on one or more cloud platforms based on the policy. (*Supra* § I.D.3.b.)

i. Claim 1[d]

237. Auradkar disclosed managing access to the searchable encrypted file by the one or more users based on policy.

238. Specifically, Auradkar disclosed that “[w]ith policy and enforcement within the control of users or an enterprise, data and corresponding [were]

encrypted and accessible only with specific capabilities granted to a user.”

(Auradkar, ¶112.) For example, “when **users 508** of data 512 attempt to access such data 512, various crypto access controls 514 **guided by...enterprise access policy 530** protect[ed] the data 512 from unwarranted access by users 508.”

(Auradkar, ¶117, FIG. 5.)

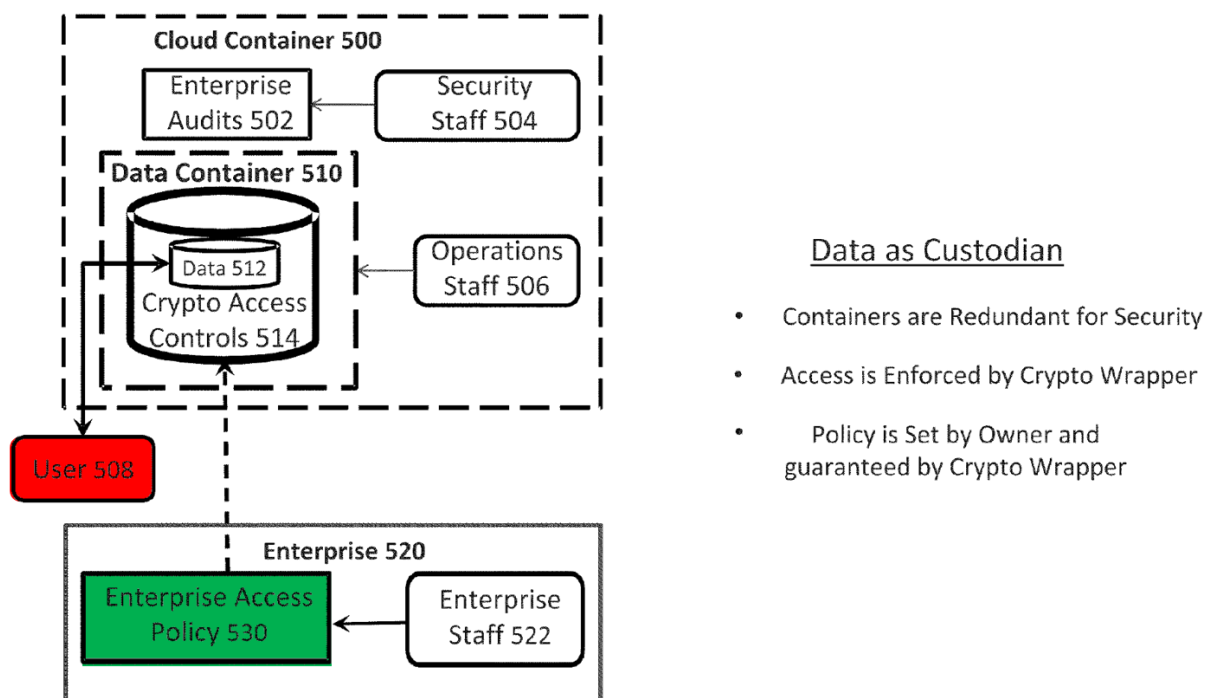


FIG. 5 (Annotated)

239. (See also *id.*, (“Enterprise access policy 530 [could] affect how data 512 stored in a data container 510 of a cloud container 500 [could] be **accessed, manipulated, retrieved, searched...**”), ¶113 (“[E]nterprise 220 ‘own[ed]’ its policy 224 and enforcement 222 of the policy 224 with respect to users 226 and their use of system resources of the enterprise 220 as well as with respect to

external users 230 (e.g., mobile workers). With data as its own custodian, the actual data and/or logs 205 [could] be separated from policy 224 and enforcement 222 by storing the data in cloud 200, however, the operations staff 210 of the cloud 200 [were] unable to view, modify, tamper or delete the data and/or logs 205 without detection.”)

4. Claim 16

240. In my opinion, Auradkar, Chambers, and Chiueh rendered obvious claim 16.

a. Claim 16[pre]

241. Auradkar disclosed all aspects of the preamble in claim 16.

242. In particular, Auradkar disclosed a secure cloud storage system for encrypting files and storing encrypted files to a cloud storage, as shown in Figures 15, 53, and 54. (*See, e.g.*, Auradkar, Abstract (“A digital escrow pattern and trustworthy platform [was] provided for data services including mathematical transformation techniques, such as searchable encryption techniques, for obscuring data stored at remote site or in a cloud service, distributing trust across multiple entities to avoid a single point of data compromise.”), FIGS. 15, 53-54.)

Technology for Confidentiality: Searchable Encryption

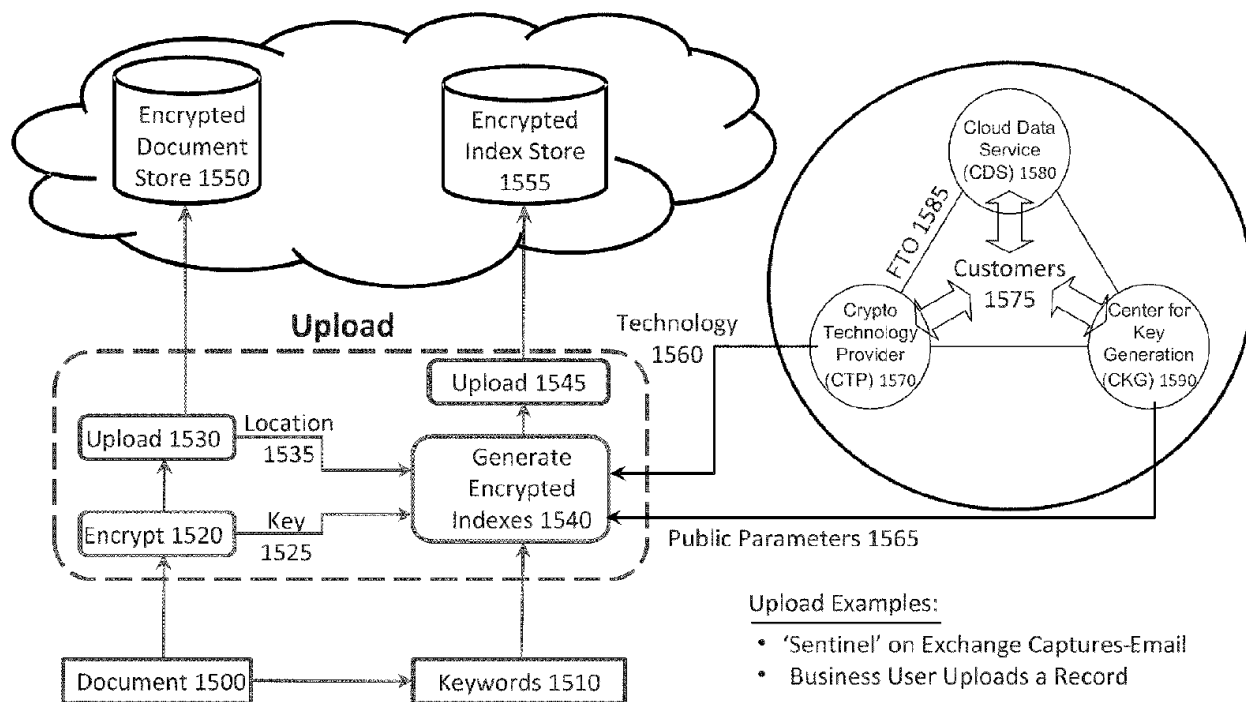


FIG. 15

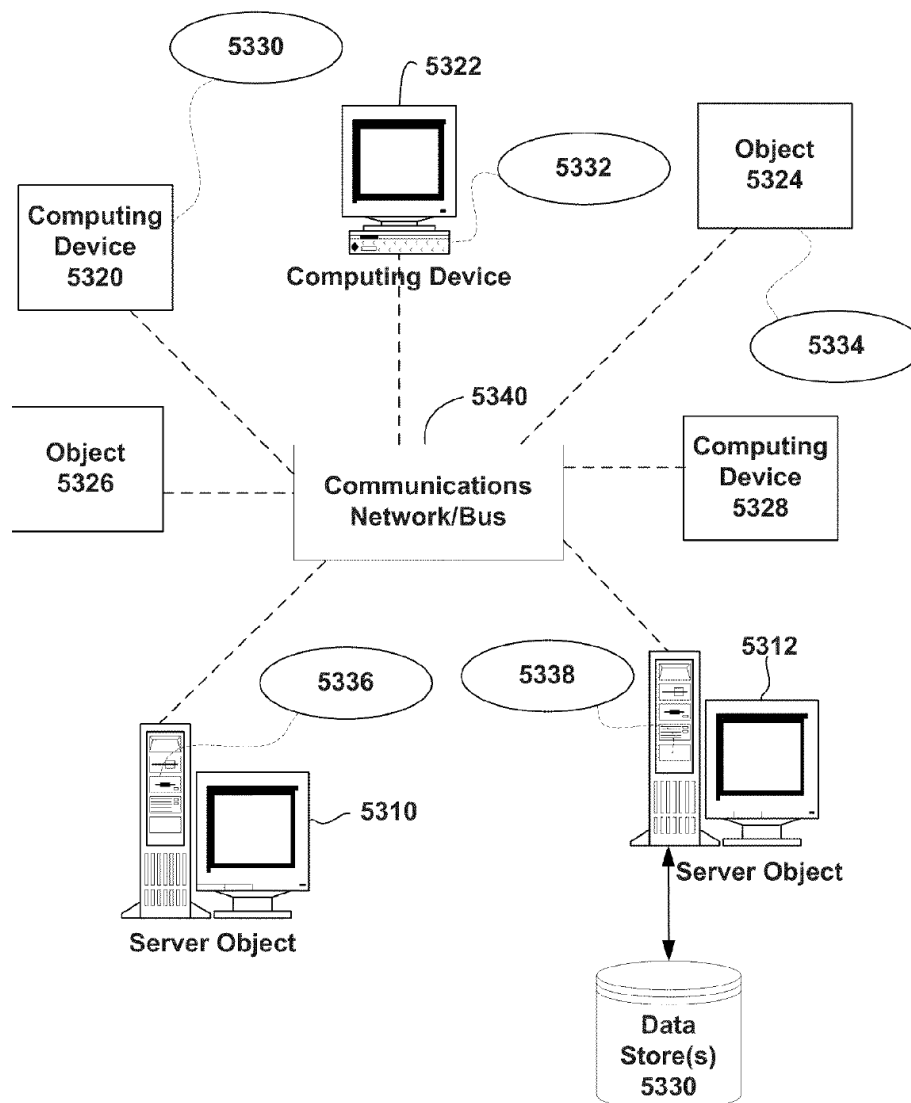
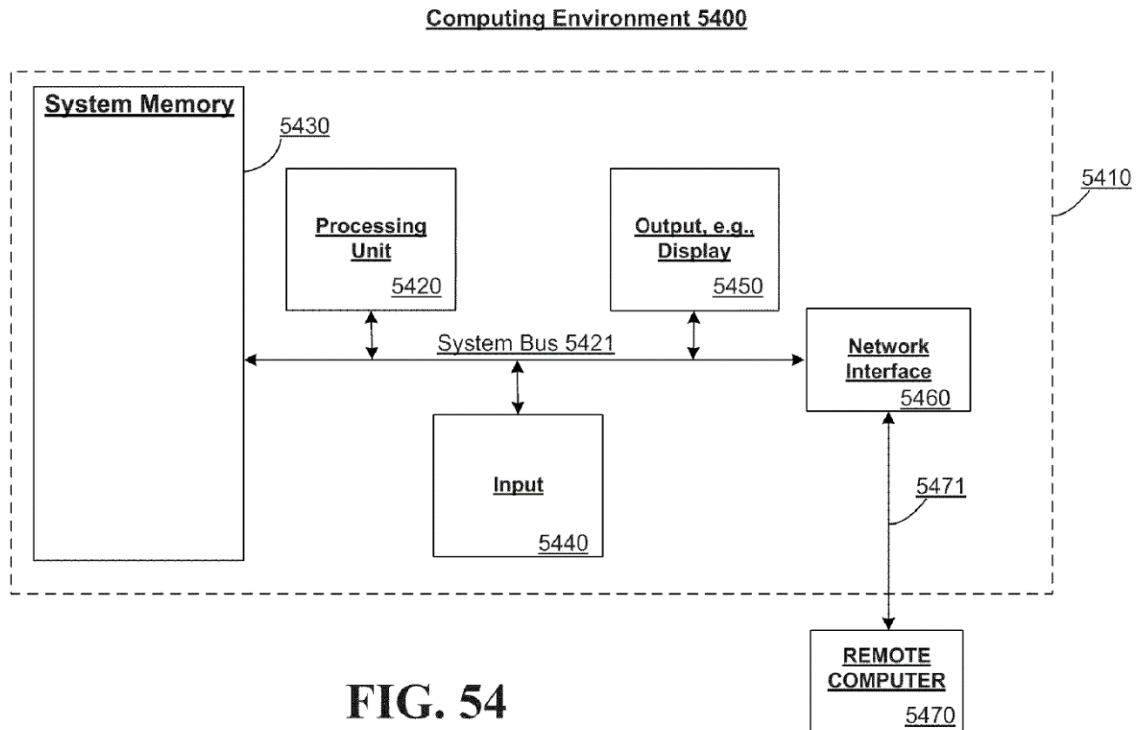
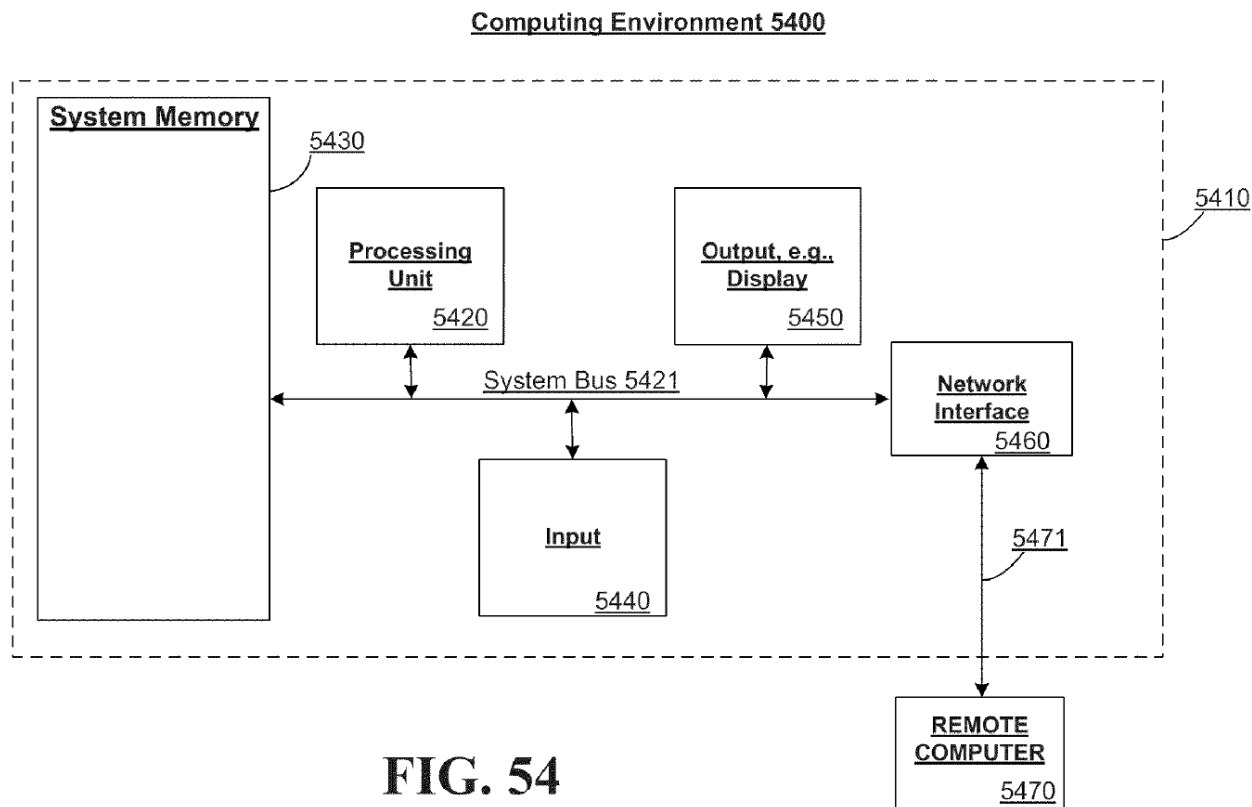


FIG. 53



b. Claim 16[a]: “one or more processors;”

243. Auradkar described that the embodiments disclosed therein could be implemented with a computing system 5410 illustrated in FIG. 54, which included a processing unit 5420. (Auradkar, ¶¶256-257, FIG. 54.) A POSITA would have understood that Auradkar’s processing unit 5420 constituted a processor.



244. A POSITA would have understood that Auradkar’s processing unit 5420 constituted a processor.

c. Claim 16[b]: “a communication interface mechanism;”

245. Auradkar disclosed that its computing system 5410 (system) included a network interface 5460, which a POSITA would have found obvious to constitute the claimed “communication interface mechanism” because Auradkar’s network interface 5460 was used to communicate with remote computer 5470 via network 5471 (e.g., local area network (LAN)). (Auradkar, ¶260, FIG. 54.)

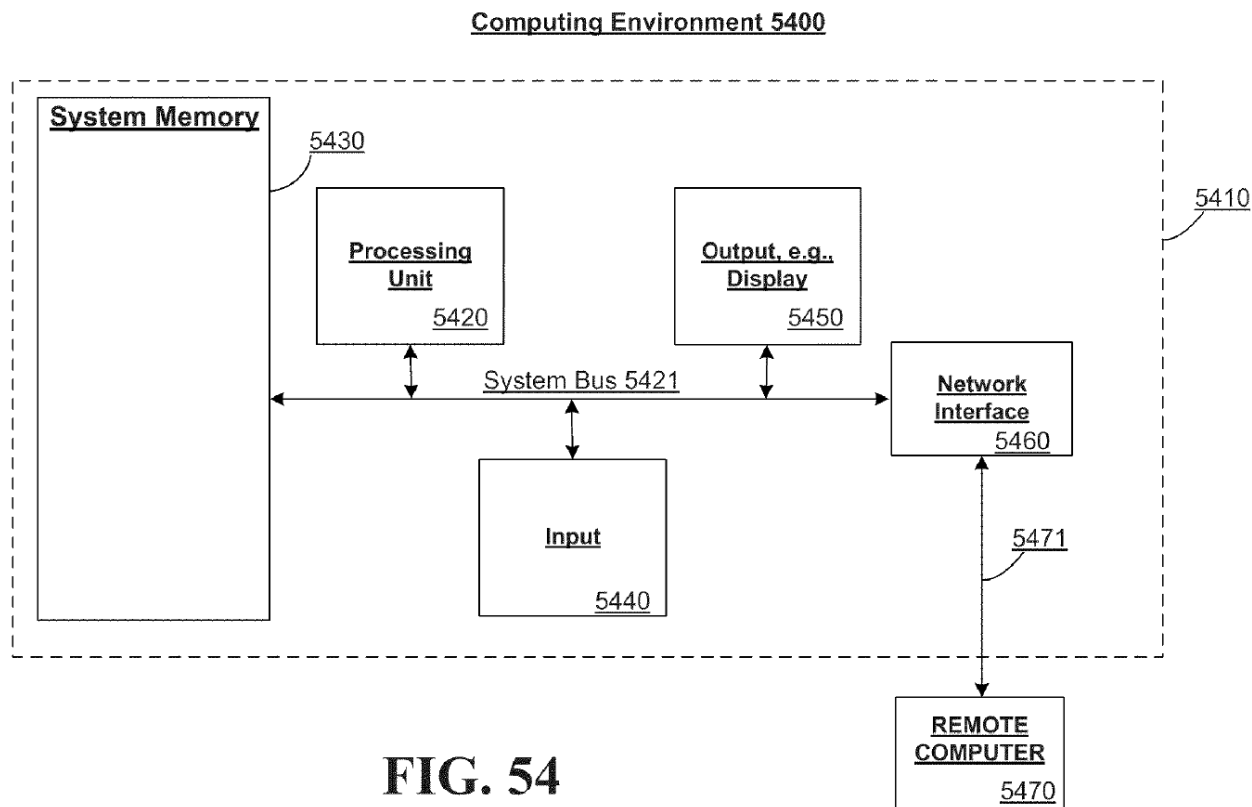
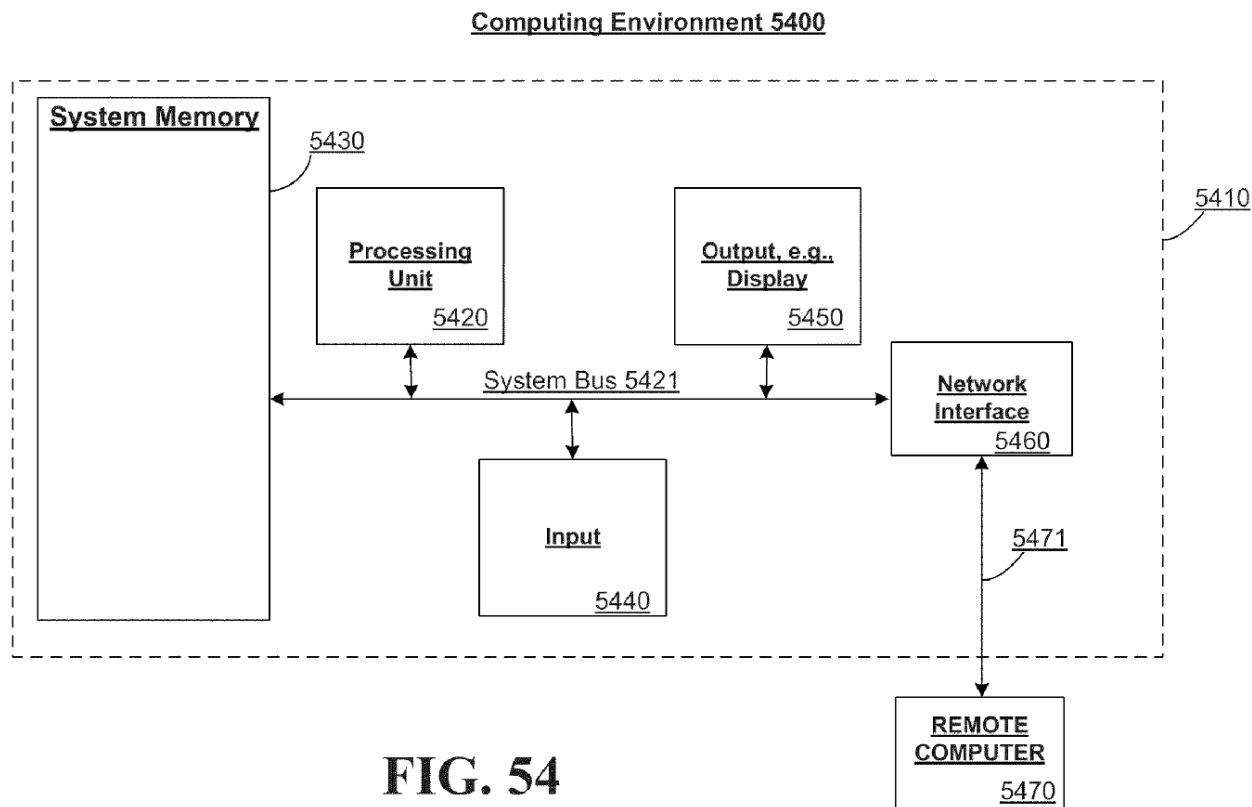


FIG. 54

- d. **Claim 16[c]: “one or more internal data storage devices operatively coupled to the one or more processors and storing:”**

246. Auradkar disclosed that its computing system 5410 (system) included system memory 5430 (internal data storage device) “to the processing unit 5420” (processor). (Auradkar, ¶258, FIG. 54.)



247. Auradkar’s system memory 5430 included “computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) and/or random access memory (RAM)” and stored “application programs, other program modules, and program data.” (*Id.*) A POSITA would have found it obvious that the “application programs” and “programs modules” stored in system memory 5430 could be implemented for performing the functions disclosed in Auradkar, including the claimed functions (as explained below), because Auradkar contemplated that its computing system 5410 and its components could be used for implementing the disclosed embodiments. (Auradkar, ¶¶255-256.)

- e. **Claim 16[c][i]: “a policy assignment module, which, when executed by the one or more processors, implements a policy selected from a group of policies for a user,”**

248. Auradkar and Chambers rendered obvious this limitation.

a policy assignment module [and other claimed “encryption,” “storage,” “management,” and “mediation” modules], which, when executed by the one or more processors:³

249. Auradkar disclosed that its system memory 5430 stored various “application programs” and “programs modules” that could be “executed by one or more computers,” such as computing system 5410, to implement the disclosed embodiments. (Auradkar, ¶¶255, 258, FIG. 54.) A POSITA would have found it obvious that to implement these prior art software functions, processing unit 5420 of computing system 5410 would have executed corresponding modules, because executing modules by a processing unit like Auradkar’s to carry out these functions had been well known. (See, e.g., Auradkar, ¶255; Chambers, ¶¶27, 77-78; Cidon, ¶50.)

250. Auradkar and Chambers disclosed “implement[ing] a policy selected from a group of policies for a user.” (See *supra* § I.D.3.b.)

251. In the Auradkar-Chambers combination, a POSITA would have added Chambers’ access control module and its functionalities for selecting an ACP from

³ The “module” limitations in claim 16 are addressed together in this limitation.

its ACPs and automatically assigning and implanting the selected policy into Auradkar's system (as illustrated in modified Figure 15 of Auradkar below) to allow Auradkar's system to automatically assign and implementing policies (Chambers, ¶¶21, 46, FIG. 4A (illustrating "Group policy" for "staff" and "iPad group")) to users.

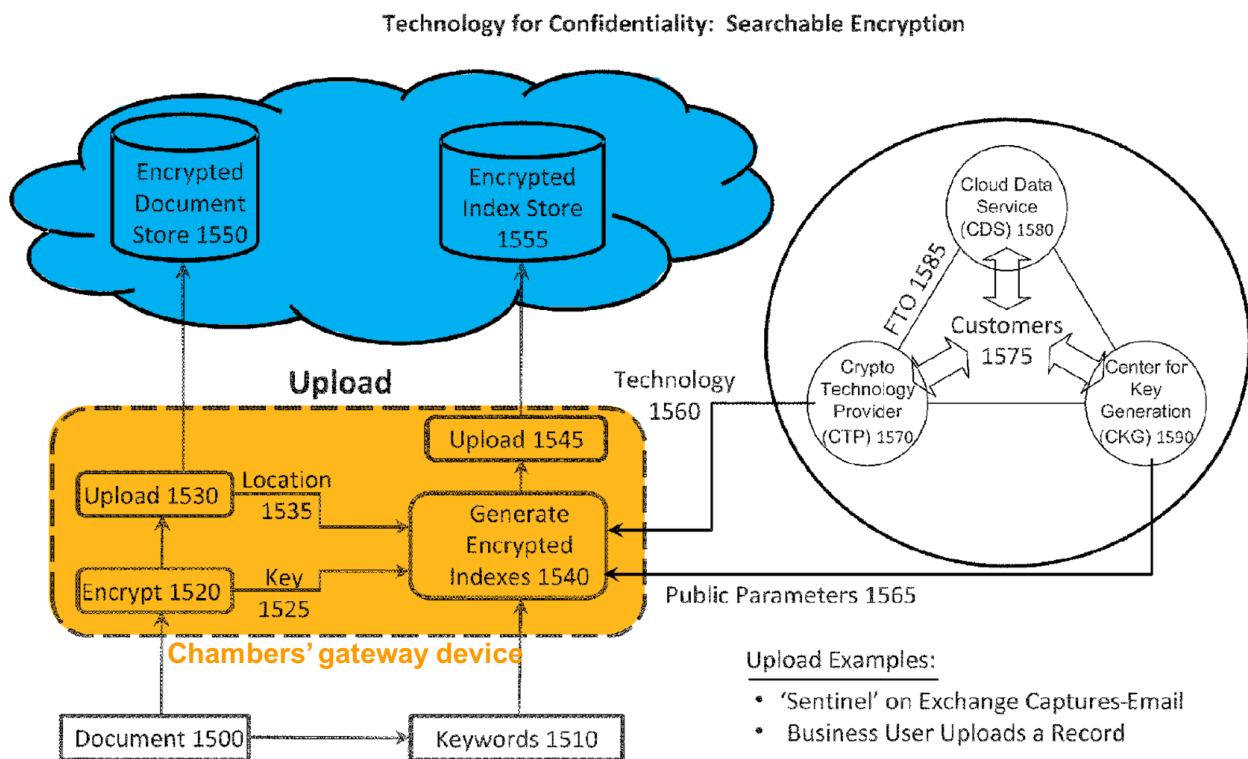


FIG. 15 (Annotated)

252. As I explained above, a POSITA would have been motivated to make this modification with a reasonable expectation of success. (*See supra* § I.D.3.b.)

f. Claim 16[c][i][A]: “wherein the policy defines and manages access to and processing of a file to be uploaded on a cloud,”

253. Auradkar's policy defined and managed access to and processing of a

file to be uploaded on a cloud. (See § I.D.3.b (Ground 4, limitation 1[a][i].)

g. Claim 16[c][i][B]: “wherein the policy defines access rights of the user;”

254. Auradkar’s policy defined access rights of the user. (See § I.D.3.c (Ground 4, limitation 1[a][ii]).)

h. Claim 16[c][ii]: “an encryption module, which, when executed by the one or more processors:”

255. As explained above, a POSITA would have found it obvious that Auradkar’s system would have included various modules, including the claimed “encryption module,” when executed by its processing unit (processor) to perform, among others, the claimed functions. (See *supra* § I.D.4.e.)

i. Claim 16[c][ii][A]: “using cryptographic key information defined by the policy, produc[ing] a searchable encrypted file by encrypting content of the file to be stored across one or more cloud platforms;”

256. Auradkar rendered obvious this limitation. (See *supra* § I.D.3.d.)

j. Claim 16[c][ii][B]: “divid[ing] the file into a plurality of chunks;”

257. Auradkar disclosed obvious this limitation. (See *supra* § I.D.3.e.)

k. Claim 16[c][ii][C]: “create[ing] namespaces for one or more of the plurality of chunks; and”

258. Auradkar or the Auradkar-Chiueh combination suggested this limitation. (See *supra* § I.D.3.fError! Reference source not found..)

l. Claim 16[c][ii][D]: “configure[ing] the

namespaces of the one or more chunks such that content of the file is encrypted in a manner that makes it searchable;"

259. Auradkar or the Auradkar-Chiueh combination rendered obvious this limitation. (*See supra* §I.D.3.g.)

- m. Claim 16[c][iii]: “a storage module, which, when executed by the one or more processors, stores the searchable encrypted file on the one or more cloud platforms based on the policy; and”**

260. As I explained above, a POSITA would have found it obvious that Auradkar would have included various modules, including the claimed “storage module,” when executed by its processing unit (processor) to perform, among others, the claimed “storage” function, which was suggested by Auradkar. (*See supra* §§ I.D.4.e, I.D.3.h.)

- n. Claim 16[c][iv]: “a management module, which, when executed by the one or more processors manages and controls access to the searchable encrypted file by the user based on the policy.”**

261. As explained above, a POSITA would have found it obvious that Auradkar would have included various modules, including the claimed “management module,” when executed by its processing unit (processor) to perform various functions, including the claimed function, which was disclosed by Auradkar. (*See supra* § I.D.4.e, I.D.3.i.)

5. Claims 2/17

262. In my opinion, Auradkar rendered claims 2 and 17 obvious.

263. Specifically, Auradkar disclosed “enterprise 220 ‘owns’ its policy 224 and enforcement 222 of the policy 224 with respect to...external users 230 (e.g., **mobile workers**)” and the policy could “affect how data 512 stored in a data container 510 of a cloud container 500 [could] be **accessed**, manipulated, **retrieved**, searched.” (Auradkar, ¶¶113, 117.)

264. Auradkar also described that subscribers, which could be mobile workers, “**download**[ed]...the data based on the dynamically definable capabilities granted by owner/publisher,” such as enterprise 220. (Auradkar, ¶183.)

265. A POSITA would have found it obvious, from these teachings of Auradkar, that the owner of the data stored in the cloud (e.g., enterprise 220) could configure and implement, via Auradkar’s system, a policy for its mobile workers that allowed the workers to download the searchable encrypted data (*see supra* § I.D.3.d (explaining users could retrieve encrypted searchable files from the cloud)) to local devices, such as “PDAs, audio/video devices, mobile phones, MP3 players, laptops” associated with the mobile workers, because Auradkar already contemplated that users could access and retrieve encrypted data stored on the cloud according to policies, and the users (e.g., subscribers) could download encrypted data. (Auradkar, ¶¶248, 112.)

6. Claim 3

266. Auradkar rendered this claim obvious because it disclosed mobile workers could download the encrypted files, described various computing devices, such as “PDAs, audio/video devices, mobile phones, MP3 players, laptops.” (Auradkar, ¶248, FIG. 53.) (See *supra* §I.D.6.)

7. Claim 8

267. In my opinion, Auradkar disclosed claim 8 because it described that its file **encryption** occurred outside of the **cloud**, which did not “[s]ee” encrypted records stored therein. (Auradkar, FIG. 14, ¶¶127-128.)

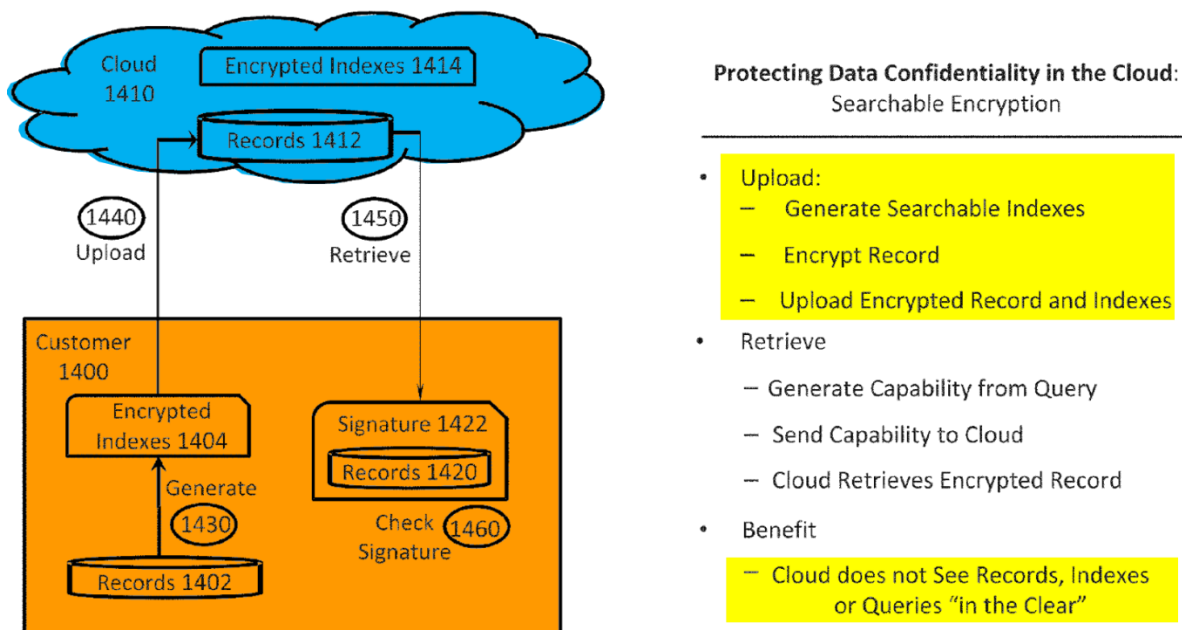


FIG. 14 (Annotated)

268. Indeed, Auradkar contemplated that a cloud service provider (CSP) to which encrypted data were stored could be “another entity that [was] not fully

trusted.” (Auradkar, ¶138; *see also* Auradkar, ¶75 (explaining that cloud data service (CDS) could be “provided by any storage provider”).)

8. Claims 9/23

269. In my opinion, Auradkar disclosed claims 9/23 because it described that “enterprise 220 ‘owns’ its policy 224 and enforcement 222 of the policy 224 with respect to **users** 226 and **their** use of system resources of the enterprise 220 as well as with respect to external **users** 230 (e.g., mobile **workers**). (Auradkar, ¶¶113, 117 (“Enterprise access policy 530 [could] affect how data 512 stored in a data container 510 of a cloud container 500 can be accessed, manipulated, retrieved, searched, etc. Accordingly, when **users** 508 of data 512 attempt to access such data 512, various crypto access controls 514 guided by...enterprise access policy 530 protect the data 512 from unwarranted access by **users** 508. Different enterprise access policy 530 can be reflected by the crypto access controls 514 of data container 510 to **apply to different accessing entities**...to ensure that visibility [was] restricted to **those** to whom access should be enabled.”); Chambers, FIG. 4A (illustrating “Group policy” for “staff” and “iPad group”).)

9. Claims 10/24

270. In my opinion, Auradkar disclosed claims 10 and 24 because its policy could specify “how data could be stored, where it could be stored, how long it could be stored” (Auradkar, ¶96), how a file was encrypted (processed) (e.g.,

specifying “medical data must be backed up once a day using AES 256 encryption; *id.*, ¶96), and how a file was accessed (e.g., “Top Secret content cannot be accessed from a wireless or remote device”; *id.*, ¶160.

10. Claims 11/25

271. In my opinion, Auradkar disclosed claims 11 and 25 because it disclosed “storing the encrypted data by one or more CSPs 2870, such as CSPs 2872, 2874, 2876, or 2878.” (Auradkar, ¶189, FIG. 28.)

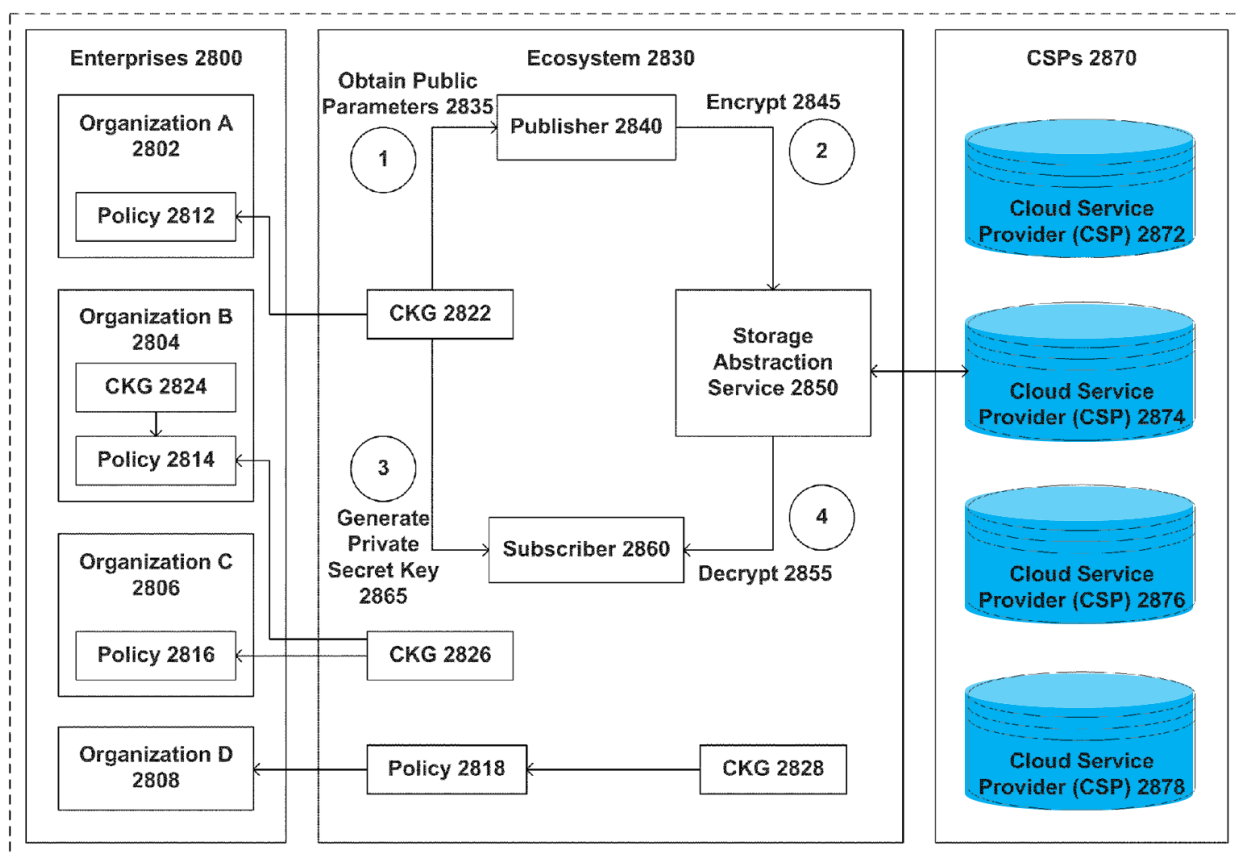


FIG. 28 (Annotated)

272. (See also *id.*, ¶230 (describing data (which could be an encrypted file) were “automatically...distributed across a set of data stores, which [could] be of

the same type, or as shown in FIG. 52, containers of different types 5212, 5214, . . . , 5216”), FIG. 52.), ¶19 (explaining that cloud data servers were “provided in a way that distributes trust across multiple entities to avoid a single point of data compromise”).)

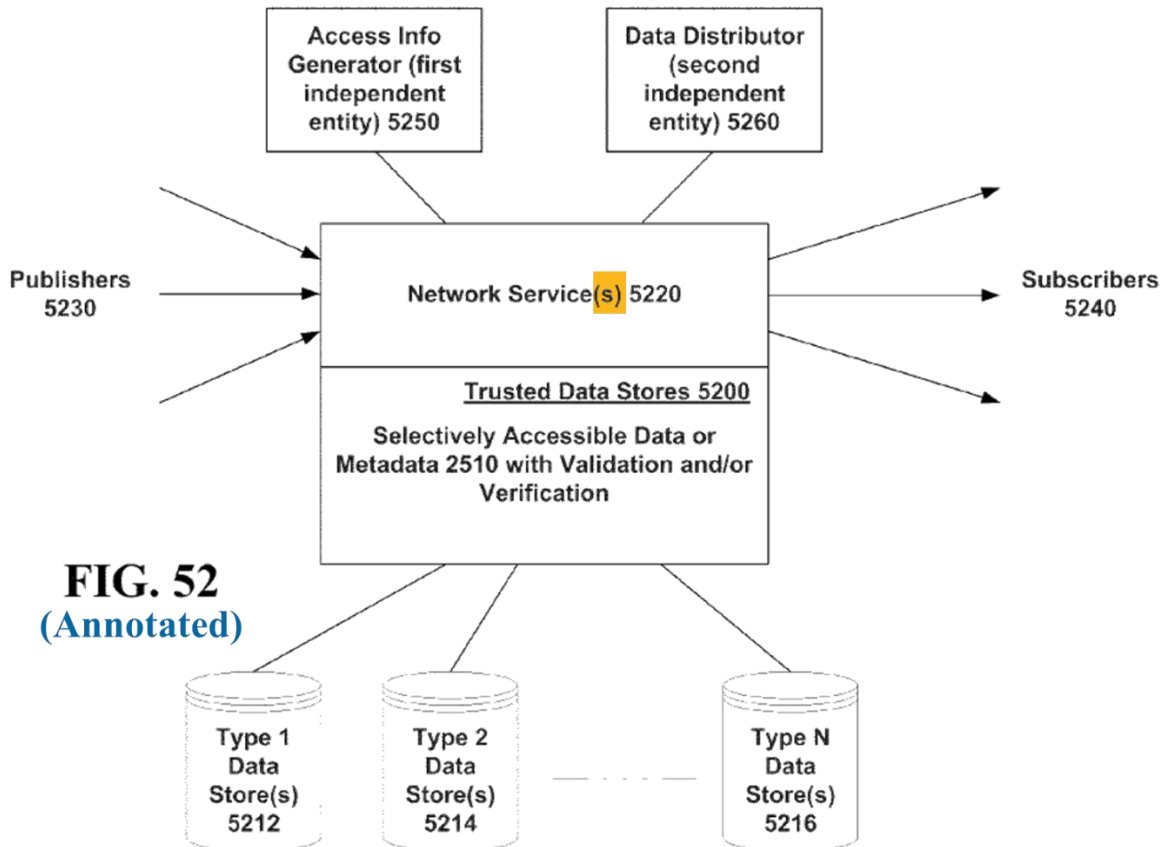


FIG. 52
(Annotated)

11. Claim 12

273. In my opinion, Auradkar disclosed claim 12 because Auradkar’s system could upload and store searchable encrypted data to a cloud data service (CDS) that was “provided by **any** storage provider” (Auradkar, ¶175), “thereby avoiding vendor lock-in” (’678 Patent, 13:54-59).

274. For example, Auradkar described a **storage abstraction service**

(SAS), separate from **CSPs** 2870, stored encrypted data to these CSPs, “hid[ing] storage semantics in connection with storing the encrypted data by” CSPs 2870.

(Auradkar, ¶189, FIG. 28.)

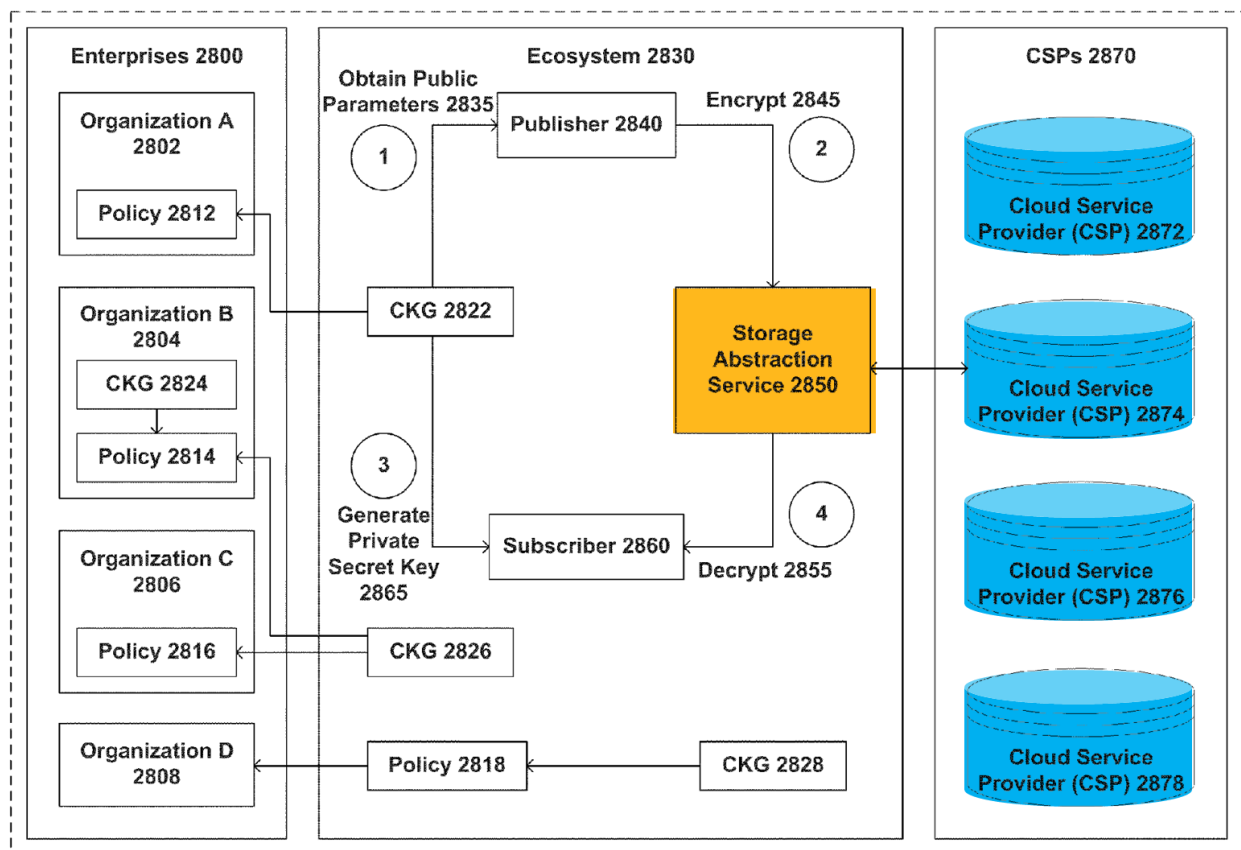


FIG. 28 (Annotated)

275. Auradkar explained that its SAS used storage transfer format (STF), “a standard interchange format that [could] be used for exchanging encrypted data and metadata **across repositories**,” such as private cloud store 3000, SQL data store 3002, simple storage web service store 3004. (Auradkar, ¶193, FIG. 30.)

STF is a Standard Interchange Format for Exchanging Encrypted Data and Metadata Across Repositories

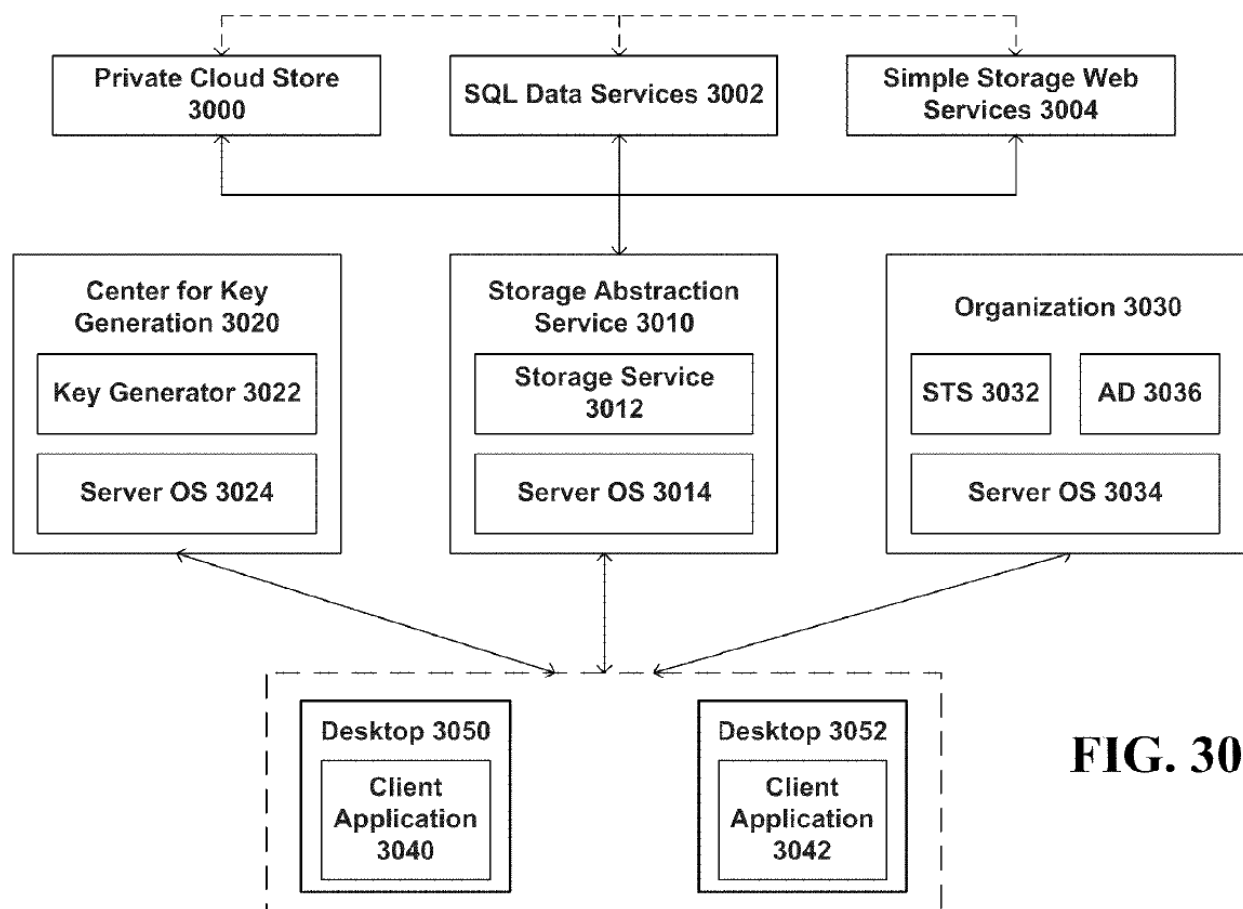


FIG. 30

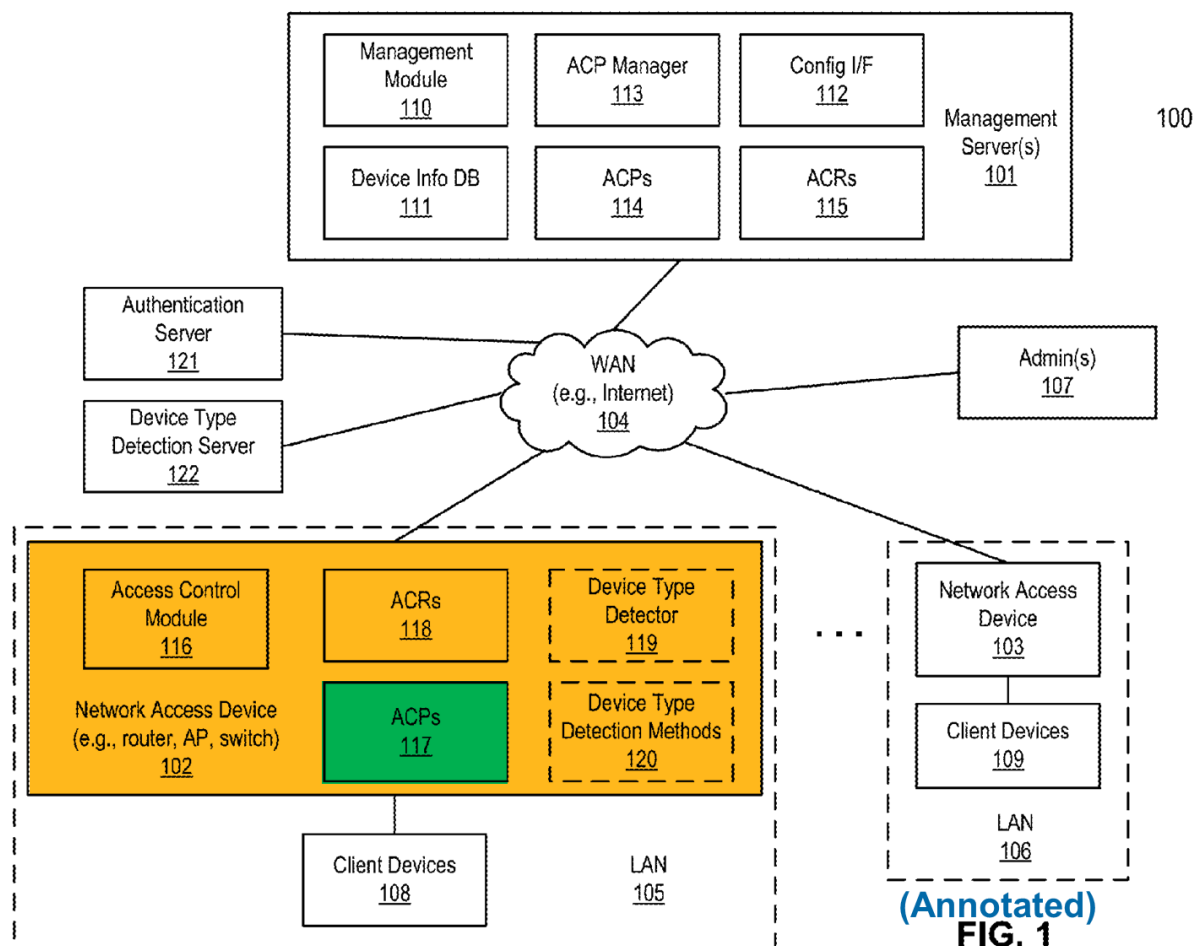
276. Thus, based on these teachings of Auradkar, a POSITA would have understood that Auradkar’s system stored encrypted data independent of the cloud service provider(s), because it used a standard format for storing data to multiple CSPs, which could be a cloud storage provided by “**any** storage provider” or “**another** entity that [was] **not fully trusted**.” (Auradkar, ¶¶75, 138.)

12. Claims 13/26

277. In my opinion, Auradkar and Chambers disclosed claims 13/26.

278. Chambers also disclosed that NAD 102 could receive ACPs 114 from

management server 101, which operated as a policy database storing policies for NAD 102. (Chambers, ¶¶27.) Chambers also disclosed that its network access device (NAD; gateway device) included an access control module (ACM) 116 that “select[ed] and applie[d] appropriate...[access control policies (]ACPs[)] 117” to a client device. (Chambers, ¶¶46, 21, FIG. 1; *supra* §I.D.4.e.)



279. A POSITA would have found it obvious that Chambers’ ACPs 117 (policies) could have been stored in a database operatively coupled to the NAD (gateway), because storing ACPs in a database would have allowed easier retrieval of a policy and maintenance of data integrity and Chambers already contemplated

storing ACPs in a database (Chambers, claim 1).

280. A POSITA would have been motivated to incorporate Chambers' policy database, which was coupled with the NAD (gateway), for storing policies and functionality for selecting a policy from policies into Auradkar's system, such that the modified system would have allowed selecting and assigning a policy from existing policies, instead of generating a policy ad-hoc for a user, thereby improving efficiency for assigning and implementing policies. (Chambers, ¶¶21, 46.) Additionally, a POSITA would have been motivated to add these features to Auradkar's system to reap the aforementioned well-known benefits of storing policies in a database. A POSITA would have reasonably expected success in combining Chambers' gateway and functionalities with Auradkar's system. (See *supra* §I.D.3.b.)

13. Claim 14/27

281. In my opinion, Auradkar rendered obvious claims 14/27.

282. Auradkar disclosed that its policy could define and manage "where it could be stored" (one or more authorized locations to which the file may be stored), a type of encryption (e.g., AES) and an encryption key (key that is 256 bits). (Auradkar, ¶96; see *supra* §I.D.3.))

283. Auradkar also disclosed generating encrypted indexes 1540 to be used in searches (search indices) using an encryption key 1525 and decrypting the

encrypted document using a decryption key 1644. (Auradkar, ¶¶130, 133, FIGS. 15-16.)

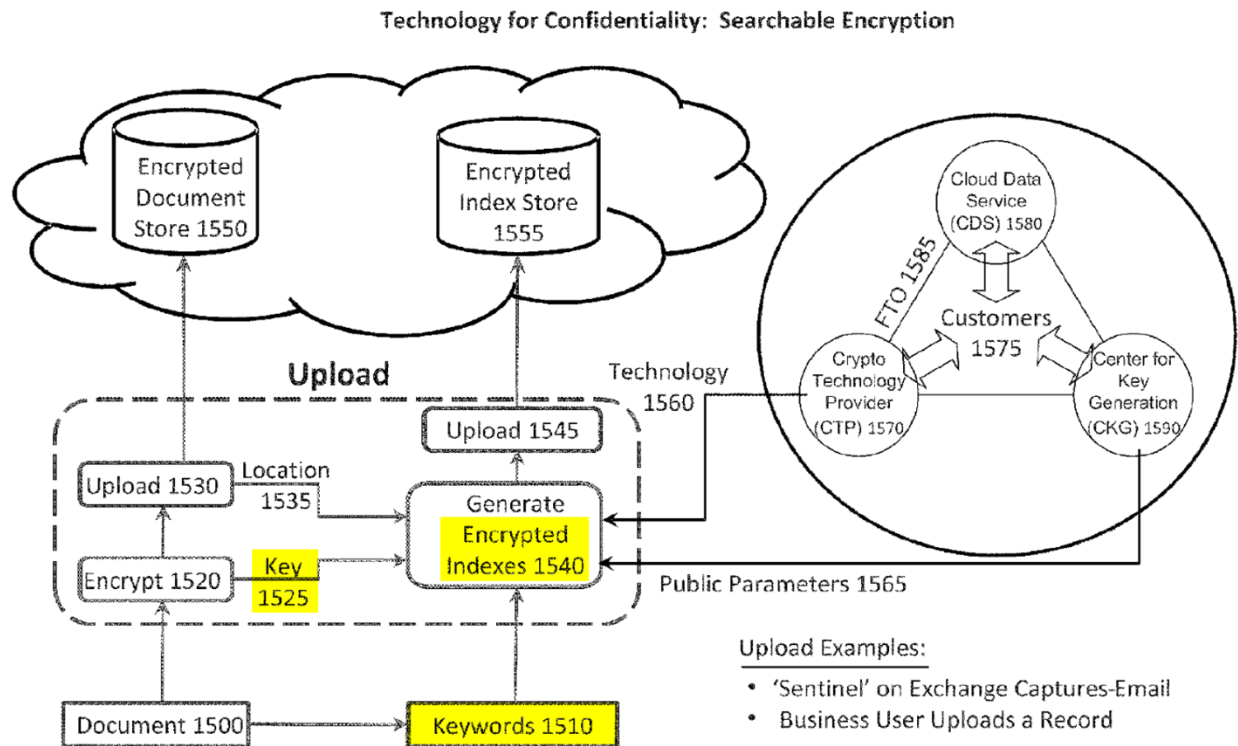
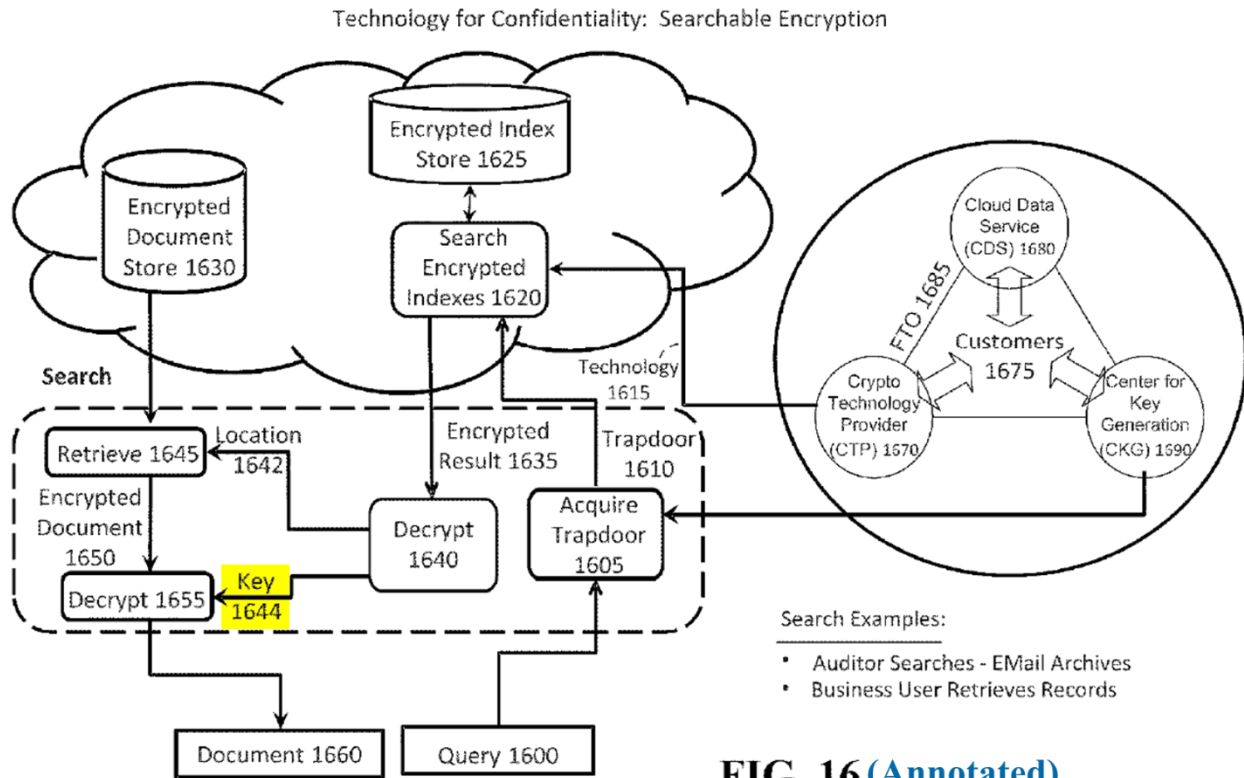


FIG. 15 (Annotated)



284. A POSITA would have found it obvious that Auradkar’s policy for a user would have also defined and managed encrypted indexes 1540 (e.g., the key for encrypting plaintext keywords to obtain encrypted indexes), encryption key 1525 (e.g., the strength thereof), and decryption key 1644 (e.g., the strength thereof), such that Auradkar’s system would have ensured encryption, decryption, and search operations for that user be consistent with the policy assigned thereto. For example, Auradkar explained that the policy could require “sensitive[] medical data” to be encrypted using AES 256 encryption, and a POSITA would have found it obvious the policy could also have defined and managed the key for encrypting indexes, encryption key 1525, and decryption key 1644 associated with the

medical data, such that they would also comply with those high security requirements. (Auradkar, ¶96.)

14. Claim 15

285. In my opinion, Auradkar disclosed claim 15.

286. Auradkar disclosed encrypting a tag 1200 that included the filename of the payload (FName: fox.tx) and keywords (“fox,” “dog”)⁴ into an encrypted tag 1210, which included an **encrypted filename and keywords**. (Auradkar, ¶125, 150, FIG. 12.)

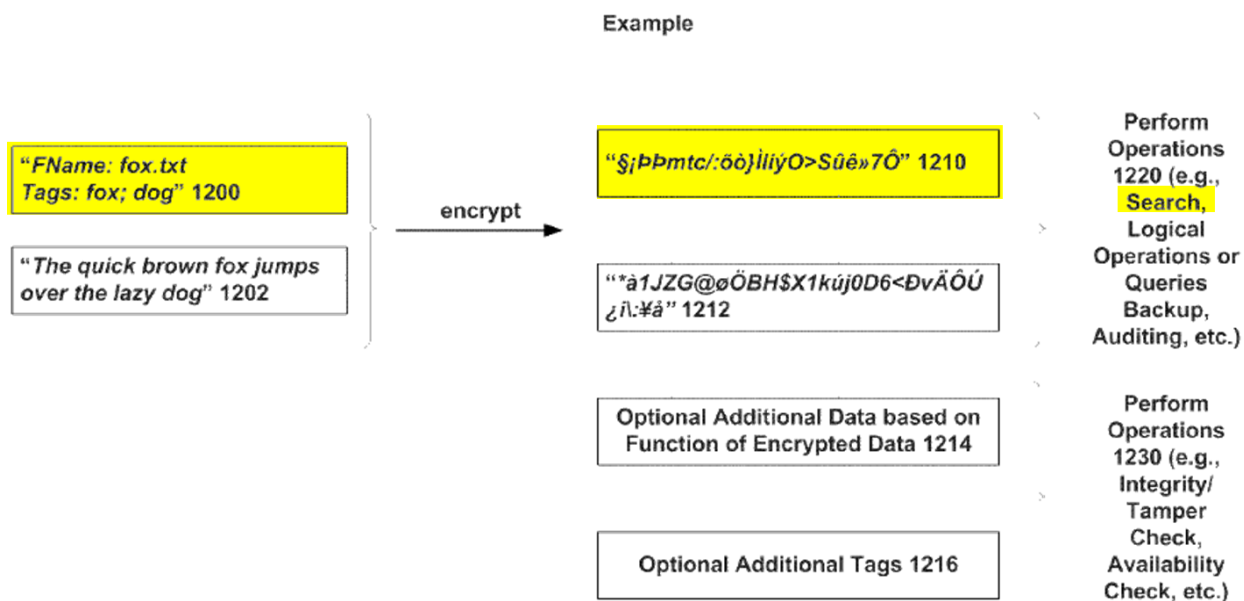


FIG. 12 (Annotated)

287. Auradkar also disclosed encrypting search indexes. For example,

⁴ The '678 Patent also refers “indices” as “keywords.” ('678 Patent, 12:7.)

Auradkar disclosed encrypting keywords 1510 associated with document 1500 to generate encrypted indexes 1540, which were used for searching the encrypted document. (Auradkar, ¶130, FIG. 15.)

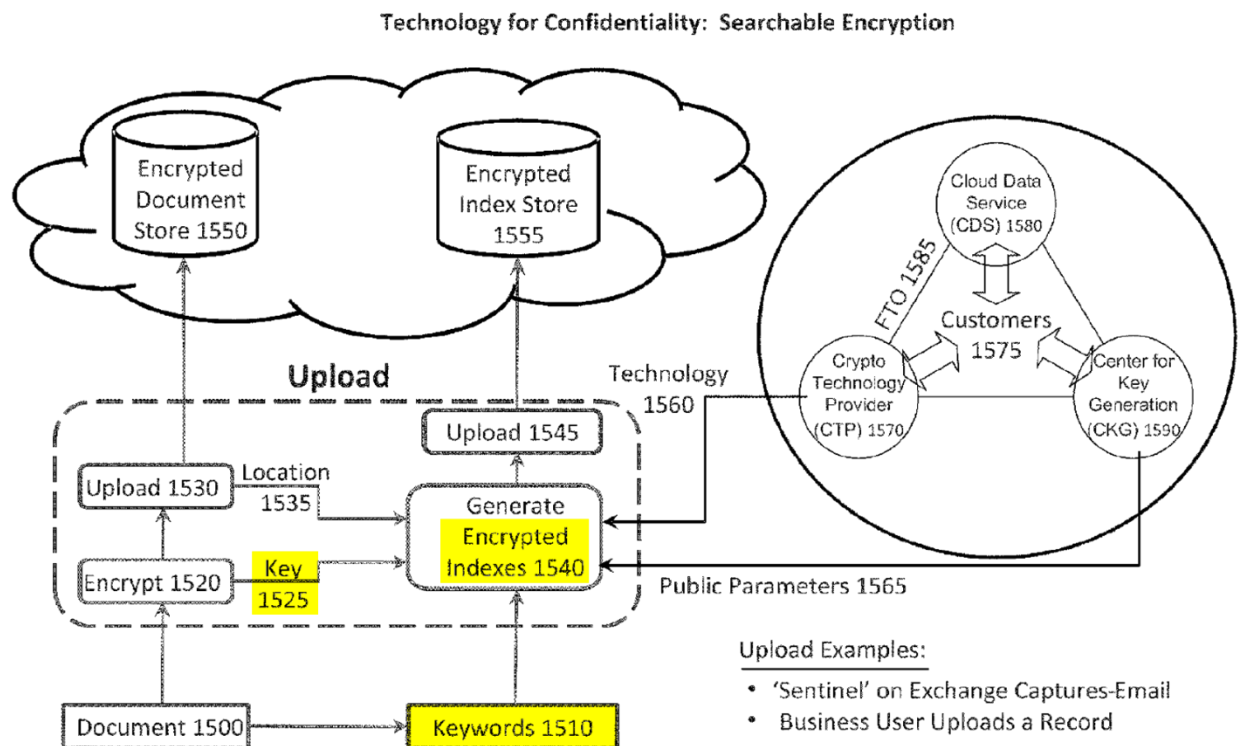


FIG. 15 (Annotated)

15. Claim 21

288. As I explained in Ground 1, claim 16, from which claim 21 depends, does not require (or prohibit) that the recited functions or modules be performed by a single gateway device. (See *supra* §§**Error! Reference source not found.**-**Error! Reference source not found.**) Additionally, claim 21 requires that various recited modules (or functions) are implemented on different devices. (See *supra* §§**Error! Reference source not found.**.)

289. In my opinion, Auradkar rendered obvious claim 21.

Software Implementation of Modules:

290. As explained above, a POSITA would have found it obvious that Auradkar's system would have included various software programs or modules, including the claimed various modules for performing corresponding functions. (See *supra* §§I.D.4.e, I.D.4.h, I.D.4.m-I.D.4.n.) Auradkar also contemplated that “various embodiments...described herein [could] be implemented in connection with any computer or other client or server device, which [could] be deployed...in a distributed computing environment,” such as the one illustrated in FIG. 53, including various computing devices, servers. (Auradkar, ¶¶247-248, FIG. 53, ¶¶246, 252.)

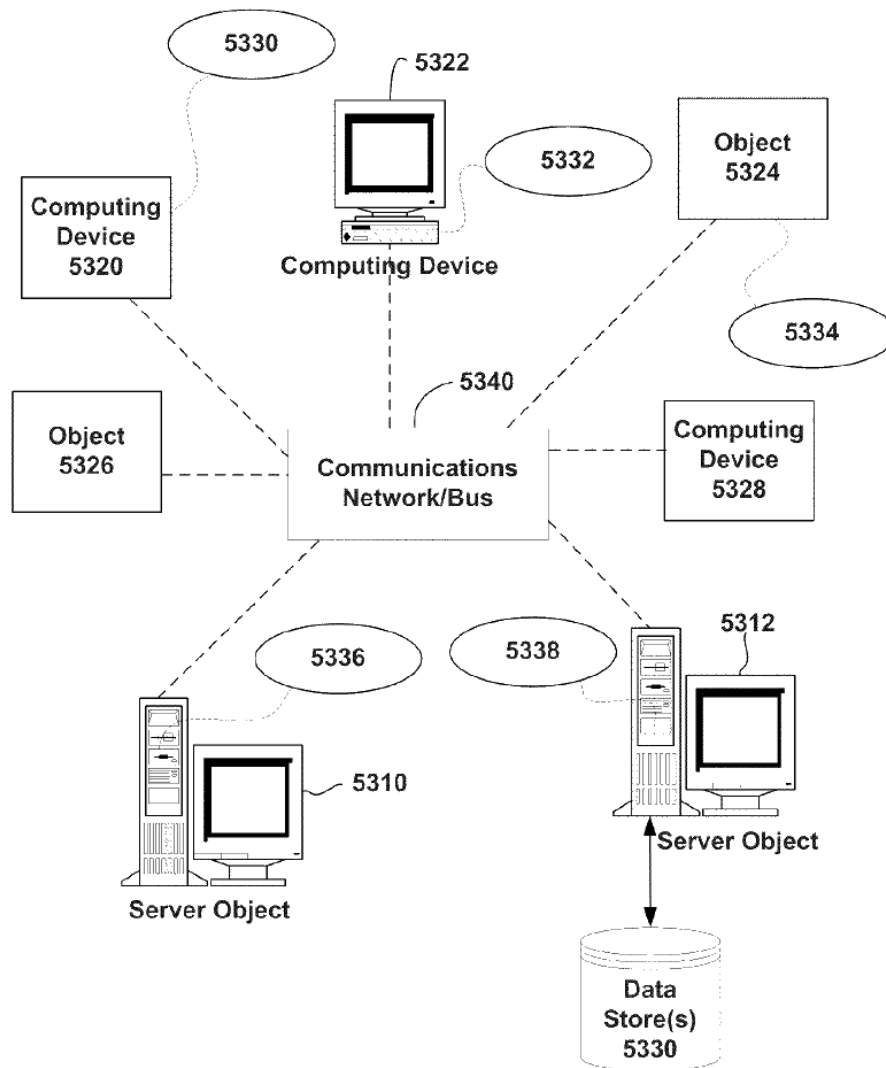


FIG. 53

Implementation using multiple devices:

291. Chambers also contemplated having multiple devices, such as “a gateway device,..., an access point (AP)..., a network switch,” as part of its network access device (NAD) to perform the functions described therein. (Chambers, ¶81.)

292. It had also been well-known that implementing functions with a distributed computing environment would increase processing power and

efficiency and improve balance task load. Therefore, a POSITA would have been motivated, from the teachings of Auradkar and Chambers, to implement these disclosed programs or modules on different devices, such as Auradkar's servers for access management and Chambers' network access device (gateway; *see supra* §I.D.3.b) and network switch (Chambers, ¶81; network controller) to reap these benefits of distributed computing.

293. A POSITA would have reasonably expected success in implementing these modules in the aforementioned devices because these references already contemplated that the functions could be implemented in a distributed computing environment, and the modification would have yielded expected results.

16. Claim 22

294. In my opinion, Auradkar rendered obvious claim 22 because it described a software abstraction service (mediation module) that made the operations for generating a searchable encrypted content independent of the cloud storage service.

295. As I explained above, Auradkar disclosed that storing the searchable encrypted file on the cloud platform was carried out independent of the service provider of the cloud platform. (*See supra* §I.D.11.) For example, Auradkar disclosed a storage abstraction service (SAS) that used storage transfer format (STF), "a standard interchange format that [could] be used for exchanging

encrypted data and metadata **across repositories,**” such as private cloud store 3000, SQL data store 3002, simple storage web service store 3004.

(Auradkar, ¶193, FIG. 30; *see also id.*, ¶¶189, 191, FIGS. 28-29.)

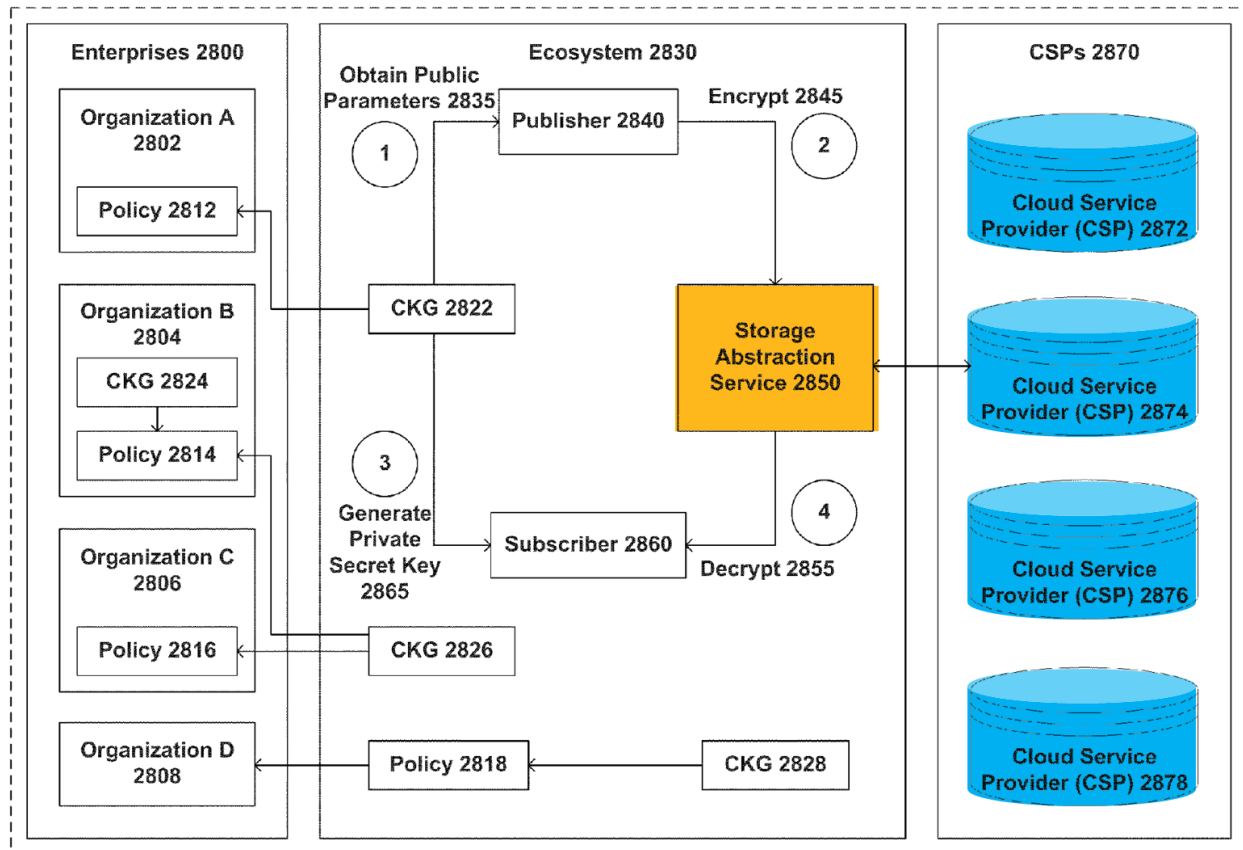


FIG. 28 (Annotated)

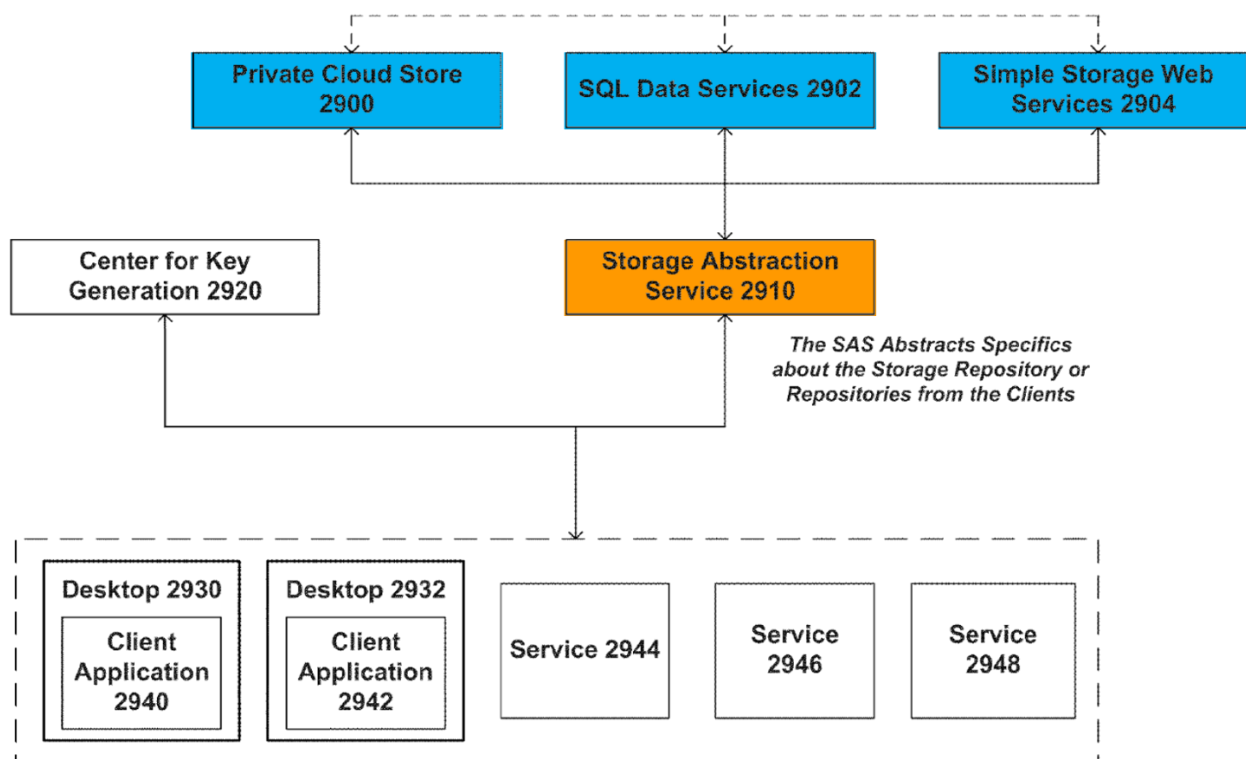


FIG. 29 (Annotated)

296. A POSITA would have found it obvious that Auradkar’s SAS constituted the claimed mediation module, because it also “facilitat[ed] geographical and provider diversity in data placement, making the system agnostic with respect to specific cloud vendor APIs” (’678 Patent, 4:46-49.) by, e.g., using STF for exchanging encrypted data across different cloud storages. (Auradkar, ¶193.)

297. A POSITA would have further understood the role of Auradkar’s SAS as the claimed mediation entity because Auradkar also described that its system was designed use a cloud service provider (CSP) that may “**not fully trusted.**” (Auradkar, ¶¶75, 139, 127-128, FIG. 14.) Auradkar further described that its

federal trust overlay (FTO) services (e.g., encryption) were “independent of the data services in the cloud...[and] **operated by parties other than the operator of the data services** and [were] able to provide strong guarantees regarding confidentiality, tamper detection and non-repudiation for the data hosted by the cloud services.” (Auradkar, ¶¶237, 75.)

298. From these teachings of Auradkar, a POSITA would have found it obvious that Auradkar’s disclosed functionalities for policy assignment and management, encryption, and storage would have been carried out independent of the cloud storage service provider, because Auradkar contemplated that encrypted data could be stored to a cloud hosted by an untrusted entity, which should not be given access to any of these functionalities. (Auradkar, ¶237, ¶78 (describing that “enterprises [could] own policy and control enforcement in a granular manner, even if data and application [were] hosted in the cloud”).)

E. Ground 5: Claim 6 was obvious over Auradkar, Chiueh, Chambers, and Inoue

1. Claim 6: “wherein the step of configuring the namespaces comprises storing content of the file as filenames.”

299. In my opinion, Auradkar and Inoue rendered claim 6 obvious.

300. The ’678 Patent describes storing keywords from file content, which “would typically be used as search terms in the form of search queries,” as filenames. (’678 Patent, 12:7-9.)

301. As I explained Auradkar disclosed generating and encrypting keywords (file content) for searching encrypted documents that include these keywords. (See *supra* §I.D.3.d.) For example, Auradkar described encrypting a “document 1500 with which various **keywords 1510** [were] associated” and uploading the encrypted document to “the cloud into an encrypted document store 1550.” (Auradkar, ¶130, FIG. 15.)

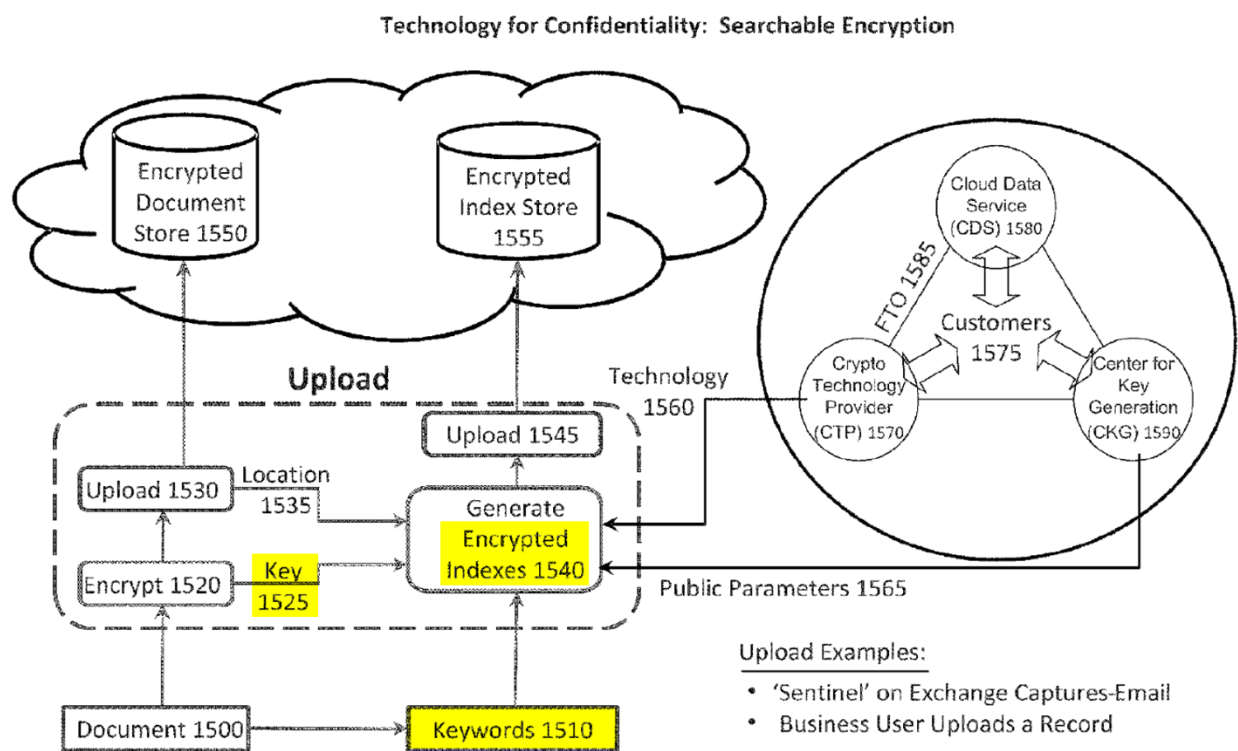


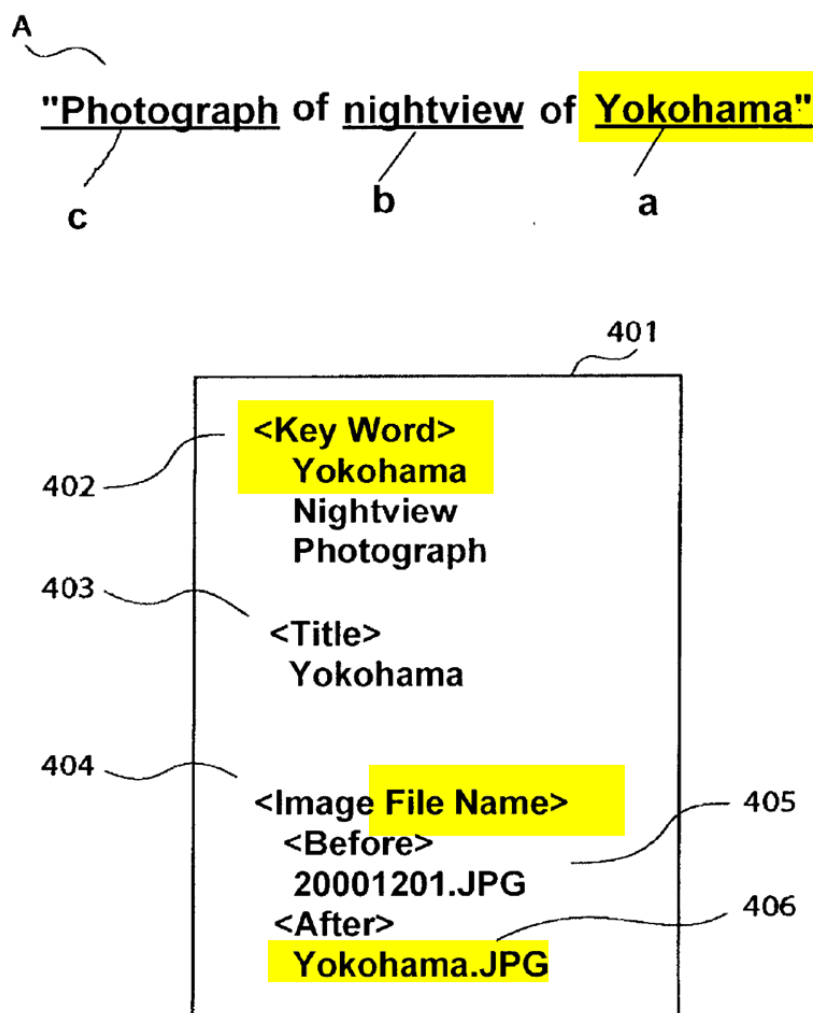
FIG. 15 (Annotated)

302. While Auradkar did not explicitly disclose storing keywords as filenames, Inoue disclosed this feature.

303. Specifically, Inoue disclosed “correlate[ing] one or more extracted keywords (character string data) with the image data as keywords for image data

searches” (content of the file) and “set[ting] one of the keywords as the title (a filename) of the image data.” (Inoue, ¶104, FIG. 4.)

Fig. 4
(Annotated)



304. A POSITA would have been motivated to incorporate Inoue’s functionality of assigning a keyword extracted from file content (e.g., Auradkar’s keywords 1510) into the Auradkar-Chiueh-Chambers to produce filenames that

also included a keyword. In doing, in addition to Auradkar-Chiueh searching method through indexes (*see supra* §I.D.3.d), the modified system would also have allowed searching files by searching their filenames (which could be done with a simple string matching operation) via its filesystem, which is usually faster and more efficient.

305. A POSITA would have found further motivation to combine with a reasonable expectation of success because Inoue, like Auradkar and Chiueh, was concerned with facilitating content searching based on keywords. (Auradkar, ¶130; Inoue, Abstract; Chiueh, ¶40.) Thus, a POSITA would have understood that those references sought to address similar problems for searching files, and would have benefited from the complementary techniques of Inoue to facilitate and expand the search options.

306. A POSITA would have also reasonably expected success in combining Inoue with Auradkar because assigning keywords to filenames had been well-known, Auradkar's technique would have been compatible with Auradkar's system, and the combination would have yielded expected results. (Fellenstei, ¶8; Ex. 1017, ¶8.)

II. CONCLUSION

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; that these statements were made with knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001; and further that such willful false statements may jeopardize the validity of the application or any patent issued thereon. I declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct.

Executed on January 31, 2023 in Atlanta, GA.

A handwritten signature in black ink, appearing to read 'Wenke Lee', is written above a horizontal line.

Wenke Lee, Ph.D.

Wenke Lee
Professor and John P. Imlay Jr. Chair
School of Cybersecurity and Privacy and School of Computer Science
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
wenke@cc.gatech.edu
<http://www.cc.gatech.edu/~wenke/>

Contents

EDUCATIONAL BACKGROUND	2
EMPLOYMENT HISTORY	2
CURRENT FIELDS OF INTEREST	2
I. TEACHING	4
A. Courses Taught	4
B. Curriculum Development	6
C. Individual Student Guidance	8
II. RESEARCH AND CREATIVE SCHOLARSHIP	10
A. Theses	10
B. Published Journal Papers	10
C. Published Books and Parts of Books	11
D. Edited Proceedings and Books	11
E. Conference Presentations (refereed)	12
E.1 with Proceedings (refereed)	12
F. Workshop Presentations (refereed)	26
F.1 with Proceedings	26
G. Other	27
H. Research Proposals and Grants	29
I. Research Honors and Awards	36
III. SERVICE	38
A. Professional Activities	38
A.1 Membership and Activities in Professional Societies	38
A.2 Journal Edit-oral Board Activities	38
A.3 Conference Committee Activities	38
B. On-campus Georgia Tech Committees	42
IV. NATIONAL AND INTERNATIONAL PROFESSIONAL RECOGNITION	44
A. Patents	44
V. OTHER CONTRIBUTIONS	46
A. Seminar Presentations	46

EDUCATIONAL BACKGROUND

Degree	Year	University	Field
Ph.D.	1999	Columbia University	Computer Science
M.S.	1990	The City College of New York	Computer Science
B.S.	1988	Sun Yat-Sen University, China	Computer Science

EMPLOYMENT HISTORY

Title	Organization	Years
John P. Imlay Jr. Chair	College of Computing	3/2016–present
Director	Institute for Information Security & Privacy (IISP)	7/2015–7/2021
	Georgia Institute of Technology	
Director	Georgia Tech Information Security Center (GTISC)	8/2012–6/2015
	Georgia Institute of Technology	
Professor	College of Computing	2/2009–present
	Georgia Institute of Technology	
Associate Professor	College of Computing	2/2005–2/2009
	Georgia Institute of Technology	
Assistant Professor	College of Computing	8/2001–2/2005
	Georgia Institute of Technology	
Assistant Professor	Department of Computer Science	8/1999–8/2001
	North Carolina State University	
Research Assistant	Department of Computer Science	1994–1999
	Columbia University	
Research Staff Member	IBM T.J. Watson Research Center	5/1997–8/1997
Member of Technical Staff	AT&T Research	5/1996–8/1996
Senior Software Analyst	Intergraph Corporation	1991–1994
Software Analyst	Intergraph Corporation	1990–1991

CURRENT FIELDS OF INTEREST

Information Security

Main research thrusts are: “Malware analysis and detection” – developing “live” and “safe” environment to uncover malware behaviors and machine-learning based techniques to automatically generate detection models and forensic analysis rules; “Adversarial Machine Learning” – studying the vulnerabilities of machine learning based systems due to adversarial manipulations and developing countermeasures; “Software debloating and protocol subsetting” – developing program and protocol analysis and rewriting techniques to remove unnecessary features and code from program binaries; “Privacy-preserving biometric-based authentication and search” – researching efficient methods to protect the privacy of biometric data while enabling important security applications including re-

mote authentication and video surveillance; “Adaptive analysis framework for advanced persistent threats” – developing game-theoretic approaches to model and analyze advanced persistent threats.

I. TEACHING

A. Courses Taught

<u>Semester/Year</u>	<u>Course</u>	<u>Number of Students</u>	<u>Comments</u>
Fall 2021	OMS CS/Cybersecurity 6264 Information Security Labs: System and Network Defenses	25	Online
Fall 2021	OMS CS/Cybersecurity 6262 Network Security	360	Online
Fall 2021	OMS CS/Cybersecurity 6035 Introduction to Information Security	980	Online
Fall 2021	CS 6262 Network Security	40	At CoC
Summer 2021	OMS CS/Cybersecurity 6262 Network Security	200	Online
Summer 2021	OMS CS/Cybersecurity 6035 Introduction to Information Security	460	Online
Spring 2021	OMS CS/Cybersecurity 6264 Information Security Labs: System and Network Defenses	25	Online
Spring 2021	OMS CS/Cybersecurity 6262 Network Security	300	Online
Spring 2021	OMS CS/Cybersecurity 6035 Introduction to Information Security	800	Online
Fall 2020	OMS CS/Cybersecurity 8803 Information Security Labs: System and Network Defenses	25	Online
Fall 2020	OMS CS/Cybersecurity 6262 Network Security	260	Online
Fall 2020	OMS CS/Cybersecurity 6035 Introduction to Information Security	850	Online
Fall 2020	CS 6262 Network Security	40	At CoC
Summer 2020	OMS CS/Cybersecurity 8803 Information Security Labs: System and Network Defenses	20	Online
Summer 2020	OMS CS/Cybersecurity 6262 Network Security	150	Online
Summer 2020	OMS CS/Cybersecurity 6035 Introduction to Information Security	450	Online
Spring 2020	OMS CS/Cybersecurity 8803 Information Security Labs: System and Network Defenses	40	Online
Spring 2020	OMS CS/Cybersecurity 6262 Network Security	360	Online
Spring 2020	OMS CS/Cybersecurity 6035 Introduction to Information Security	884	Online
Fall 2019	OMS CS/Cybersecurity 6262 Network Security	156	Online
Fall 2019	OMS CS/Cybersecurity 6035 Introduction to Information Security	770	Online
Fall 2019	CS 6262 Network Security	43	At CoC
Summer 2019	OMS CS/Cybersecurity 6262 Network Security	129	Online
Summer 2019	OMS CS/Cybersecurity 6035 Introduction to Information Security	456	Online
Spring 2019	OMS CS/Cybersecurity 6262 Network Security	275	Online
Spring 2019	OMS CS/Cybersecurity 6035 Introduction to Information Security	938	Online

<u>Semester/Year</u>	<u>Course</u>	<u>Number of Students</u>	<u>Comments</u>
Fall 2018	OMSCS 6262 Network Security	154	Online
Fall 2018	OMSCS 6035 Introduction to Information Security	525	Online
Fall 2018	CS 6262 Network Security	47	At CoC
Summer 2018	OMSCS 6262 Network Security	191	Online
Summer 2018	OMSCS 6035 Introduction to Information Security	338	Online
Spring 2018	OMSCS 6262 Network Security	220	Online
Spring 2018	OMSCS 6035 Introduction to Information Security	428	Online
Fall 2017	OMSCS 6262 Network Security	300	Online
Fall 2017	OMSCS 6035 Introduction to Information Security	350	Online
Fall 2017	CS 4235/6035 Introduction to Information Security	120	At CoC
Summer 2017	OMSCS 6262 Network Security	300	Online
Summer 2017	OMSCS 6035 Introduction to Information Security	350	Online
Spring 2017	OMSCS 6262 Network Security	300	Online
Spring 2017	OMSCS 6035 Introduction to Information Security	350	Online
Fall 2016	OMSCS 6262 Network Security	300	Online
Fall 2016	OMSCS 6035 Introduction to Information Security	350	Online
Fall 2016	CS 4235/6035 Introduction to Information Security	80	At CoC
Summer 2016	OMSCS 6035 Introduction to Information Security	350	Online
Spring 2016	OMSCS 6035 Introduction to Information Security	350	Online
Fall 2015	OMSCS 6035 Introduction to Information Security	230	Online
Fall 2015	CS 4235/6035 Introduction to Information Security	80	At CoC
Fall 2014	CS 4235/6035 Introduction to Information Security	80	At CoC
Spring 2013	CS 4235 Introduction to Information Security	50	At CoC
Fall 2012	CS 4235 Introduction to Information Security	41	At CoC
Spring 2012	CS 6262 Network Security	35	At CoC

<u>Semester/Year</u>	<u>Course</u>	<u>Number of Students</u>	<u>Comments</u>
Fall 2010	CS 3210 Operating System Design	25	At CoC
Spring 2010	CS 6262 Network Security	45	At CoC
Fall 2009	CS 4237 Computer and Network Security	15	At CoC
Spring 2009	CS 4237 Computer and Network Security	15	At CoC
Fall 2008	CS 6262 Network Security	55	At CoC
Spring 2008	CS 8803ANS Advanced Systems and Network Security	25	At CoC
Fall 2007	CS 4237 Computer and Network Security	20	At CoC
Spring 2007	CS 6262 Network Security	45	At CoC
Fall 2006	CS 6265 Information Security Lab (w/ Mustaque)	18	At CoC
Fall 2006	CS 4803 Computer and Network Security	30	At CoC
Spring 2006	CS 6262 Network Security	45	At CoC
Fall 2005	CS 4803 Computer and Network Security	30	At CoC
Spring 2005	CS 6262 Network Security	30	At CoC
Fall 2004	CS 4803K Computer and Network Security	30	At CoC
Spring 2004	CS 6262 Network Security	30	At CoC
Fall 2003	CS 4803K Computer and Network Security	35	At CoC
Spring 2003	CS 8803K Advanced Systems and Network Security	16	At CoC
Fall 2002	CS 6262 Network Security	28	At CoC
Spring 2002	CS 6262 Network Security	30	New at CoC
Fall 2000	CSC 591Q Special Topics: Network Security	65	At NC State
Fall 1999	CSC 591Q Special Topics: Network Security	45	At NC State
Summer 1998	CS 3139 Data Structures and Algorithms	20	At Columbia

B. Curriculum Development

CS 6264/OMS CS/Cybersecurity 6264: Information Security Labs: System and Network Defenses. This graduate-level course helps students develop both in-depth knowledge and hands-on skills in a number of important cybersecurity areas, including software security, malware, and threat analysis, end-point security, network security, web security, mobile security, and machine learning-based security analytics. The lecture materials of each topic area are drawn from the latest research papers and prototypes, and a comprehensive project is designed to help students master each area. The lecture materials explain the design principles of cutting-edge security tools, and the projects are designed to let students extend these tools. Most of the tools are in the open-source, and therefore, students can continue to build on and use these tools beyond this course. Recorded videos for on-line offering as part of OMSCS as well as OMS Cybersecurity.

CS 6262/OMS CS/Cybersecurity 6262: Network Security. Graduate-level course in network security with topics including large-scale attacks and impacts, penetration testing and security assessments, security of Internet protocols (IP, TCP, DNS, and BGP), advanced web security, advanced malware analysis, advanced network monitoring, Internet-scale threat analysis, bitcoins and cryptocurrencies, big data and security, cloud security, and attack-tolerant systems. Completely revamped

in 2016 with new course materials drawing from research papers and projects developed by my Ph.D. students. Recorded videos for on-line offering via Udacity as part of OMSCS as well as OMS Cybersecurity.

CS 4235/OMS CS/Cybersecurity 6035: Introduction to Information Security. Cross-listed undergraduate and graduate introductory course in information security. It teaches the basic concepts, principles, and fundamental approaches to secure computers and networks. Its main topics include: security basics, security management and risk assessment, software security, operating systems security, database security, cryptography algorithms and protocols, network authentication and secure network applications, malware, network threats and defenses, web security, mobile security, legal and ethical issues, and privacy. Completely revamped in 2015 with new materials covering the up-to-date threat models, attack methods, and new technologies and policy considerations. Recorded videos for on-line offering via Udacity as part of OMS CS as well as OMS Cybersecurity.

CS 6262: Network Security. Graduate-level course focusing on the fundamental principles as well as advanced techniques in network security. This course covers cryptography and its application to network and operating system security; authentication; security for electronic mail; Java security; Firewalls and intrusion detection.

CS 4237: Computer and Network Security. New undergraduate course (first offered as CS 4803 in Fall 2003) focusing on the fundamental principles and techniques in computer and network security.

CS 8803: Advanced Systems and Network Security. Graduate course on advanced topics such as network and host-based intrusion detection, botnet detection, software security, malware analysis, and virtual machine monitoring. Students are required to study research papers and work on research projects.

C. Individual Student Guidance

1. Postdoctoral Fellows Supervised

Erkam Uzun: 2021 -

Yisroel Mirsky: 2019 - 2021; Tenure-track Lecturer in the Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Israel

Guangliang Yang: 2019 - 2021; Assistant Professor in the Department of Computer Science, Fudan University, Shanghai, China

Hong Hu: 2017-2019; Tenure-track Assistant Professor in the College of Information Sciences and Technology, Pennsylvania State University

Sangho Lee: 2015-2018; Researcher in Microsoft Research (MSR) Redmond

Tielei Wang: 2011 - 2014; Start-up in China

Simon Pak Ho Chung: 2011 -2014; Research Scientist in SCS at Georgia Tech

Roberto Perdisci: 2009 - 2010; Professor with tenure in the Department of Computer Science, University of Georgia

Daniel Xiapu Luo: 2008 - 2010; Associate Professor with tenure at Hong Kong Polytechnic University

2. Ph.D. Students Supervised

Xinzhou Qin: Graduated in Summer 2005; Juniper Networks

Yian Huang: Graduated in Summer 2006; Google

Prahlad Fogla: Graduated in Summer 2007; Google

Guofei Gu: Graduated in Summer 2008; Professor with tenure in the Department of Computer Science, Texas A&M University

Bryan Payne: Graduated in Summer 2010; Director of Product & Application Security, Netflix

Monirul Sharif: Graduated in Fall 2010; Engineering Manager III (L7), Google

Kapil Singh: Graduated in Summer 2011; Research Staff Member, IBM T.J. Watson Research Center

Manos Antonakakis: Graduated in Spring 2012; Associate Professor with tenure in ECE at Georgia Tech

Martim Carbone: Graduated in Summer 2012; Staff Engineer, VMware

Junjie Zhang: Graduated in Summer 2012; Associate Professor with tenure in the Department of Computer Science and Engineering, Wright State University

Long Lu: Graduated in Summer 2013; Associate Professor with tenure in the College of Computer and Information Science, Northeastern University

Brendan Dolan-Gavitt: Graduated in Summer 2014; tenure-track Assistant Professor in the Department of Computer Science and Engineering, NYU Tandon School of Engineering (Polytechnic Institute)

Yacin Nadji: Graduated in Summer 2015; Corelight

Xinyu Xing: Graduated in Summer 2015; Associate Professor of Computer Science, Northwestern University

Chengyu Song: Graduated in Summer 2016; Tenure-track Assistant Professor in the Department of Computer Science and Engineering, UC Riverside

Byoungyoung Lee: Graduated in Summer 2016; Associate Professor in the Department of Electrical and Computer Engineering, Seoul National University, South Korea

Yeongjin Jang: Graduated in Summer 2017; Tenure-track Assistant Professor in the Department of Electrical Engineering and Computer Science, Oregon State University

Kangjie Lu: Graduated in Summer 2017; Tenure-track Assistant Professor in the Department of Computer Science and Engineering, University of Minnesota

Wei Meng: Graduated in Summer 2017; Tenure-track Assistant Professor in the Department of Computer Science and Engineering, The Chinese University of Hong Kong

Yizheng Chen: Graduated in Fall 2017; Post-doc in EECS, UC Berkeley

Yang Ji: Graduated in Fall 2019; Research Scientist in Palo Alto Networks

Ruian Duan: Graduated in Fall 2019; Research Scientist in Palo Alto Networks

Erkam Uzun: Graduated in Summer 2021; Post-doc at Georgia Tech

Chenxiong Qian: Graduated in Summer 2021; Tenure-track Assistant Professor in the Department of Computer Science, The University of Hong Kong

Evan Downing: In Progress

Kyuhong Park: In Progress

Carter Yagemann: In Progress

Joey Allen: In Progress

Matthew Landen: In Progress

Burak Sahin: In Progress

Yongheng Chen: In Progress

Feng Xiao: In Progress

Haoran Wang: In Progress

Ding Zhang: In Progress

Zhang Yang: In Progress

II. RESEARCH AND CREATIVE SCHOLARSHIP

A. Theses

Ph.D. Thesis

Title: A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems.

Date Completed: June 1999

Advisor: Professor Salvatore J. Stolfo.

University: Columbia University

B. Published Journal Papers (refereed)

1. Yisroel Mirsky and Wenke Lee. The Creation and Detection of Deepfakes: A Survey. *ACM Computing Surveys*. 54(1), 2021.
2. Kangjie Lu, Meng Xu, Chengyu Song, Taesoo Kim, and Wenke Lee. Stopping Memory Disclosures via Diversification and Replicated Execution. *IEEE Transactions on Dependable and Secure Computing (TDSC)*. 18(1). 2021.
3. Shana Moothedath, Dinuka Sahabandu, Joey Allen, Andrew Clark, Linda Bushnell, Wenke Lee, and Radha Poovendran. A Game-Theoretic Approach for Dynamic Information Flow Tracking to Detect Multistage Advanced Persistent Threats. *IEEE Transactions on Automatic Control*. 65(12): 5248-5263, 2020.
4. Junjie Zhang, Roberto Perdisci, Wenke Lee, Unum Sarfraz, and Xiapu Luo. Building a Scalable System for Stealthy P2P-Botnet Detection. *IEEE Transactions on Information Forensics and Security*, January 2014.
5. Roberto Perdisci, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. McPAD: A Multiple Classifier System for Accurate Payload-Based Anomaly Detection. *Computer Networks*, Vol. 53, 2009.
6. Roberto Perdisci, Andrea Lanzi, and Wenke Lee. Classification of Packed Executables for Accurate Computer Virus Detection. *Pattern Recognition Letters* Vol. 29, No. 14 (October 2008).
7. Prahlad Fogla and Wenke Lee. q-Gram Matching Using Tree Models. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 4 (April 2006).
8. W. Fan, M. Miller, S. Stolfo, Wenke Lee, and P. Chan. Using Artificial Anomalies to Detect Unknown and Known Network Intrusions. *Knowledge and Information Systems*, Vol. 6, No. 5 (September 2004), Springer.
9. Yongguang Zhang, Wenke Lee, and Yian Huang. Intrusion Detection Techniques for Mobile Wireless Networks. *ACM/Kluwer Wireless Networks Journal (ACM WINET)*, 9(5), September 2003.

10. Joao B.D. Cabrera, Lundy Lewis, Xinzhou Qin, Wenke Lee, and Raman K. Mehra. Proactive Intrusion Detection and Distributed Denial of Service Attacks - A Case Study in Security Management. *Journal of Network and Systems Management*. 10(2), June 2002.
11. Wenke Lee, Wei Fan, Matt Miller, Sal Stolfo, and Erez Zadok. Toward Cost-Sensitive Modeling for Intrusion Detection and Response. *Journal of Computer Security*, 10(1,2), 2002.
12. Wenke Lee and Sal Stolfo. A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.
13. Wenke Lee, Sal Stolfo, and Kui Mok. Adaptive Intrusion Detection: A Data Mining Approach. *Artificial Intelligence Review*, Kluwer Academic Publishers, 14(6):533-567, December 2000.
14. Wenke Lee and Gail E. Kaiser. Interfacing Oz with the PCTE OMS: A Case Study of Integrating a Legacy System with a Standard Object Management System. *Journal of Systems Integration*, Kluwer Academic Publishers, 9(4):329-358, December 1999.

C. Published Books and Parts of Books

1. Xinzhou Qin and Wenke Lee. Discovering Novel Attack Strategies from INFOSEC Alerts. *Data Warehousing and Data Mining Techniques for Cyber Security*. Anoop Singhal (eds), Springer, 2007.
2. Yongguang Zhang and Wenke Lee. Security in Mobile Ad-Hoc Networks. *Ad Hoc Networks: Technologies and Protocols*. P. Mohapatra and S. Krishnamurthy (eds), Springer, 2004.
3. Xinzhou Qin, Wenke Lee, Lundy Lewis, Joao B.D. Cabrera. Using MIB II Variables for Network Intrusion Detection. *Applications of Data Mining in Computer Security*. D. Barbara and S. Jajodia (eds), Kluwer Academic Publishers, May 2002.
4. Joao B.D. Cabrera, Lundy Lewis, Xinzhou Qin, Wenke Lee, Raman K. Mehra. Proactive Intrusion Detection - A Study on Temporal Data Mining. *Applications of Data Mining in Computer Security*. D. Barbara and S. Jajodia (eds), Kluwer Academic Publishers, May 2002.
5. Wenke Lee, Sal Stolfo, and Kui Mok. Algorithms for Mining System Audit Data. *Data Mining, Rough Sets, and Granular Computing*. T. Y. Lin, Y. Y. Yao, and L. A. Zadeh (eds), Physica-Verlag, 2002.
6. Wenke Lee and Naser Barghouti. Jadve: An Extensible Data Visualization Environment. In *Object-Oriented Applications Frameworks*, M. Fayad, D. Schmidt, and R. Johnson (eds), John Wiley & Sons, 1999.

D. Edited Proceedings and Books

1. *Botnet Detection: Countering the Largest Security Threat (Advances in Information Security)*, Wenke Lee, Cliff Wang, and David Dagon (Eds.), Springer, 2007.

2. *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, Wenke Lee, Ludovic Me, and Andreas Wespi (Eds.), Lecture Notes in Computer Science, Vol. 2212, Springer, 2001.

E. Conference Presentations (refereed)

E.1. Conference Presentations with Proceedings

1. Carter Yagemann, Simon Chung, Brendan Saltaformaggio, and Wenke Lee. Automated Bug Hunting with Data-Driven Symbolic Root Cause Analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021.
2. Carter Yagemann, Mohammad Nouredine, Wajih Hassan, Simon Chung, Adam Bates, and Wenke Lee. Validating the Integrity of Audit Logs Against Execution Repartitioning Attacks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2021.
3. Erkam Uzun, Simon P. Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee. Fuzzy Labeled Private Set Intersection with Applications to Private Real-Time Biometric Search. In *Proceedings of the 2021 USENIX Security Symposium*. 2021.
4. Evan Downing, Kyuhong Park, Yisroel Mirsky, and Wenke Lee. DeepReflect: Discovering Malicious Functionality through Binary Reconstruction. In *Proceedings of the 2021 USENIX Security Symposium*. 2021.
5. Carter Yagemann, Matthew Pruett, Simon P. Chung, Kennon Bittick, Brendan Saltaformaggio, and Wenke Lee. ARCUS: Symbolic Root Cause Analysis of Exploits in Production Systems. In *Proceedings of the 2021 USENIX Security Symposium*. 2021.
6. Feng Xiao, Jianwei Huang, Yichang Xiong, Guangliang Yang, Hong Hu, Guofei Gu, and Wenke Lee. Abusing Hidden Properties to Attack the Node.js Ecosystem. In *Proceedings of the 2021 USENIX Security Symposium*. 2021.
7. Kyuhong Park, Burak Sahin, Yongheng Chen, Jisheng Zhao, Evan Downing, Hong Hu, and Wenke Lee. Identifying Behavior Dispatchers for Malware Analysis. In *Proceedings of the 16th ACM ASIA Conference on Computer and Communications Security (ACM AsiaCCS 2021)*.
8. Erkam Uzun, Carter Yagemann, Simon P. Chung, Vladimir Kolesnikov, and Wenke Lee. Cryptographic Key Derivation from Biometric Inferences for Remote Authentication. In *Proceedings of the 16th ACM ASIA Conference on Computer and Communications Security (ACM AsiaCCS 2021)*.
9. Dongsong Yu, Guangliang Yang, Guozhu Meng, Xiaorui Gong, Xiu Zhang, Xiaobo Xiang, Xiaoyu Wang, Yue Jiang, Kai Chen, Wei Zou, Wenke Lee, and Wenchang Shi. SEPAL: Towards a Large-scale Analysis of SEAndroid Policy Customization. In *Proceedings of The Web Conference 2021 (WWW 2021)*.

10. Yongheng Chen, Rui Zhong, Hong Hu, Hangfan Zhang, Yupeng Yang, Dinghao Wu, and Wenke Lee. One Engine to Fuzz 'em All: Generic Language Processor Testing with Semantic Validation. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland)*. 2021.
11. Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2021.
12. Kangjie Lu, Meng Xu, Chengyu Song, Taesoo Kim, and Wenke Lee. Stopping Memory Disclosures via Diversification and Replicated Execution. *IEEE Transactions on Dependable and Secure Computing (TDSC)*. 18(1), 2021.
13. Carter Yagemann, Simon P. Chung, Erkam Uzun, Sai Ragam, Brendan Saltaformaggio, and Wenke Lee. On the Feasibility of Automating Stock Market Manipulation. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*. 2020.
14. Chenxiong Qian, Hyungjoon Koo, ChangSeok Oh, Taesoo Kim, and Wenke Lee. Slimium: Debloating the Chromium Browser with Feature Subsetting. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020.
15. Joey Allen, Zheng Yang, Matthew Landen, Raghav Bhat, Harsh Grover, Andrew Chang, Yang Ji, Roberto Perdisci, and Wenke Lee. Mnemosyne: An Effective and Efficient Postmortem Watering Hole Attack Investigation System. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020.
16. Rui Zhong, Yongheng Chen, Hong Hu, Hangfan Zhang, Wenke Lee, and Dinghao Wu. SQUIRREL: Testing Database Management Systems with Language Validity and Coverage Feedback. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2020.
17. Dinuka Sahabandu, Joey Allen, Shana Moothedath, Linda Bushnell, Wenke Lee, and Radha Poovendran. Quickest Detection of Advanced Persistent Threats: A Semi-Markov Game Approach. In *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. 2020.
18. D. Sahabandu, S. Moothedath, J. Allen, A. Clark, L. Bushnell, Wenke Lee, and R. Poovendran. Dynamic Information Flow Tracking Games for Simultaneous Detection of Multiple Attackers. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, Nice, France, December 2019.
19. S. Misra, S. Moothedath, H. Hosseini, J. Allen, L. Bushnell, Wenke Lee, and R. Poovendran. Learning Equilibria in Stochastic Information Flow Tracking Games with Partial Knowledge. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, Nice, France, December 2019.

20. Dinuka Sahabandu, Shana Moothedath, Joey Allen, Linda Bushnell, Wenke Lee, and Radha Poovendran. Stochastic Dynamic Information Flow Tracking Game with Reinforcement Learning. In *Proceedings of the 2019 Conference on Decision and Game Theory for Security*, Stockholm, Sweden, October 2019.
21. Chenxiong Qian, Hong Hu, Mansour Alharthi, Pak Ho Chung, Taesoo Kim, and Wenke Lee. RAZOR: A Framework for Post-deployment Software Debloating. In *Proceedings of the 28th USENIX Security Symposium*. Santa Clara, CA, August 2019.
22. Dinuka Sahabandu, Shana Moothedath, Linda Bushnell, Radha Poovendran, Joey Allen, Wenke Lee, and Andrew Clark. A Game Theoretic Approach for Dynamic Information Flow Tracking with Conditional Branching. In *Proceedings of the 2019 American Control Conference (ACC)*. Philadelphia, PA, July 2019.
23. Ruian Duan, Ashish Bijlani, Yang Ji, Omar Alrawi, Yiyuan Xiong, Moses Ike, Brendan Saltaformaggio, and Wenke Lee. Automating Patching of Vulnerable Open-Source Software Versions in Application Binaries. In *Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2019.
24. Joey Allen, Matthew Landen, Sanya Chaba, Yang Ji, Simon Pak Ho Chung, and Wenke Lee. Improving Accuracy of Android Malware Detection with Lightweight Contextual Awareness. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC)*. December, 2018.
25. Dinuka Sahabandu, Baicen Xiao, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. DIFT Games: Dynamic Information Flow Tracking Games for Advanced Persistent Threats. In *Proceedings of The 57th IEEE Conference on Decision and Control (CDC)*. Miami Beach, FL, December 2018.
26. Shana Moothedath, Dinuka Sahabandu, Andrew Clark, Sangho Lee, Wenke Lee, and Radha Poovendran. Multi-Stage Dynamic Information Flow Tracking Game. In *Proceedings of The 9th Conference on Decision and Game Theory for Security (GameSec)*. Seattle, WA, October 2018.
27. Hong Hu, Chenxiong Qian, Carter Yagemann, Simon Pak Ho Chung, Bill Harris, Taesoo Kim, and Wenke Lee. Enforcing Unique Code Target Property for Control-Flow Integrity. In *Proceedings of The 25th ACM Conference on Computer and Communications Security (CCS 2018)*. Toronto, Canada, October 2018.
28. Andrea Possemato, Andrea Lanzi, Simon Pak Ho Chung, Wenke Lee, and Yanick Fratantonio. ClickShield: Are You Hiding Something? Towards Eradicating Clickjacking on Android. In *Proceedings of The 25th ACM Conference on Computer and Communications Security (CCS 2018)*. Toronto, Canada, October 2018.
29. Yang Ji, Sangho Lee, Mattia Fazzini, Joey Allen, Evan Downing, Taesoo Kim, Alessandro Orso, and Wenke Lee. Enabling Refinable Cross-Host Attack Investigation with Efficient Data Flow Tagging and Tracking. In *Proceedings of the 27th USENIX Security Symposium*. Baltimore, MD, August 2018

30. Wei Meng, Chenxiong Qian, Shuang Hao, Kevin Borgolte, Giovanni Vigna, and Christopher Kruegel, and Wenke Lee. Rampart: Protecting Web Applications from CPU-Exhaustion Denial-of-Service Attacks. In *Proceedings of the 27th USENIX Security Symposium*. Baltimore, MD, August 2018
31. Erkam Uzun, Simon Pak Ho Chung, Irfan Essa, and Wenke Lee. rtCaptcha: A Real-Time CAPTCHA Based Liveness Detection System. In *Proceedings of The 2018 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2018.
32. Antonio Bianchi, Yanick Fratantonio, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, Simon Pak Ho Chung, and Wenke Lee. Broken Fingers: On the Usage of the Fingerprint API in Android. In *Proceedings of the 2018 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2018.
33. Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee. Identifying Open-Source License Violation and 1-day Security Risk at Large Scale. In *Proceedings of The 24th ACM Conference on Computer and Communications Security (CCS 2017)*. Dallas, Texas, October 2017.
34. Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee. RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking. In *Proceedings of The 24th ACM Conference on Computer and Communications Security (CCS 2017)*. Dallas, Texas, October 2017.
35. Ren Ding, Chenxiong Qian, Chengyu Song, Bill Harris, Taesoo Kim, and Wenke Lee. Efficient Protection of Path-Sensitive Control Security. In *Proceedings of the 26th USENIX Security Symposium*. Vancouver, BC, Canada, August 2017.
36. Meng Xu, Kangjie Lu, Taesoo Kim, and Wenke Lee. Bunshin: Compositing Security Mechanisms through Diversification. In *Proceedings of the 2017 USENIX Annual Technical Conference*. Santa Clara, CA, July 2017.
37. Yanick Fratantonio, Chenxiong Qian, Pak Chung, and Wenke Lee. Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop. In *Proceedings of The 2017 IEEE Symposium on Security and Privacy*. San Jose, CA, May 2017 (**Distinguished Practical Paper Award**).
38. Kangjie Lu, Marie-Therese Walter, David Pfaff, Stefan Nuernberger, Wenke Lee, and Michael Backes. Unleashing Use-Before-Initialization Vulnerabilities in the Linux Kernel Using Targeted Stack Spraying. In *Proceedings of The 2017 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2017.
39. Le Guan, Jun Xu, Shuai Wang, Xinyu Xing, Lin Lin, Heqing Huang, Peng Liu, and Wenke Lee. From Physical to Cyber: Escalating Protection for Personalized Auto Insurance. In *Proceedings of The 14th ACM Conference on Embedded Networked Sensor Systems (SenSys 2016)*. Stanford, CA, November 2016.

40. Kangjie Lu, Chengyu Song, Taesoo Kim, and Wenke Lee. UniSan: Proactive Kernel Memory Initialization to Eliminate Data Leakages. In *Proceedings of The 23rd ACM Conference on Computer and Communications Security (CCS 2016)*. Vienna, Austria, October 2016.
41. Yizheng Chen, Panagiotis Kintis, Manos Antonakakis, Yacin Nadji, David Dagon, Wenke Lee, and Michael Farrell. Financial Lower Bounds of Online Advertising Abuse - A Four Year Case Study of the TDSS/TDL4 Botnet. In *Proceedings of The 13th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2016)*. Sebastian, Spain, July 2016.
42. Chengyu Song, Hyungon Moon, Monjur Alam, Insu Yun, Byoungyoung Lee, Taesoo Kim, and Wenke Lee, Yunheung Paek. HDFI: Hardware-Assisted Data-Flow Isolation. In *Proceedings of The 2016 IEEE Symposium on Security and Privacy*. San Jose, CA, May 2016.
43. Wei Meng, Byoungyoung Lee, Xinyu Xing, and Wenke Lee. TrackMeOrNot: Enable Flexible Control on Web Tracking. In *Proceedings of The 25th International World Wide Web Conference (WWW), April 2016*. San Diego, CA, February 2016.
44. Wei Meng, Ren Ding, Simon P. Chung, Steven Han, and Wenke Lee. The Price of Free: Privacy Leakage in Personalized Mobile In-Apps Ads. In *Proceedings of The 2016 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2016.
45. Kangjie Lu, Wenke Lee, Stefan Nrnberger, and Michael Backes. How to Make ASLR Win the Clone Wars: Runtime Re-Randomization. In *Proceedings of The 2016 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2016.
46. Chengyu Song, Byoungyoung Lee, Kangjie Lu, William Harris, Taesoo Kim and Wenke Lee. Enforcing Kernel Security Invariants with Data Flow Integrity. In *Proceedings of The 2016 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February, 2016.
47. Meng Xu, Yeongjin Jang, Xinyu Xing, Taesoo Kim, and Wenke Lee. UCognito: Private Browsing without Tears. In *Proceedings of The 22nd ACM Conference on Computer and Communications Security (CCS)*. Denver, CO, October 2015.
48. Kangjie Lu, Chengyu Song, Byoungyoung Lee, Simon P. Chung, Taesoo Kim, and Wenke Lee. ASLR-Guard: Stopping Address Space Leakage for Code Reuse Attacks. In *Proceedings of The 22nd ACM Conference on Computer and Communications Security (CCS)*. Denver, CO, October 2015.
49. Byoungyoung Lee, Chengyu Song, Taesoo Kim, and Wenke Lee. Type Casting Verification: Stopping an Emerging Attack Vector. In *Proceedings of The 24th USENIX Security Symposium*. Washington, D.C., August 2015. (Awarded the **Internet Defense Prize** by Facebook and USENIX)
50. Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. Unraveling the Relationship Between Ad-Injecting Browser Extensions and Malvertising. In *Proceedings of The 24th International World Wide Web Conference (WWW)*. Florence, Italy, May 2015.

51. Chengyu Song, Chao Zhang, Tielei Wang, Wenke Lee, and David Melski. Exploiting and Protecting Dynamic Code Generation. In *Proceedings of The 2015 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February, 2015.
52. Byoungyoung Lee, Chengyu Song, Yeongjin Jang, Tielei Wang, Taesoo Kim, Long Lu, and Wenke Lee. Preventing Use-after-free with Dangling Pointers Nullification. In *Proceedings of The 2015 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February, 2015.
53. Kangjie Lu, Zhichun Li, Vasileios P. Kemerlis, Zhenyu Wu, Long Lu, Cong Zheng, Zhiyun Qian, Wenke Lee, and Guofei Jiang. Checking More and Alerting Less: Detecting Privacy Leakages via Enhanced Data-flow Analysis and Peer Voting. In *Proceedings of The 2015 Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February, 2015.
54. Wei Meng, Xinyu Xing, Anmol Sheth, Udi Weinsberg, and Wenke Lee. Your Online Interests Pwned! A Pollution Attack Against Targeted Advertising. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*. Scottsdale, AZ, November 2014.
55. Yeongjin Jang, Chengyu Song, Simon Chung, Tielei Wang, and Wenke Lee. All-y Attacks: Exploiting Accessibility in Operating Systems. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*. Scottsdale, AZ, November 2014.
56. Billy Lau, Pak Ho Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. Mimesis Aegis: A Mimicry Privacy Shield - A System's Approach to Data Privacy on Public Cloud. In *Proceedings of the 23rd USENIX Security Symposium*. San Diego, CA, August 2014.
57. Tielei Wang, Yeongjin Jang, Yizheng Chen, Pak Ho Chung, Billy Lau, and Wenke Lee. On the Feasibility of Large-Scale Infections of iOS Devices. In *Proceedings of the 23rd USENIX Security Symposium*. San Diego, CA, August 2014.
58. Byoungyoung Lee, Long Lu, Tielei Wang, Taesoo Kim, and Wenke Lee. From Zygote to Morula: Fortifying Weakened ASLR on Android. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2014.
59. Xinyu Xing, Wei Meng, Dan Doozan, Nick Feamster, Wenke Lee, and Alex C. Snoeren. Exposing Inconsistent Web Search Results with Bobble. In *Proceedings of The 2014 Passive and Active Measurement Conference (PAM)*, Los Angeles, CA, March 2014.
60. Yeongjin Jang, Simon P. Chung, Bryan D. Payne, and Wenke Lee. Gyrus: A Framework for User-Intent Monitoring of Text-Based Networked Applications. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2014.
61. Yacin Nadjji, Manos Antonakakis, Roberto Perdisci, and Wenke Lee. Beheading Hydras: Performing Effective Botnet Takedowns. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, November 2013.

62. Brendan Dolan-Gavitt, Tim Leek, Josh Hodosh, and Wenke Lee. Tappan Zee (North) Bridge: Mining Memory Accesses for Introspection. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, November 2013.
63. Xinyu Xing, Wei Meng, Dan Doozan, Alex C. Snoeren, Nick Feamster, and Wenke Lee. Take this Personally: Pollution Attacks on Personalized Services. In *Proceedings of the 22nd USENIX Security Symposium*, Washington, D.C., August 2013.
64. Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, and Wenke Lee. Jekyll on iOS: When Benign Apps Become Evil. In *Proceedings of the 22nd USENIX Security Symposium*, Washington, D.C., August 2013.
65. Yacin Nadji, Manos Antonakakis, Roberto Perdisci, and Wenke Lee. Connected Colors: Unveiling the Structure of Criminal Networks. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, St. Lucia, October 2013 (to appear).
66. Junjie Zhang, Yinglian Xie, Fang Yu, David Soukal, and Wenke Lee. Intention and Origination: An Inside Look at Large-Scale Bot Queries. In *Proceedings of The 20th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2013.
67. Charles Lever, Manos Antonakakis, Bradley Reaves, Patrick Traynor and Wenke Lee. The Core of the Matter: Analyzing Malicious Traffic in Cellular Carriers. In *Proceedings of The 20th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2013.
68. Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS)*, Raleigh, NC. October 2012.
69. Martim Carbone, Matthew Conover, Bruce Montague, and Wenke Lee. Secure and Robust Monitoring of Virtual Machines through Guest-Assisted Introspection. In *Proceedings of the 15th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*. Amsterdam, The Netherlands. September, 2012.
70. Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium*. Bellevue, WA. August 2012.
71. Kapil Singh, Helen Wang, Alexander Moshchuk, Collin Jackson, and Wenke Lee. Practical End-to-End Web Content Integrity. In *Proceedings of the 21st International World Wide Web Conference (WWW)*, Lyon, France, April 2012.
72. Xiapu Luo, Peng Zhou, Junjie Zhang, Roberto Perdisci, Wenke Lee, and Rocky K.C. Chang. Exposing Invisible Timing-Based Traffic Watermarks with BACKLIT. In *Proceedings of The 27th Annual Computer Security Applications Conference (ACSAC 2011)*, Orlando, FL, December 2011.

73. Yacin Nadj, Manos Antonakakis, Roberto Perdisci, and Wenke Lee. Understanding the Prevalence and Use of Alternative Plans in Malware with Network Games. In *Proceedings of The 27th Annual Computer Security Applications Conference (ACSAC 2011)*, Orlando, FL, December 2011.
74. Long Lu, Roberto Perdisci, and Wenke Lee. SURF: Detecting and Measuring Search Poisoning. In *Proceedings of The 18th ACM Conference on Computer and Communications Security (CCS)*. Chicago, IL, October 2011.
75. Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nick Vasiloglou, and David Dagon. Detecting Malware Domains at the Upper DNS Hierarchy. In *Proceedings of The 20th USENIX Security Symposium*, San Francisco, CA August 2011.
76. Junjie Zhang, Roberto Perdisci, Wenke Lee, Unam Sarfraz, and Xiapu Luo. Detecting Stealthy P2P Botnets Using Statistical Traffic Fingerprints. In *Proceedings of The 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011)*, Hong Kong, China, June 2011.
77. Xiapu Luo, Peng Zhou, Edmond W. W. Chan, Rocky K. C. Chang, and Wenke Lee. A Combinatorial Approach to Network Covert Communications with Applications in Web Leaks. In *Proceedings of The 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011)*, Hong Kong, China, June 2011.
78. Brendan Dolan-Gavitt, Tim Leek, Michael Zhivich, Jon Giffin, and Wenke Lee. Virtuoso: Narrowing the Semantic Gap in Virtual Machine Introspection. In *Proceedings of The 2010 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2011.
79. Junjie Zhang, Jay Stokes, Christian Seifert, and Wenke Lee. ARROW: Generating Signatures to Detect Drive-By Downloads. In *Proceedings of The 20th International World Wide Web Conference (WWW)*, Hyderabad, India, March 2011.
80. Junjie Zhang, Xiapu Luo, Roberto Perdisci, Guofei Gu, Wenke Lee, and Nick Feamster. Boosting the Scalability of Botnet Detection Using Adaptive Traffic Sampling. In *Proceedings of The 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Hong Kong, March 2011.
81. Xiapu Luo, Peng Zhou, Edmond W.W. Chan, Wenke Lee, Rocky K. C. Chang, and Roberto Perdisci. HTTPoS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *Proceedings of The 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2011.
82. Qing Hui, Xiapu Luo, and Wenke Lee. Control of Low-Rate Denial-of-Service Attacks on Web Servers and TCP Flows. In *Proceedings of The 49th IEEE Conference on Decision and Control (CDC)*, Atlanta, GA, December 2010.
83. Long Lu, Vinod Yegneswaran, Phil Porras, and Wenke Lee. BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware Infections. In *Proceedings of The 17th ACM Conference on Computer and Communications Security (CCS)*, Chicago, IL, October 2010.

84. Xiapu Luo, Junjie Zhang, Roberto Perdisci, and Wenke Lee. On the Secrecy of Spread-Spectrum Flow Watermarks. In *Proceedings of The 15th European Symposium on Research in Computer Security (ESORICS)*, Athens, Greece, September 2010.
85. Manos Antonakakis, David Dagon, Xiapu Luo, Roberto Perdisci, and Wenke Lee. A Centralized Monitoring Infrastructure for Improving DNS Security. In *Proceedings of The 13th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Ottawa, Ontario, Canada, September 2010.
86. Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a Dynamic Reputation System for DNS. In *Proceedings of The 19th USENIX Security Symposium*, Washington, DC, August 2010.
87. Kapil Singh, Samrit Sangal, Nehil Jain, Patrick Traynor, and Wenke Lee. Evaluating Bluetooth as a Medium for Botnet Command and Control. In *Proceedings of The 7th Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA)*, Bonn, Germany, July 2010.
88. Kapil Singh, Alexander Moshchuk, Helen J. Wang, and Wenke Lee. On the Incoherencies in Web Browser Access Control Policies. In *Proceedings of The 2010 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2010.
89. Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral Clustering of HTTP-based Malware and Signature Generation using Malicious Network Traces. In *Proceedings of The 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, April 2010.
90. Roberto Perdisci, Igino Corona, David Dagon, and Wenke Lee. Detecting Malicious Flux Service Networks through Passive Analysis of Recursive DNS Traces. In *Proceedings of The 25th Annual Computer Security Applications Conference (ACSAC 2009)*, Honolulu, HI, December 2009.
91. Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active Botnet Probing to Identify Obscure Command and Control Channels. In *Proceedings of The 25th Annual Computer Security Applications Conference (ACSAC 2009)*, Honolulu, HI, December 2009.
92. Monirul Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. Secure In-VM Monitoring Using Hardware Virtualization. In *Proceedings of The 16th ACM Conference on Computer and Communications Security (CCS 2009)*, Chicago, IL, November, 2009.
93. Martim Carbone, Weidong Cui, Long Lu, Wenke Lee, Marcus Peinado, and Xuxian Jiang. Mapping Kernel Objects to Enable Systematic Integrity Checking. In *Proceedings of The 16th ACM Conference on Computer and Communications Security (CCS 2009)*, Chicago, IL, November, 2009.
94. Kapil Singh, Sumeer Bhola, and Wenke Lee. xBook: Redesigning Privacy Control in Social Networking Platforms. In *Proceedings of The 18th USENIX Security Symposium*, Montreal, Canada, August, 2009.

95. Roberto Perdisci, Manos Antonakakis, Xiapu Luo, and Wenke Lee. WSEC DNS: Protecting Recursive DNS Resolvers from Poisoning Attacks. In *Proceedings of The 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2009)*, Lisbon, Portugal, June 2009.
96. Monirul Sharif, Andrea Lanzi, Jon Giffin, and Wenke Lee. Automatic Reverse Engineering of Malware Emulators. In *Proceedings of The 2009 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2009. **(Best Student Paper Award)**
97. Andrea Lanzi, Monirul Sharif, and Wenke Lee. K-Tracer: A System for Extracting Kernel Malware Behavior. In *Proceedings of The 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, San Diego, CA, February 2009.
98. David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P. Lee, and Wenke Lee. Recursive DNS Architectures and Vulnerability Implications. In *Proceedings of The 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, San Diego, CA, February 2009.
99. Roberto Perdisci, Andrea Lanzi, and Wenke Lee. McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables. In *Proceedings of The 24th Annual Computer Security Applications Conference (ACSAC 2008)*, Anaheim, CA, December 2008.
100. Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: Malware Analysis via Hardware Virtualization Extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, Alexandria, VA, October 2008.
101. David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased DNS Forgery Resistance Through 0x20-Bit Encoding. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*, Alexandria, VA, October 2008.
102. Monirul Sharif, Vinod Yegneswaran, Hassen Saidi, Phillip Porras, and Wenke Lee. Eureka: A Framework for Enabling Static Malware Analysis. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS)*, Malaga, Spain, October 2008.
103. Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proceedings of The 17th USENIX Security Symposium (Security'08)*, San Jose, CA, July 2008.
104. Kapil Singh, Abhinav Srivastava, Jon Giffin, and Wenke Lee. Evaluating Email's Feasibility for Botnet Command and Control. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008)*, Anchorage, Alaska, June 2008.
105. Bryan D. Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An Architecture for Secure Active Monitoring Using Virtualization. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2008.

106. Guofei Gu, Alvaro A. Cardenas, and Wenke Lee. Principled Reasoning and Practical Applications of Alert Fusion in Intrusion Detection Systems. In *Proceedings of the ACM Symposium on InformAction, Computer and Communications Security (ASIACCS'08)*, Tokyo, Japan, March 2008.
107. David Dagon, Niels Provos, Chris Lee, and Wenke Lee. Corrupted DNS Resolution Paths: The Rise of a Malicious Resolution Authority. In *Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS 2008)*, San Diego, CA, February 2008.
108. Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS 2008)*, San Diego, CA, February 2008.
109. Monirul Sharif, Andrea Lanzi, Jonathon Giffin, and Wenke Lee. Impeding Malware Analysis using Conditional Code Obfuscation. In *Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS 2008)*, San Diego, CA, February 2008.
110. Bryan D. Payne, Martim Carbone, and Wenke Lee. Secure and Flexible Monitoring of Virtual Machines. In *Proceedings of The 23rd Annual Computer Security Applications Conference (ACSAC 2007)*, Miami Beach, FL, December 2007.
111. David Dagon, Guofei Gu, Chris Lee and Wenke Lee. A Taxonomy of Botnet Structures. In *Proceedings of The 23rd Annual Computer Security Applications Conference (ACSAC 2007)*, Miami Beach, FL, December 2007.
112. Guofei Gu, Zesheng Chen, Phillip Porras and Wenke Lee. Misleading and Defeating Importance-Scanning Malware Propagation. In *Proceedings of The 3rd International Conference on Security and Privacy in Communication Networks (SecureComm'07)*, Nice, France, September 2007.
113. Takehiro Takahashi and Wenke Lee. An Assessment of VoIP Covert Channel Threats. In *Proceedings of The 3rd International Conference on Security and Privacy in Communication Networks (SecureComm'07)*, Nice, France, September 2007.
114. Monirul Sharif, Kapil Singh, Jonathon Giffin and Wenke Lee. Understanding Precision in Host Based Intrusion Detection: Formal Analysis and Practical Models. In *Proceedings of The 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Surfers Paradise, Australia, September 2007.
115. Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proceedings of The 16th USENIX Security Symposium (Security'07)*, Boston, MA, August 2007.
116. David Cash, Yan Zong Ding, Yevgeniy Dodis, Wenke Lee, Richard Lipton, and Shabsi Wal-fish. Intrusion-Resilient Key Exchange in the Bounded Retrieval Model. In *Proceedings of The Fourth IACR Theory of Cryptography Conference (TCC 2007)*, Amsterdam, The Netherlands, February 2007.

117. Roberto Perdisci, Guofei Gu, and Wenke Lee. Using an Ensemble of One-Class SVM Classifiers to Harden Payload-based Anomaly Detection Systems. In *Proceedings of The 2006 IEEE International Conference on Data Mining (ICDM '06)*, Hong Kong, China, December 2006.
118. Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *Proceedings of The 22th Annual Computer Security Applications Conference (ACSAC 2006)*, Miami Beach, FL, December 2006.
119. Prahlad Fogla and Wenke Lee. Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques. In *Proceedings of The 13th ACM Conference on Computer and Communications Security (CCS 2006)*, Alexandria, VA, October 2006.
120. Guofei Gu, Prahlad Fogla, David Dagon, Wenke Lee, and Boris Skoric. Towards an Information-Theoretic Framework for Analyzing Intrusion Detection Systems. In *Proceedings of The 11th European Symposium Research Computer Security (ESORICS 2006)*, Hamburg, Germany, September 2006.
121. Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, and Wenke Lee. Polymorphic Blending Attacks. In *Proceedings of The 15th USENIX Security Symposium (SECURITY '06)*, Vancouver, B.C., Canada, August 2006.
122. Collin Mulliner, Giovanni Vigna, David Dagon, and Wenke Lee. Using Labeling to Prevent Cross-Service Attacks Against Smart Phones. In *Proceedings of The 3rd Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2006)*, Berlin, Germany, July 2006.
123. Hongmei Deng, Roger Xu, Jason H. Li, Frank Zhang, Renato Levy, and Wenke Lee. Agent-Based Cooperative Anomaly Detection for Wireless Ad Hoc Networks. In *Proceedings of The 12th International Conference on Parallel and Distributed Systems (ICPADS 2006)*, Minneapolis, Minnesota, July 2006.
124. Guofei Gu, Prahlad Fogla, Wenke Lee, and Douglas Blough. DSO: Dependable Signing Overlay. In *Proceedings of The 4th International Conference on Applied Cryptography and Network Security (ACNS '06)*, Singapore, June 2006.
125. Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Sharif. Misleading Worm Signature Generators Using Deliberate Noise Injection (full paper). In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2006.
126. Guofei Gu, Prahlad Fogla, David Dagon, Wenke Lee, and Boris Skoric. Measuring Intrusion Detection Capability: An Information-Theoretic Approach. In *Proceedings of ACM Symposium on InformAction, Computer and Communications Security (ASIACCS '06)*, Taipei, Taiwan, March 2006.
127. David Dagon, Cliff Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proceedings of The 13th Annual Network and Distributed System Security Symposium (NDSS 2006)*, San Diego, CA, February 2006.

128. Yongguang Zhang, Yi-an Huang, and Wenke Lee. An Extensible Environment for Evaluating Secure MANET. In *Proceedings of The 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, Athens, Greece, September 2005.
129. Jonathon T. Giffin, David Dagon, Somesh Jha, Wenke Lee, and Barton P. Miller. Environment-Sensitive Intrusion Detection. In *Proceedings of The 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, Seattle, WA, September 2005.
130. David Dagon, Wenke Lee, and Richard Lipton. Protecting Secret Data from Insider Attacks. *Proceedings of The Ninth International Conference on Financial Cryptography and Data Security (FC'05)*, Roseau, Dominica, February, 2005.
131. Guofei Gu, David Dagon, Xinzhou Qin, Monirul I. Sharif, Wenke Lee, and George F. Riley. Worm Detection, Early Warning, and Response Based on Local Victim Information. *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, December 2004.
132. Xinzhou Qin and Wenke Lee. Attack Plan Recognition and Prediction Using Causal Networks. *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC 2004)*, Tucson, Arizona, December 2004.
133. Joao B.D. Cabrera, Jaykumar Gosar, Wenke Lee, and Raman K. Mehra. On the Statistical Distribution of Processing Times in Network Intrusion Detection. *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC 2004)*, Bahamas, December 2004.
134. George F. Riley, Monirul I. Sharif, and Wenke Lee. Simulating Internet Worms. *Proceedings of the 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Volendam, The Netherlands, October 2004.
135. Yian Huang and Wenke Lee. Attack Analysis and Detection for Ad Hoc Routing Protocols. *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, Sophia Antipolis, France, September 2004.
136. David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian Grizzard, John Levin, and Henry Owen. HoneyStat: Local Worm Detection Using Honey pots. *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, Sophia Antipolis, France, September 2004.
137. Xinzhou Qin and Wenke Lee. Discovering Novel Attack Strategies from INFOSEC Alerts. *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS 2004)*, Sophia Antipolis, France, September 2004.
138. H. Feng, Jonathon T. Giffin, Yong Huang, Somesh Jha, Wenke Lee, and Barton P. Miller. Formalizing Sensitivity in Static Analysis for Intrusion Detection. *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2004.

139. Xinzhou Qin and Wenke Lee. Statistical Causality Analysis of INFOSEC Alert Data. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, September 2003.
140. H. Feng, Oleg Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly Detection Using Call Stack Information. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.
141. Yi-an Huang, Wei Fan, Wenke Lee, and Philip S. Yu. Cross-Feature Analysis for Detecting Ad-Hoc Routing Anomalies. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS)*, Providence, RI, May 2003.
142. Joao B.D. Cabrera, Wenke Lee, R.K. Prasanth, Lundy Lewis, and Raman K. Mehra. Optimization and Control Problems in Real Time Intrusion Detection. *Proceedings of the 41st IEEE Conference on Decision and Control (CDC 2002)*, Las Vegas, December, 2002.
143. Wenke Lee, Joao B.D. Cabrera, Ashley Thomas, Niranjana Balwalli, Sunmeet Saluja, and Yi Zhang. Performance Adaptation in Real-Time Intrusion Detection Systems. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)*, Zurich, Switzerland, October 2002.
144. Xinzhou Qin, Wenke Lee, Lundy Lewis, and Joao B.D. Cabrera. Integrating Intrusion Detection and Network Management. *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, May 2002.
145. Wei Fan, Matt Miller, Sal Stolfo, Wenke Lee, and Phil Chan. Using Artificial Anomalies to Detect Unknown and Known Network Intrusions. *Proceedings of The First IEEE International Conference on Data Mining (ICDM)*, San Jose, CA, November 2001.
146. Wenke Lee, Sal Stolfo, Phil Chan, Eleazar Eskin, Wei Fan, Matt Miller, Shlomo HersHKop, and Junxin Zhang. Real Time Data Mining-based Intrusion Detection. *Proceedings of the 2001 DARPA Information Survivability Conference and Exposition (DISCEX II)*, Anaheim, CA, June 2001.
147. Wenke Lee and Dong Xiang. Information-Theoretic Measures for Anomaly Detection. *Proceedings of The 2001 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001.
148. Joao B.D. Cabrera, L. Lewis, X. Qin, Wenke Lee, Ravi Prasanth, B. Ravichandran, and Raman Mehra. Proactive Detection of Distributed Denial of Service Attacks Using MIB Traffic Variables - A Feasibility Study. *Proceedings of The Seventh IFIP/IEEE International Symposium on Integrated Network Management (IM 2001)*, Seattle, WA, May 2001.
149. Yongguang Zhang and Wenke Lee. Intrusion Detection in Wireless Ad-Hoc Networks. *Proceedings of The Sixth International Conference on Mobile Computing and Networking (MobiCom 2000)*, Boston, MA, August 2000.
150. Wei Fan, Wenke Lee, Sal Stolfo, and Matt Miller. A Multiple Model Cost-Sensitive Approach for Intrusion Detection. *Proceedings of The Eleventh European Conference on Machine Learning (ECML 2000)*, LNAI 1810, Barcelona, Spain, May 2000.

151. Sal Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Phil Chan. Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX '00)*, Hilton Head, SC, January 2000.
152. Wenke Lee, Sal Stolfo, and Kui Mok. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*, San Diego, CA, August 1999.
153. Wenke Lee, Sal Stolfo, and Kui Mok. A Data Mining Framework for Building Intrusion Detection Models. *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1999.
154. Wenke Lee, Sal Stolfo, and Kui Mok. Mining Audit Data to Build Intrusion Detection Models. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD '98)*, New York, NY, August 1998.
155. Wenke Lee and Sal Stolfo. Data Mining Approaches for Intrusion Detection. *Proceedings of the Seventh USENIX Security Symposium (SECURITY '98)*, San Antonio, TX, January 1998.
156. Sal Stolfo, Andreas Prodromidis, Shelley Tselepis, Wenke Lee, Wei Fan, and Phil Chan. JAM: Java Agents for Meta-learning over Distributed Databases. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD '97)*, Newport Beach, CA, August 1997.
157. Naser S. Barghouti, John Mocenigo, and Wenke Lee. Grappa: A GRAPH PACKage in Java. *Proceedings of the Fifth Annual Symposium on Graph Drawing (Graph Drawing '97)*, Rome, Italy, September 1997.
158. Wenke Lee, Gail Kaiser, Paul Clayton, and Eric Sherman. OzCare: A Workflow Automation System for Care Plans. *Proceedings of the American Medical Informatics Association Annual Fall Symposium*, Washington DC, October 1996.

F. Workshop Presentations (refereed)

F.1. Workshop Presentations with Proceedings

1. Yi-an Huang and Wenke Lee. Hotspot-Based Traceback for Mobile Ad Hoc Networks. In *Proceedings of The ACM Workshop on Wireless Security (WiSe 2005)*, Cologne, Germany, September 2005.
2. Monirul Sharif, George Riley, and Wenke Lee. Comparative Study between Analytical Models and Packet-Level Worm Simulations. In *Proceedings of The 19th Workshop on Parallel and Distributed Simulation (PADS 2005)*, Monterey, CA, June 2005.
3. Chris Clark, Wenke Lee, David Schimmel, Didier Contis, Mohamed Kone, and Ashley Thomas. A Hardware Platform for Network Intrusion Detection and Prevention. *The 3rd Workshop on Network Processors and Applications (NP3)*, Madrid, Spain, February 2004.

4. Yian Huang and Wenke Lee. A Cooperative Intrusion Detection System for Ad Hoc Networks. *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03)*, Fairfax VA, October 2003.
5. Mustaque Ahamad, Wenke Lee, Ling Liu, Leo Mark, Edward Omicienski, Calton Pu, and Andre dos Santos. Guarding the Next Internet Frontier: Countering Denial of Information Attacks. *Proceedings of the 2002 New Security Paradigms Workshop*, Virginia Beach, Virginia, September 2002.
6. Xinzhou Qin, Wenke Lee, Lundy Lewis, and Joao B.D. Cabrera. Using MIB III Variables for Network Anomaly Detection - A Feasibility Study. *ACM Workshop on Data Mining for Security Applications*, Philadelphia, PA, November 2001.
7. Yongguang Zhang, Harrick Vin, Lorenzo Alvisi, Wenke Lee, and Son K. Dao. Heterogeneous Networking: A New Survivability Paradigm. *Proceedings of the 2001 New Security Paradigms Workshop*, Cloudcroft, New Mexico, September 2001.
8. Wenke Lee, Wei Fan, Matt Miller, Sal Stolfo, and Erez Zadok. Toward Cost-Sensitive Modeling for Intrusion Detection and Response. *ACM Workshop on Intrusion Detection Systems*, Athens, Greece, November 2000.
9. Wenke Lee, Rahul Nimbalkar, Kam Yee, Sunil Patil, Pragnesh Desai, Thuan Tran, and Sal Stolfo. A Data Mining and CIDF Based Approach for Detecting Novel and Distributed Intrusions. *Proceedings of The Third International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, LNCS 1907, Toulouse, France, October 2000.
10. Wenke Lee, Chris Park, and Sal Stolfo. Towards Automatic Intrusion Detection using NFR. *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, April 1999.
11. Wenke Lee, Sal Stolfo, and Phil Chan. Learning Patterns from Unix Process Execution Traces for Intrusion Detection. *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997.
12. Sal Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Phil Chan. Credit Card Fraud Detection Using Meta-Learning: Issues and Initial Results. *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997.
13. Gail Kaiser and Wenke Lee. Pay No Attention to the Man Behind the Curtain. *NSF Workshop on Workflow and Process Automation*, May 1996.
14. Wenke Lee. Data Modeling and Management for Large Spatial Databases. *The Third International Workshop in Geographic Information Systems*, Beijing, China, August 1993.

G. Other

1. Matt Blaze, Sampath Kannan, Insup Lee, Oleg Sokolsky, Jonathan Smith, Angelos Keromytis, and Wenke Lee. Dynamic Trust Management. *IEEE Computer*, February 2009.

2. Martim Carbone, Diego Zamboni, and Wenke Lee. Taming Virtualization. *IEEE Security & Privacy*, vol. 6, no. 1, January/February 2008.
3. Bryan D. Payne, Reiner Sailer, Ramon Caceres, Ronald Perez, and Wenke Lee. A Layered Approach to Simplified Access Control in Virtualized Systems. *ACM SIGOPS Operating Systems Review*, 4(2), July 2007.
4. Wenke Lee. 2002. Applying Data Mining to Intrusion Detection: The Quest for Automation, Efficiency, and Credibility. *SIGKDD Explorations*, 4(2), December 2002.
5. Wenke Lee and Wei Fan. 2001. Mining System Audit Data: Opportunities and Challenges. *SIGMOD Record*, 30(4), December 2001.
6. Salvatore J. Stolfo, Wenke Lee, Philip K. Chan, Wei Fan, and Eleazar Eskin. 2001. Data Mining-Based Intrusion Detectors: An Overview of the Columbia IDS Project. *SIGMOD Record*, 30(4), December 2001.

H. Research Proposals and Grants

a. Approved and Funded

1. **Proof-driven Refinement of Infrastructure Software Modules (PRISM)**
Brendan Saltaformaggio (PI); co-PIs: Wenke Lee, Taesoo Kim, Alessandro Orso, Qirun Zhang, Yisroel Mirsky, Martin Osterloh, Jason Li, Michael Brown, Sukarno Mertoguno, Kevin Hamlen, and Nicholas Evancich
DARPA I2O
Funded at \$10,647,759, 7/1/2021-12/1/2025.
2. **D3: Debloating, Dialecting and Diversification for Attack Resilient Software with Real-time Constraints**
Taesoo Kim (PI); co-PI: Wenke Lee
Technology Innovation Institute Sole Proprietorship LLC
Funded at \$1,575,000, 9/1/2020-8/31/2023.
3. **HECTOR: Halting Emergent Computation Through Oracle Reinforcement**
Wenke Lee (PI); co-PIs: Sukarno Mertoguno (GTRI) and Clayton Kerce (GTRI)
DARPA I2O
Funded at \$1,000,000, 1/15/2020-7/15/2021.
4. **Interactive Editing Techniques for Subsetting and Dialecting Network Protocols**
Taesoo Kim (PI); co-PIs: Wenke Lee, Alex Orso, and Brendan Saltaformaggio
ONR
Funded at \$2,976,781, 8/1/2018-7/31/2021
5. **SaTC: CORE: Medium: Understanding and Fortifying Machine Learning Based Security Analytics**
Polo Chau (PI); co-PIs: Wenke Lee, Le Song, and Taesoo Kim
NSF
Funded at \$1,200,000, 8/1/2017-7/31/2021
6. **Comprehensive System Debloating via Path-Based Learning and Late-Stage OS Composition**
Wenke Lee (for Bill Harris) (PI); co-PIs: Bill Harris, Taesoo Kim, Alessandro Orso, and Santosh Pande
ONR
Funded at \$7,500,000, 8/1/2017-7/31/2022
7. **CyberCorps Scholarship-for-Service (SFS)**
Mustaque Ahamad (PI); co-PIs: Wenke Lee, Taesoo Kim, Sy Goodman, and Manos Antonakakis
NSF
Funded at \$5,000,000, 9/1/2016-10/31/2021.

8. **Multi-Layer Adaptive and Proactive Strategic Cyber Defense**
Radha Poovendran (PI, University of Washington); co-PIs: Wenke Lee (Georgia Tech) and Tamer Basar (UIUC)
ARO
Funded at \$1,000,000, 8/1/2016-7/31/2019.
9. **Intel Science & Technology Center for Adversary-Resilient Security Analytics (ISTC-ARSA)**
Wenke Lee (PI); co-PIs: Taesoo Kim, Polo Chau (CSE), and Le Song (CSE)
Funded as a \$1,500,000 gift.
10. **THEIA: Tagging and Tracking of Multi-Level Host Events for Transparent Computing and Information Assurance**
Wenke Lee (PI); co-PIs: Taesoo Kim, Alex Orso, and Simon Chung
DARPA
Funded at \$4,193,126, 6/26/2015-6/26/2019.
11. **ADAPT: Analytical Framework for Actionable Defense against Advanced Persistent Threats**
Radha Poovendran (PI, University of Washington); co-PIs: Wenke Lee (Georgia Tech), Shankar Sastry and Anthony Joseph (UC Berkeley), Joao Hespanha (UCSB), Tamer Basar (UIUC), and Maryam Fazel and Linda Bushnell (University of Washington)
ONR MURI
Funded at \$7,500,000, 6/1/2016-5/31/2021.
12. **Embedasploit: A “Pen-Test in a Box” for Industrial Control Systems**
Wenke Lee (PI); co-PIs: Sal Stolfo at Columbia, Brendan Dolan-Gavitt at NYU Tandon School of Engineering
Office of Naval Research
Funded at \$1,250,351, 5/1/2015-4/30/2018.
13. **BFT++: Attack Tolerance in Hard Real-Time Systems**
Taesoo Kim (PI); co-PI: Wenke Lee
Office of Naval Research
Funded at \$1,245,720, 4/1/2015-3/31/2018.
14. **TWC SBE: TTP Option: Medium: Collaborative: EPICA: Empowering People to Overcome Information Controls and Attacks**
Wenke Lee (PI); co-PIs: Nick Feamster, Hongyuan Zha, and Hans Klein
NSF
Funded at \$1,100,000, 8/1/14-7/31/17.
15. **Comprehensive Understanding of Malicious Overlay Networks**
Homeland Security Advanced Project Research Agency (HSARPA of DHS)
Wenke Lee (PI); co-PIs: David Dagon at Georgia Tech, Chris Smoak of GTRI, Roberto Perdisci (UGA), April Lorenzen of Dissect Cyber, Paul Vixie of Internet Systems Consortium, Jody Westby of Global Cyber Risk, and Matt Jonkman of Open Information Security

Foundation.

Funded at \$1,873,078 for project total, with \$673,078 to CoC/GT, 10/1/12–9/30/15.

16. EAGER: The Conceptual Landscape of Information Manipulation

Wenke Lee (PI); co-PIs: Nick Feamster, and Hans Klein (Public Policy)

NSF

Funded at \$200,000, 8/1/12–7/31/14.

17. Integrating Security Concepts in an Undergraduate Computer Sciences Curriculum

Intel

Wenke Lee (PI); co-PIs: Mustaque Ahamad, Nick Feamster, and Paul Royal

Funded as unrestricted gift at \$50,000, 11/2011;

additional, unrestricted gift at \$35,000, 3/2012;

additional, unrestricted gift at \$50,000, 11/2012;

a total of \$135,000.

18. Monitoring Free and Open Access to Information on the Internet

Google

Nick Feamster and Wenke Lee (co-PIs)

Funded as unrestricted gift: \$1,000,000 base for 12/15/2010–12/14/2012;

\$500,000 option for 12/15/2012–12/14/2013;

additional gift at \$500,000, for 12/15/13–12/14/2014;

a total of \$2,000,000.

19. Dimensions of Network Intelligence

HSARPA of DHS

Wenke Lee (PI) and Paul Royal (co-PI)

Funded at \$1,022,000, 9/2012–9/2017.

20. Dynamic Reputation for DNS

Google

Wenke Lee (PI)

Funded as unrestricted gift at \$75,000, 7/2010.

21. PEASOUP: Preventing Exploits Against Software Of Uncertain Provenance

IARPA/STONESOUP Program (GrammaTech subcontract)

Wenke Lee (co-PI), with David Melski (PI) at GrammaTech, Jack Davidson and John Knight

(co-PIs) at University of Virginia, and Tom Bracewell (co-PI) at Raytheon

Funded at \$13,000,000 for project total, with \$1,695,887 to Georgia Tech, 6/1/2010–5/31/2014.

22. Research in Virtual Machine Monitoring Techniques

Sandia National Labs/Sandia Corp.

Wenke Lee (PI)

Funded at \$50,000, 9/15/2010–9/15/2011.

23. TC: SMALL: A Foundational and Practical Platform for Host Security Applications

NSF CNS/Trustworthy Computing

Wenke Lee (PI)

Funded at \$429,571, 7/15/2010–7/15/2013.

24. Botnet Attribution and Removal: From Axioms to Theory to Practice

ONR MURI

Wenke Lee (PI); Co-PIs: Nick Feamster, and Jon Giffin at Georgia Tech; Farnam Jahanian, Mike Bailey, and Kang Shin at University of Michigan, Ann Arbor; Giovanni Vigna and Chris Kruegel at University of California, Santa Barbara; John Mitchell at Stanford University
Funded: \$7,500,000 project total, \$2,600,000 for Georgia Tech, 5/1/2009–4/30/2014.

25. Collaborative Research: CT-L: CLEANSE: Cross-Layer Large-Scale Efficient Analysis of Network Activities to Secure the Internet

NSF CNS/Cyber Trust

Wenke Lee (PI); Co-PIs: Nick Feamster, Jon Giffin, Mustaque Ahamad, and Xiaoming Huo (ISyE) at Georgia Tech; Mike Reiter and Fabian Monrose at University of North Carolina at Chapel Hill; Farnam Jahanian and Mike Bailey at University of Michigan; Phil Porras and Vinod Yegneswaran at SRI International; and Paul Vixie at Internet Software Consortium.
Funded: \$1,829,297 project total, \$758,297 for Georgia Tech, 9/1/2008–8/31/2012.

26. SMITE: Scalable Monitoring in the Extreme

DARPA

Wenke Lee and Nick Feamster (co-PIs at Georgia Tech); BBN subcontract
Funded: \$574,999 for Georgia Tech, 4/1/2008–3/31/2010

27. Countering Botnets: Anomaly-Based Detection, Comprehensive Analysis, and Efficient Mitigation

DHS HSARPA

Wenke Lee (PI); with Nick Feamster and Jon Giffin at Georgia Tech, David Dagon at Damballa, Rick Wesson and Adam Waters at Support Intelligence, Paul Vixie at Internet Systems Consortium, and Jody Westby at Global Cyber Risk
Funded: \$1,050,730 project total, \$300,000 for Georgia Tech, 4/1/2008–3/31/2010.
Add-on: \$305,066 project total, \$190,066 to Georgia Tech, 7/1/2009–9/30/2010.

28. CRI-IAD: Collaborative Research: Enabling Security and Network Management Research for Future Networks

NSF CRI

Nick Feamster (co-PI), Alex Gray, and Wenke Lee at Georgia Tech, and Z. Morley Mao (co-PI) and Farnam Jahanian at University of Michigan
Funded: \$397,426 for Georgia Tech (3/1/2008–2/28/2011)

29. Collaborative Research: CT-T: Logic and Data Flow Extraction for Live and Informed Malware Execution

NSF CNS/CyberTrust

Wenke Lee (co-PI); co-PIs: Paul Barford at University of Wisconsin at Madison, and Phil Porras at SRI International
Funded: \$220,000 for Georgia Tech (9/1/2007–8/31/2009)

30. **Foundational and Systems Support for Quantitative Trust Management**
Office of Naval Research MURI (University of Pennsylvania subcontract)
Sampath Kannan (PI); co-PIs: Insup Lee, Oleg Sokolsky, Matt Blazes, and Jonathan Smith at University of Pennsylvania, Wenke Lee at Georgia Tech, and Angelos Keromytis at Columbia University.
Funded: \$1,125,000 for Georgia Tech (project total: \$4,000,000) (5/1/2007–4/30/2012)
31. **Collaborative Research: CT-ISG: Modeling and Measuring Botnets**
NSF CNS/CyberTrust
Wenke Lee (PI) and Cliff C. Zou (co-PI) at University of Central Florida
Funded: \$350,000 for project total, Georgia Tech at \$175,000, (8/15/2006-8/15/2009)
32. **CT-ISG: Trusted Passage: Managing Distributed Trust to Meet the Needs of Emerging Applications**
NSF CNS/CyberTrust
Mustaque Ahamad (PI), Wenke Lee (co-PI), and Karsten Schwan (co-PI)
Funded: \$350,000, (8/15/2006-8/15/2009)
33. **Cyber Threat Analytics**
ARO via subcontract from SRI International
Funded: \$80,000, (7/15/2006-7/15/2007)
34. **ARL CTA**
ARL via subcontract from SPARTA/Telcordia
Funded: \$75,000, (7/15/2006-7/15/2007)
35. **Computer Network Defenses Technologies**
AFRL via subcontract from Scientific Systems Co. Funded: \$30,000, (9/13/2006-3/13/2007)
36. **ARO Workshop on Research in Botnets**
ARO
Funded: \$25,000, (4/1/2006-6/1/2007)
37. **Next-Generation Botnet Detection and Response**
DARPA (via ARO)
Funded: \$291,346, (12/15/2005-6/14/07)
38. **An Information-Theoretic Framework for Evaluating and Optimizing Intrusion Detection Performance**
Army Research Office
Funded: \$199,745, (4/1/2005-1/30/07)
39. **Preventing SQL Code Injection by Combining Static and Runtime Analysis, funded by Department of Homeland Security**
Department of Homeland Security
Alex Orso (PI) and Wenke Lee (co-PI)
Funded: \$200,000, (7/1/2005-12/31/2006)

40. **Anomaly and Misuse Detection in Network Traffic Streams – Checking and Machine Learning Approaches**
Office of Naval Research MURI (University of Pennsylvania subcontract)
Sampath Kannan (PI), Insup Lee, Oleg Sokolsky, and Linda Zhao at University of Pennsylvania, Wenke Lee at Georgia Tech, and Diana Spears and William Spears at University of Wyoming.
Funded: \$400,000 for Georgia Tech (\$2,000,000 project total) (7/15/2004–6/1/2006)
41. **Intrusion Detection and Security Management Technologies for Mobile Ad Hoc Networks**
Army Research Lab (Scientific Systems Company Inc. subcontract)
Funded: \$65,000 (2/1/2005-2/1/2006)
42. **A Course Module on Wireless Security**
Microsoft, Curriculum Development Grant
Funded: \$30,000 (unrestricted gift)
43. **Intrusion Detection Techniques for Mobile Ad Hoc Networks**
NSF, CISE/CCR/Trusted Computing
Funded: \$275,000 (8/15/2003-7/31/2006)
REU Funded: \$15,000 (9/10/2004-7/31/2006)
44. **High-Speed Intrusion Detection Sensors**
Army Research Lab (Scientific Systems Company Inc. subcontract)
Funded: \$65,000 (8/7/2003-8/6/2005)
45. **Distributed Intrusion Detection Feasibility Study**
U.S. Government (Columbia University subcontract)
Funded: \$45,000 (6/1/2003-6/30/2004)
46. **Agile Security for Storing Sensitive and Critical Information**
NSF, CISE/Trusted Computing
Mustaque Ahamad (PI), and co-PIs: Wenke Lee, Doug Blough, and H. Venkateswaran
Funded: \$460,000 (8/15/2002-7/31/2005)
47. **CAREER: Adaptive Intrusion Detection Systems**
NSF, CISE/CAREER
Funded: \$350,000 (8/15/2002–7/31/2007)
48. **ITR/SI: Guarding the Next Internet Frontier: Countering Denial of Information**
NSF, ITR
Mustaque Ahamad (PI), and co-PIs: Wenke Lee, Andre dos Santos, Ling Liu, Leo Mark, Ed Omiecinski, and Calton Pu
Funded: \$1,694,769 (9/1/2001–8/31/2006)
49. **Vulnerability Assessment Tools for Complex Information Networks**
Army Research Office MURI (Harvard University subcontract)
Yu-Chi Ho (PI) at Harvard University, co-PIs: Avrom Pfeffer at Harvard University, Christos

G. Cassandras at Boston University, Wei-Bo Gong at University of Massachusetts, Amherst, and Wenke Lee at Georgia Tech

Funded: \$400,000 for Georgia Tech (\$2,000,000 project total) (5/1/2001–4/30/2006)

50. **A Data Mining Approach for Building Cost-sensitive and Light Intrusion Detection Models.**

DARPA, ATO

Wenke Lee (PI), and co-PIs: Salvatore J. Stolfo at Columbia University and Philip Chan at Florida Institute of Technology

Funded: \$854,000 for Georgia Tech (\$2,000,001 project total) (6/1/2000–8/31/2003)

I. Research Honors and Awards

1. IEEE Fellow, 2021.
2. ACM SIGSAC (Special Interest Group on Security, Audit and Control) Outstanding Innovation Award for “contributions to network and systems security, in particular, machine-learning based approaches to security analytics, including tackling intrusion and botnet detection”, 2019.
3. CCS Test of Time Award, The 25th ACM Conference on Computer and Communications Security (CCS), for the paper “Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: Malware Analysis via Hardware Virtualization Extensions” published in CCS 2008 that have had the greatest impact on security research and practice over the past decade, 2018.
4. ACM Fellow, 2017.
5. Distinguished Practical Paper Award, for the paper “Yanick Fratantonio, Chenxiong Qian, Pak Chung, and Wenke Lee. Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop” in Proceedings of The 2017 IEEE Symposium on Security and Privacy, 2017.
6. Internet Defense Prize, awarded by Facebook and USENIX, for the paper “Byoungyoung Lee, Chengyu Song, Taesoo Kim, and Wenke Lee. Type Casting Verification: Stopping an Emerging Attack Vector” in Proceedings of The 24th USENIX Security Symposium, 2015.
7. Cyber Is A Global Sport Award (awarded to David Dagon and Wenke Lee), Department of Homeland Security, Science and Technology Directorate, Cyber Security Division, 2015.
8. Outstanding Achievement in Research Program Development Award, (awarded to Mustaque Ahamad, Wenke Lee, Paul Royal, and David Dagon), Georgia Institute of Technology, 2015.
9. Outstanding Community Service Award, the IEEE Technical Committee on Security and Privacy, 2013
10. Outstanding Faculty Leadership for the Development of GRAs Award, Georgia Institute of Technology, 2012.
11. Sigma Xi Faculty Best Paper Award, Georgia Institute of Technology, 2010.
12. Best Student Paper Award, for the paper “Monirul Sharif, Andrea Lanzi, Jon Giffin, and Wenke Lee. Automatic Reverse Engineering of Malware Emulators” in Proceedings of The 2009 IEEE Symposium on Security and Privacy, 2009.
13. Outstanding Senior Faculty Research Award, College of Computing, Georgia Institute of Technology, 2009.
14. NSF CAREER Award, 2002.

15. Best Paper Award, Applied Research Category, the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99), August 1999 (with Kui Mok and Sal Stolfo).
16. Honorable mention (runner-up) for Best Paper Award, Applied Research Category, the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98), August 1998.
17. Honorable mention (runner-up) for Best Paper Award, Applied Research Category, the 3rd International Conference on Knowledge Discovery and Data Mining (KDD '97), August 1997.

III. SERVICE

A. Professional Activities

A.1 Membership and Activities in Professional Societies

1. Fellow, Association for Computing Machinery (ACM)
2. Fellow, Institute of Electrical and Electronics Engineers (IEEE)

A.2 Journal Editorial Board Activities

1. IEEE Transactions on Dependable and Secure Computing (TDSC) 2010 - 2011.
2. ACM Transactions on Information and System Security (TISSEC) 2005 - 2011.

A.3 Conference Committee Activities

1. The 31st USENIX Security Symposium (2022).
2. The 2020 ACM Conference on Computer and Communications Security.
3. The 2020 IEEE Symposium on Security & Privacy.
4. The 30th USENIX Security Symposium (2021).
5. The 2019 ACM Conference on Computer and Communications Security.
6. The 2019 IEEE Symposium on Security & Privacy.
7. The 2018 ACM Conference on Computer and Communications Security.
8. The 2018 IEEE Symposium on Security & Privacy.
9. The 2018 IEEE European Symposium on Security & Privacy.
10. The 2017 ACM Conference on Computer and Communications Security.
11. The 26th USENIX Security Symposium (Security '17).
12. Co-chair, Security and Privacy Track, WWW 2017.
13. Annual Network and Distributed System Security Symposium (NDSS) 2017.
14. The 25th USENIX Security Symposium (Security '16).
15. The 23rd ACM Conference on Computer and Communications Security (CCS 2016).
16. The 2016 IEEE European Symposium on Security & Privacy.
17. Annual Network and Distributed System Security Symposium (NDSS) 2016.

18. The 22nd ACM Conference on Computer and Communications Security (CCS 2015).
19. Co-chair, Security and Privacy Track, WWW 2015.
20. Annual Network and Distributed System Security Symposium (NDSS) 2015.
21. European Symposium on Research in Computer Security (ESORICS) 2014.
22. The 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2014).
23. Security and Privacy Track, WWW 2014.
24. *PC Chair*, The 2013 IEEE Symposium on Security and Privacy.
25. The 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2013).
26. The 20th ACM Conference on Computer and Communications Security (CCS 2013).
27. The 16th International Symposium on Recent Advances in Intrusion Detection (RAID 2013).
28. The 19th ACM Conference on Computer and Communications Security (CCS 2012).
29. The 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DMIVA 2012).
30. The 7th USENIX Workshop on Hot Topics in Security (HotSec '12).
31. *PC co-Chair*, The 2012 IEEE Symposium on Security and Privacy.
32. Financial Cryptography and Data Security (FC 2012).
33. The 19th Annual Network and Distributed System Security Symposium (NDSS 2012).
34. The 8th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2011).
35. The 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011).
36. The 20th USENIX Security Symposium (Security '11).
37. The 6th USENIX Workshop on Hot Topics in Security (HotSec '11).
38. The 2011 IEEE Symposium on Security and Privacy.
39. The 13th International Symposium on Recent Advances in Intrusion Detection (RAID 2010).
40. The 15th European Symposium on Research in Computer Security (ESORICS 2010).

41. The 7th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2010).
42. Security and Privacy Track, ICDCS 2010.
43. The 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2010).
44. Security and Privacy Track, WWW 2010.
45. *PC Chair*, The 17th Annual Network and Distributed System Security Symposium (NDSS 2010).
46. Financial Cryptography and Data Security (FC 2010).
47. The 25th Annual Computer Security Applications Conference (ACSAC 2009).
48. The 16th ACM Conference on Computer and Communications Security (CCS 2009).
49. 14th European Symposium on Research in Computer Security (ESORICS 2009).
50. The 18th USENIX Security Symposium (Security '09).
51. *PC Chair*, The Second USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), 2009.
52. *PC Co-Chair*, The Second ACM Conference on Wireless Network Security (WiSec), 2009.
53. The 2009 IEEE Symposium on Security and Privacy.
54. The 16th Annual Network and Distributed System Security Symposium (NDSS 2009).
55. The 24th Annual Computer Security Applications Conference (ACSAC 2008).
56. Internet Measurement Conference (IMC) 2008.
57. 13th European Symposium on Research in Computer Security (ESORICS 2008).
58. The 15th ACM Conference on Computer and Communications Security (CCS 2008).
59. The 17th USENIX Security Symposium (Security '08).
60. 3rd USENIX Workshop on Hot Topics in Security (HotSec '08).
61. The 2008 IEEE Symposium on Security and Privacy.
62. The 15th Annual Network and Distributed System Security Symposium (NDSS 2008).
63. The 23rd Annual Computer Security Applications Conference (ACSAC 2007).
64. The 14th ACM Conference on Computer and Communications Security (CCS 2007).
65. The 10th International Symposium on Recent Advances in Intrusion Detection (RAID 2007).

66. The 16th USENIX Security Symposium (Security '07).
67. The 2007 IEEE Symposium on Security and Privacy.
68. The 14th Annual Network and Distributed System Security Symposium (NDSS 2007).
69. The 2nd International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2006).
70. The 15th USENIX Security Symposium (Security '06).
71. The 13th Annual Network and Distributed System Security Symposium (NDSS 2006).
72. The 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005).
73. The 2005 IEEE Symposium on Security and Privacy.
74. The 2005 ACM Workshop on Wireless Security (WiSe 2005).
75. The 2005 IEEE International Conference on Data Mining (ICDM 2005).
76. The Fourteenth International World Wide Web Conference (WWW 2005).
77. The 2005 International Conference on Distributed Computing Systems (ICDCS).
78. The 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC 2004).
79. The 2004 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN 2004).
80. The 2004 ACM Workshop on Wireless Security (WiSe 2004).
81. Steering Committee and the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004).
82. The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004.
83. The 2004 SIAM International Conference on Data Mining, 2004.
84. The 2004 IEEE International Conference on Data Mining (ICDM 2004).
85. The 2004 IEEE Information Assurance Workshop.
86. Co-Chair of the the ICDM Workshop on Data Mining for Computer Security (DMSEC 2003).
87. The 2003 IEEE International Conference on Data Mining (ICDM 2003).
88. The 2003 ACM Workshop on Wireless Security (WiSe 2003).

89. Steering Committee and the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003).
90. The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003.
91. The 2003 IEEE Information Assurance Workshop.
92. Steering Committee for the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002).
93. The 2002 ACM New Security Paradigms Workshop.
94. The 2002 IEEE Symposium on Security and Privacy.
95. *PC Co-Chair*, The 4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001).
96. Program Committee and Organizational Committee for the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000.

B. On-campus Georgia Tech Committees

1. 5-Year Review Committee for the Chair of School of Computer Science, College of Computing, 2016.
2. Faculty Recruiting Committee, School of Computer Science, 2014.
3. Faculty Recruiting Committee, School of Computer Science, 2012-13.
4. RPT Committee, College of Computing, 2010-2012.
5. College of Computing Dean Search Committee, 2009-2010, College of Computing.
6. Chair, Faculty Recruiting Committee, 2008-2009, School of Computer Science, College of Computing.
7. Chair Advisory Committee, 2008-2009, School of Computer Science, College of Computing.
8. Co-Chair, Faculty Recruiting Committee, 2007-2008, School of Computer Science, College of Computing.
9. RPT Executive Committee, 2007-2008, College of Computing.
10. Faculty Recruiting Committee, 2006-2007, College of Computing.
11. Faculty Recruiting Committee, 2005-2006, College of Computing.
12. Undergraduate Curriculum Task Force, 2004-2005, College of Computing.
13. Ph.D. Admissions Committee, 2003-2004, College of Computing.

14. Faculty Recruiting Committee, 2001-2003, College of Computing.

E. Consulting and Advisory Appointments

1. Advisory Board of the NUS-Singtel Cyber Security Research & Development Laboratory, National University of Singapore. 2019-2021.
2. Secure, Accessible & Fair Elections (SAFE) Commission, State of Georgia, 2018-2019
3. Advisory Board of the Faculty of Engineering, The Chinese University of Hong Kong. 2017-
4. Board of Trustees, Pace Academy. 2017-

IV. NATIONAL AND INTERNATIONAL PROFESSIONAL RECOGNITION

A. Patents

1. "Systems and methods for using video for user and message authentication". Simon Chung, Wenke Lee, and Yeongjin Jang. U.S. Patent Number: 10,476,888, November 2019.
2. "Method and System for Detecting Malware". Manos Antonakakis, Roberto Perdisci, Wenke Lee, and Gunter Ollmann. U.S. Patent Number: 10,257,212, April 2019.
3. "Methods and systems for detecting compromised computers", David Dagon, Nick Feamster, Wenke Lee, Robert Edmonds, Richard Lipton, and Anirudh Ramachandran. U.S. Patent Number: 10,044,748, August 2018.
4. "Method and system for detecting malicious and/or botnet-related domain names". Roberto Perdisci, Wenke Lee, and Gunter Ollmann. U.S. Patent Number: 10,027,688, July 2018.
5. "Method and system for network-based detecting of malware from behavioral clustering". Roberto Perdisci and Wenke Lee. U.S. Patent Number: 9,948,671, April 2018.
6. "Method and system for detecting DGA-based malware". Manos Antonakakis, Roberto Perdisci, Wenke Lee, and Nikolaos Vasiloglou. U.S. Patent Number: 9,922,190, March 2018.
7. "Method and system for detecting malicious domain names at an upper DNS hierarchy". Manos Antonakakis, Roberto Perdisci, Wenke Lee, and Nikolaos Vasiloglou. U.S. Patent Number: 9,686,291, June 2017.
8. "Systems and methods of safeguarding user information while interacting with online service providers". Wenke Lee, Sasha Boldyreva, Simon Chung, Billy Lau, and Chengyu Song. U.S. Patent Number: 9,659,189, May 2017.
9. "Method and System for Detecting Malware". Manos Antonakakis, Roberto Perdisci, Wenke Lee, and Gunter Ollmann. U.S. Patent Number: 9,525,699, December 2016.
10. "Method and system for determining whether domain names are legitimate or malicious", Manos Antonakakis, Roberto Perdisci, David Dagon, and Wenke Lee. U.S. Patent Number: 9,516,058, December 2016.
11. "Method and systems for detecting compromised networks and/or computers", David Dagon, Nick Feamster, Wenke Lee, Robert Edmonds, Richard Lipton, and Anirudh Ramachandran. U.S. Patent Number: 9,306,969, April 2016.
12. "Systems and methods for secure in-VM monitoring", Monirul Sharif and Wenke Lee. U.S. Patent Number: 9,129,106, September 2015.
13. "Method and system for network-based detecting of malware from behavioral clustering". Roberto Perdisci, Wenke Lee, and Gunter Ollmann. U.S. Patent Number: 8,826,438, September 2014.

14. "Method and system for detecting malicious domain names at an upper DNS hierarchy". Manos Antonakakis, Roberto Perdisci, Wenke Lee, and Nikolaos Vasiloglou. U.S. Patent Number: 8,631,489, January 2014.
15. "Method and System for Detecting Malware". Manos Antonakakis, Roberto Perdisci, Wenke Lee, and Gunter Ollmann. U.S. Patent Number: 8,578,497, November 2013.
16. "Method and System for Detecting and Responding to Attacking Networks". David Dagon, Nick Feamster, Wenke Lee, Robert Edmonds, Richard Lipton, and Anirudh Ramachandran. U.S. Patent Number: 8,566,928, October 2013.
17. "Methods for Cost-Sensitive Modeling for Intrusion Detection and Response". Wei Fan, Wenke Lee, Matt Miller, and Sal Stolfo. U.S. Patent Number: 7,818,797, October 2010.
18. "Method and System for Using Intelligent Agents for Financial Transactions, Services, Accounting, and Advice". Dan Schutzer, Will Foster, Huanrui Hu, Wenke Lee, Sal Stolfo, and Wei Fan. U.S. Patent Number: 5,920,848, July 1999.

V. OTHER CONTRIBUTIONS

A. Seminar Presentations

1. “Machine Learning and Security: The Good, The Bad, and The Ugly”, Distinguished Lecture at Dartmouth College, April, 2021.
2. “Machine Learning and Security: The Good, The Bad, and The Ugly”, Distinguished Lecture at Ohio State University, January, 2021.
3. “Machine Learning and Security: The Good, The Bad, and The Ugly”, Keynote at the 2020 ACM Conference on Computer and Communications Security, November, 2021.
4. “Security Overlay”, Stony Brook University, March 2016.
5. “Security Overlay”, Princeton University, October 2015.
6. “Internet Monitoring via DNS Traffic Analysis”, Google-UMD Cybersecurity Seminar, November 2012.
7. “Botnet Detection and Response: Challenges and Opportunities”, Carnegie Mellon University, September 2007.
8. “Botnet Detection and Response”, Indiana University, March 2007.
9. “Botnet Detection and Response”, University of Pittsburgh, October 2006.
10. “Architecture Considerations for Anomaly Detection”, University of Texas at San Antonio, November, 2005
11. “Advanced Intrusion Detection Techniques”, National Security Research Institute, Daejeon, South Korea, February, 2005
12. “U.S. Policy and Recent Detection Techniques for Cyber Threats and Attacks”, National Cyber Security Center, Seoul South Korea, February, 2005
13. “Architecture Considerations for Anomaly Detection”, Purdue University, February, 2005
14. “Architecture Considerations for Anomaly Detection”, Graduate Center of The City University of New York, February, 2005
15. “Architecture Considerations for Anomaly Detection”, The University of Alabama, November, 2004
16. “Architecture Considerations for Anomaly Detection”, Carnegie Mellon University, September, 2004
17. “Architecture Considerations for Anomaly Detection”, University of Wisconsin at Madison, August, 2004

18. "Local Worm Detection and Response: Algorithms and Analytical Models", Lawrence Livermore National Laboratory, June 2004
19. "Intrusion Detection", Carnegie Mellon University, October 2003
20. "Algorithms for Recognizing New Attack Step Relationships", University of Pennsylvania, September 2003
21. "Intrusion Detection", Emory University, October 2001
22. "Developing Data Mining Techniques for Intrusion Detection: A Progress Report", Center for Education and Research in Information Assurance and Security, Purdue University, October 2000
23. "A Data Mining Framework for Building Intrusion Detection Models", Institute for Security Technology Studies, Dartmouth College, Hanover, NH, July 2000
24. "A Data Mining Framework for Building Intrusion Detection Models", Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, February 2000
25. "A Data Mining Framework for Building Intrusion Detection Models", HRL Laboratories, Malibu, CA, November 1999

PREMIERE ISSUE
TRS-80 MODEL 100*

PORTABLE **100**

*The magazine for
Model 100 users*

**September
1983
\$2.95**

THE 100...A LEGEND BEFORE ITS TIME

MANAGE YOUR STOCK PORTFOLIO
WRITERS' TOOLS: FORMAT, FIND, COUNT

COLUMNS BY:
TANDY'S BILL WALTERS
COMPUSERVE'S BILL LOUDEN

PLUS FREE CIS PROGRAMS



★ **Tandy Corporation**

TELECOMPUTING



INCREASE YOUR 100'S STORAGE WITH 128K FROM COMPUSE

Having access to 128K of storage from any phone in North America would be a real plus for any portable computer owner, but for the Model 100 owner with CompuServe it's a reality.

CompuServe provides every subscriber with 128,000 characters of disk space. He or she can use the space to send or retrieve electronic mail, store notes, and even archive Model 100 Basic programs.

I recently attended a two-day seminar in New York on personal computing. While everyone took notes with a pencil, I took mine on my Model 100! I filled my 100's RAM by noon, but with one local phone call, I was able to transmit my notes in less than three minutes to my disk space in Columbus, OH, where I later retrieved them. The Model 100 with its built-in modem and TELCOM made the connection easy.

First, access CompuServe. When you purchase a Model 100 cable from Tandy (Radio Shack catalog no. 26-1410), you get one free hour on CompuServe.

ACCESSING CIS. The Model 100 offers two methods for accessing CIS — direct-connect modem and acoustic coupler. Since the 100's acoustic coupler isn't available as I write this, I'll confine myself to the direct connect access method.

The connection to the phone line is fast and easy. Take the direct-connect modem cable and insert the computer connector (the 8-pin off-white DIP plug) into the jack marked phone at the rear of the 100.

Make sure the DIR/ACP (direct/acoustic coupler) switch on the left side of the computer is on DIR and the ORG/ANS (originate/answer) switch is on ORG. Disconnect the telephone line from the telephone — *not from the wall!* Insert the telephone line into the modem cable's beige connector and the cable's silver wire into the telephone.

CONNECTING TO CIS. Enter the TELCOM program, then press the call key (F2). Type in the CIS telephone number and include the characters after it. Press enter. The Model 100 will display "Calling" while it dials the number. After you're connected, you'll hear a beep and the 100 will enter its terminal program. You can access CIS through Tymenet or Telenet, but I'll be limiting myself to a direct connection to CIS. After entering TERM, hold the control key down and press C. You will then be asked for your user identification. Once that is entered you will be asked for your password. Your password will not appear on the screen for your security.

Your screen display will appear similar to figure 1.

To go directly to your disk space, enter EXI at the function prompt (!). Your screen will display OK. You are now in your disk storage area. To look at any files present, enter DIR for a directory. Your directory will be displayed in the format in figure 2.

The first column is your file name and extension; the second, characters used for the file; and the last, the creation date. The format of Com-

puServe file names are similar to Model 100 file names. A maximum of six characters may be used for the file name and an optional maximum of three characters may be used for an extension.

100 TO CIS. How do you get files from your Model 100 to CompuServe? Easy! CompuServe provides a file generator editor called FILGE. Think of FILGE as the bucket you'll pour your Model 100 file into. First enter FILGE file name.extension of the file you wish to create. FILGE will respond:

New file TEST.DAT created - ready

You have now opened the file and may transmit your Model 100 file to CompuServe.

Press the upload key (F3). The message "File to Upload?" will appear on your screen. Enter the file name to send to CompuServe and press enter. The prompt "Width?" will appear. Enter a width for your file. CompuServe requires a car-

9:34 EDT Monday 16-May-83

CompuServe Information Service

- 1 Home Services
- 2 Business & Financial
- 3 Personal Computing
- 4 Services for Professionals
- 5 User Information
- 6 Index

Enter your selection number,
or H for more information
!

Figure 1.

MOD100.DAT 3200 16-May-83

ADDRESS 3200 01-May-83

EAST.DO 9600 09-May-83

Figure 2.

riage return prior to the 132nd character you send it. To avoid a loss of data, enter an output width as if you were printing a report. I normally use a width of 64 since I have a Model III at home to access CIS.

The CompuServe FILGE file will now be filled with your Model 100 file. Actually, the FILGE file thinks you're manually typing in the file at 30 characters a second! Don't type on the Model 100 keyboard while you're uploading or it will be included in your file.

UP will be displayed in reverse video during the upload process. When the UP changes to normal video, the file has been transmitted. All that's necessary now is to close the FILGE file.

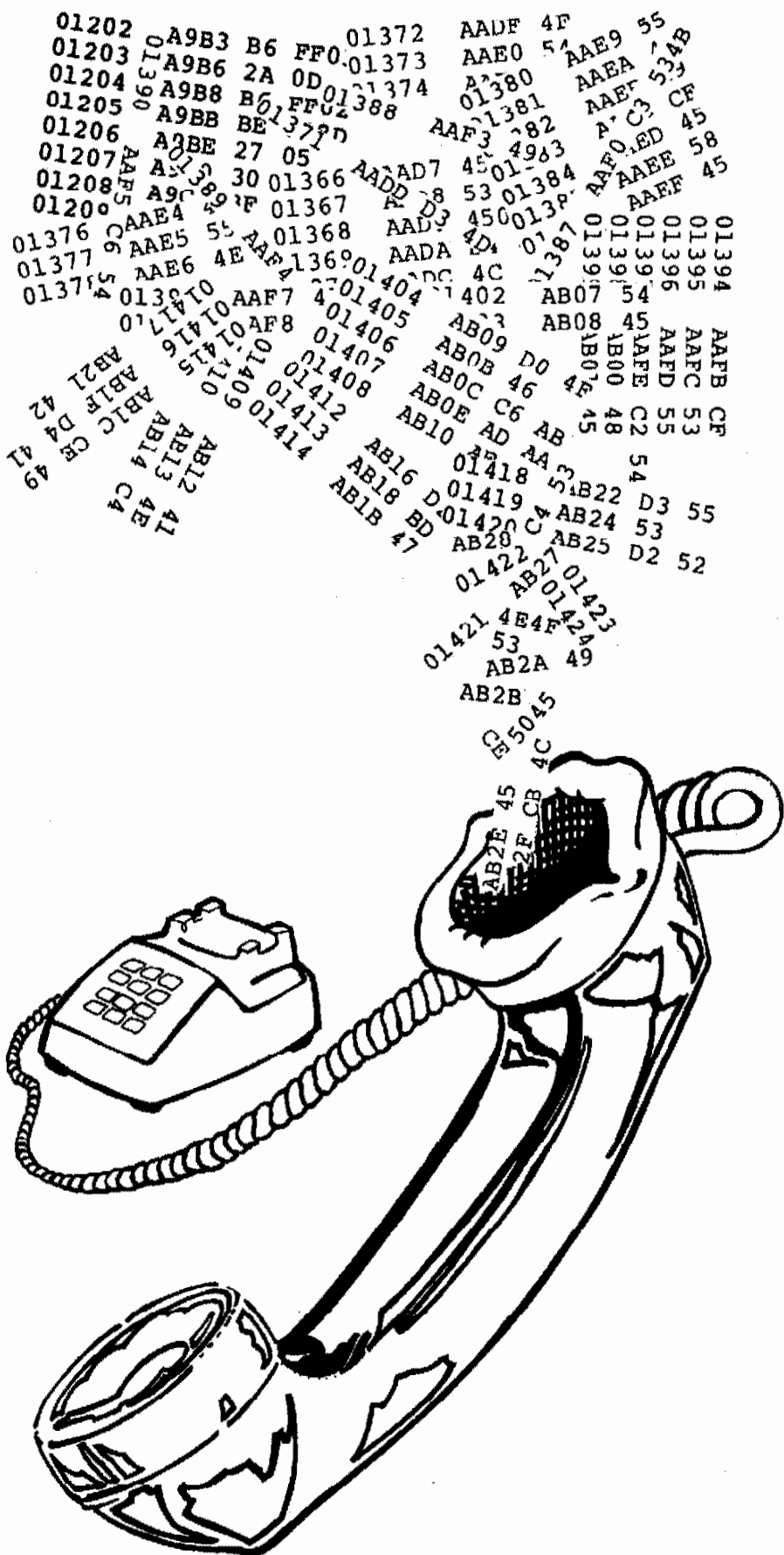
Enter a /EX on a new line. The slash (/) as the first character of a new line tells FILGE what follows is a command. The command /EX will close the file and take you back to OK.

DOWNLOADING. Downloading from CompuServe is even easier. Once you are at OK, enter TYP and the name and extension of the file you wish to download. Before you press enter, press DOWN (F2). Your Model 100 will prompt "File to Download?" Enter a file name for your 100. Press enter to begin downloading the CompuServe file. The down label will also appear in reverse video until the download is complete.

You may exit your disk area and return to the CompuServe menu mode by typing EX1 followed by enter. To disconnect from CompuServe, type BYE followed by enter. Then press BYE (F8) on your 100.

The file you downloaded may require minor editing on your Model 100. Use the editor and position it to the top of the file that should be deleted. This was the enter that started the download process. Also at the end of the file you should see a blank line and an OK. These should also be deleted.

Up- or downloading a Basic program is just as easy as a text file. The only difference is you press enter when asked for the width. However, if you think the Basic program has lines over 132 characters in length, enter 132 for the width and edit the Basic file on your Model 100 later. ▼





[Back to the homepage](#)

Dropbox Features

File Sync

Dropbox allows you to sync your files online and across your computers automatically.

2GB of online storage for free, with up to 100GB available to paying customers.

Sync files of any size or type.

Sync Windows, Mac and Linux computers.

Automatically syncs when new files or changes are detected.

Work on files in your Dropbox even if you're offline. Your changes sync once your computer has an Internet connection again.

Dropbox transfers will correctly resume where they left off if the connection drops.

Efficient sync - only the pieces of a file that changed (not the whole file) are synced. This saves you time.

Doesn't hog your Internet connection. You can manually set bandwidth limits.

File Sharing

Sharing files is simple and can be done with only a few clicks.

Shared folders allow several people to collaborate on a set of files.

You can see other people's changes instantly.

A "Public" folder that lets you link directly to files in your Dropbox.

Control who is able to access shared folders (including ability to kick people out and remove the shared files from their computers).

Automatically create shareable online photo galleries from folders of photos in your Dropbox.

Online Backup

Dropbox backs up your files online without you having to think about it.

Automatic backup of your files.

Undelete files and folders.

Restore previous versions of your files.

30 days of undo history, with unlimited undo available as a paid option.

Web Access

A copy of your files are stored on Dropbox's secure servers. This lets you access them from any computer or mobile device.

Manipulate files as you would on your desktop - add, edit, delete, rename etc.

Search your entire Dropbox for files.

A "Recent Events" feed that shows you a summary of activity in your Dropbox.

Create shared folders and invite people to them.

Recover previous versions of any file or undelete deleted files.

View photo galleries created automatically from photos in your Dropbox.

Security & Privacy

Dropbox takes the security and privacy of your files very seriously.

Shared folders are viewable only by people you invite.

All transmission of file data and metadata occurs over an encrypted channel (SSL).

All files stored on Dropbox servers are encrypted (AES-256) and are inaccessible without your account password.

Dropbox website and client software have been hardened against attacks from hackers.

Dropbox employees are not able to view any user's files.

Online access to your files requires your username and password.

Public files are only viewable by people who have a link to the file(s). Public folders are not browsable or searchable.

Mobile Device Access

The free Dropbox application for [iPhone](#), [iPad](#), and [Android](#) lets you:

Access your Dropbox on the go.

View files from within the application.

Download files for offline viewing.

Take photos and videos and sync them to your Dropbox.

Share links to files in your Dropbox.

Export your files to other applications.

Sync downloaded files so they're up-to-date.

A [mobile-optimized version of the website](#) is also available for owners of Blackberry phones and other Internet-capable mobile devices.

Find out about the [system requirements to run Dropbox](#).

Dropbox

Home
Install
Mobile
Pricing
Features
Tour

Community

Referrals
Twitter
Facebook
Wiki
Developers
Partners

Support

Help Center
Forums
Votebox
Feedback
Contact Us

About Us

Dropbox Blog
Our Team
Press
Policies
Jobs

© 2010 Dropbox

A Low-bandwidth Network File System

Athicha Muthitacharoen, Benjie Chen, and David Mazières

MIT Laboratory for Computer Science and NYU Department of Computer Science

{athicha,benjie}@lcs.mit.edu, dm@cs.nyu.edu

Abstract

Users rarely consider running network file systems over slow or wide-area networks, as the performance would be unacceptable and the bandwidth consumption too high. Nonetheless, efficient remote file access would often be desirable over such networks—particularly when high latency makes remote login sessions unresponsive. Rather than run interactive programs such as editors remotely, users could run the programs locally and manipulate remote files through the file system. To do so, however, would require a network file system that consumes less bandwidth than most current file systems.

This paper presents LBFS, a network file system designed for low-bandwidth networks. LBFS exploits similarities between files or versions of the same file to save bandwidth. It avoids sending data over the network when the same data can already be found in the server's file system or the client's cache. Using this technique in conjunction with conventional compression and caching, LBFS consumes over an order of magnitude less bandwidth than traditional network file systems on common workloads.

1 Introduction

This paper describes LBFS, a network file system designed for low-bandwidth networks. People typically run network file systems over LANs or campus-area networks with 10 Mbit/sec or more bandwidth. Over slower, wide-area networks, data transfers saturate bottleneck links and cause unacceptable delays. Interactive programs freeze, not responding to user input during file I/O, batch commands can

take many times their normal execution time, and other less aggressive network applications are starved for bandwidth. Users must therefore employ different techniques to accomplish what over the LAN they would do through the file system.

People often have occasion to work over networks slower than LANs. Even with broadband Internet access, a person working from home usually only has a fraction of a Mbit/sec of upstream bandwidth. A company with offices in several cities may have many users collaborating over a single 1.5 Mbit/sec T1 line. A consultant constantly traveling between various sites may want to sit down and access the same project files from every location.

In the absence of a network file system, people generally resort to one of two methods of accessing remote data. They either make and edit local copies of files, running the risk of an update conflict, or else they use remote login to view and edit files in place on another machine. If the network has long latency, remote login is particularly frustrating as interactive applications are slow in responding to user input. Worse yet, many graphical applications, such as figure editors and postscript previewers, consume too much bandwidth to run practically over the wide-area network.

Network file systems have the potential to alleviate the inconveniences associated with remote data access. In addition to offering the interface people already prefer for local-area networks, a file system can provide tight consistency, avoiding the problem of conflicts when two people update the same file. File systems can also better tolerate network latency than remote login sessions. By running interactive programs locally and accessing remote data through a file system, one avoids the overhead of a network round trip on every user input event.

To be practical over the wide-area network, however, a file system must consume significantly less bandwidth than most current file systems, both to maintain acceptable performance and to avoid monopolizing network links in use for other purposes. Unfortunately, application writers commonly assume that file I/O will be no slower than a megabyte or so per second. For instance, an interactive editor will stop to write out 100 KByte “auto-save” files without worrying about delaying a user's typing or consuming significant resources. A traditional file system transmits the entire con-

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Space and Naval Warfare Systems Center, San Diego, under contract N66001-00-1-8927.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SOSP01 Banff, Canada
© 2001 ACM ISBN 1-58113-389-8-1/01/10...\$5.00

tents of such files over the network, blocking the editor for the duration of the transfer. In contrast, LBFS often transmits far less data than applications write, greatly reducing the time spent waiting for file I/O.

To reduce its bandwidth requirements, LBFS exploits cross-file similarities. Files written out by applications often contain a number of segments in common with other files or previous versions of the same file. Auto-save files are only one example. Object files output by compilers, temporary files used by the RCS revision control system, postscript files, and word processing documents often contain substantial similarity from one revision to the next. Any copying or concatenation of files, such as when building program libraries out of object files, also leads to significant duplication of contents.

To exploit these inter-file similarities, the LBFS file server divides the files it stores into chunks and indexes the chunks by hash value. The LBFS client similarly indexes a large persistent file cache. When transferring a file between the client and server, LBFS identifies chunks of data that the recipient already has in other files and avoids transmitting the redundant data over the network. In conjunction with conventional compression, this technique saves over an order of magnitude of communications bandwidth on many common workloads.

LBFS provides traditional file system semantics and consistency. Files reside safely on the server once closed, and clients see the server's latest version when they open a file. Thus, LBFS can reasonably be used in place of any other network file system without breaking software or disturbing users. Other file systems have dealt with slow and even intermittent network connectivity by relaxing file system consistency. These techniques largely complement LBFS's, and could be combined with LBFS for even greater bandwidth savings. However, since altered semantics may not be suitable for all purposes, we chose to focus on reducing bandwidth to see just how much we could save without changing accepted consistency guarantees.

The next section describes related work. Section 3 gives LBFS's algorithm for finding commonality between files and explains how the LBFS protocol takes advantage of it. Section 4 describes the implementation of LBFS. Section 5 shows how effective LBFS's technique for compressing file traffic can be. Finally, Section 6 concludes.

2 Related Work

Past projects have attacked the problem of network file systems on slow networks from several angles. LBFS complements most previous work. Because it provides consistency and does not place significant hardware or file system structure requirements on the server, LBFS's approach can unobtrusively be combined with other techniques to get additional savings in network bandwidth.

A number of file systems have properties that help them tolerate high network latency. AFS [9] uses server callbacks to inform clients when other clients have modified cached files. Thus, users can often access cached AFS files without requiring any network traffic. Leases [7] are a modification to callbacks in which the server's obligation to inform a client of changes expires after a certain period of time. Leases reduce the state stored by a server, free the server from contacting clients who haven't touched a file in a while, and avoid problems when a client to which the server has promised a callback has crashed or gone off the network. The NFS4 protocol [20] reduces network round trips by batching file system operations. All of the above techniques are applicable to LBFS. In fact, LBFS currently uses leases and a large, persistent cache to provide AFS-like close-to-open consistency.

Many file systems use write-behind to tolerate latency. Echo [14] performs write-behind of metadata operations, allowing immediate completion of operations that traditionally require a network round trip. In JetFile [8], the last machine to write a file becomes the file's server, and can transmit its contents directly to the next reader.

The CODA file system [10] supports slow networks and even disconnected operation. Changes to the file system are logged on the client and written back to the server in the background when there is network connectivity. To implement this functionality, CODA provides weaker-than-traditional consistency guarantees. It allows update conflicts, which users may need to resolve manually. CODA saves bandwidth because it avoids transferring files to the server when they are deleted or overwritten quickly on the client. LBFS, in contrast, simply reduces the bandwidth required for each file transfer. Thus, LBFS could benefit from CODA-style deferred operations, and CODA could benefit from LBFS file transfer compression.

Bayou [18] further investigates conflict resolution for optimistic updates in disconnected systems, but unlike CODA, it does not provide a file system. Rather, Bayou supplies an API with which to implement application-specific merging and conflict resolution. OceanStore [2] applies Bayou's conflict resolution mechanisms to a file system and extends it to work with untrusted servers that only ever see data in encrypted format. TACT [25] explores the spectrum between absolute consistency and Bayou's weaker model.

Lee et. al. [12] have extended CODA to support operation-based updates. A proxy-client strongly connected to the server duplicates the client's computation in the hopes of duplicating its output files. Users run a modified shell that bundles up commands for the proxy-client to reexecute. Using forward error correction, the client and proxy-client can even patch up small glitches in the output files, such as different dates. When successful, operation-based updates deliver a tremendous bandwidth savings. However, the technique is fairly complementary to LBFS. LBFS works well with in-

teractive applications such as editors that would be hard to reexecute on a proxy-client. Operation-based updates can reduce communications bandwidth with command-line utilities such as image converters for which LBFS offers no savings. Operation-based updates require a dedicated proxy-client machine, making them a bit cumbersome to set up. Perhaps for this reason the technique is not in widespread use by any file system today.

Spring and Wetherall have proposed a protocol-independent technique for eliminating redundant network traffic [21]. They assume two cooperating caches at either end of a slow network link. Both caches store identical copies of the last n Megabytes of network traffic (for values of n up to 100). When one end must send data that already exists in the cache, it instead sends a token specifying where to find the data in the cache. To identify redundant traffic, the two ends index cache data by 64-byte anchors [13], randomly chosen based on hash value. When data to be sent has a 64-byte anchor in common with previous traffic, the matching region is expanded in both directions to elide the greatest amount of data. LBFS's approach is similar in spirit to the Spring and Wetherall technique. However, LBFS supports multiple clients accessing the file system and even local users changing the file system underneath the server. Thus, it cannot assume that the client and server have identical state.

Rsync [23] copies a directory tree over the network onto another directory tree containing similar files—typically from a previous version of the same tree. Rsync saves bandwidth by exploiting commonality between files. The problem is similar to synchronizing a client's file cache with the server or vice versa. In fact, Tridgell suggests applying rsync to a file system in his thesis. Though rsync was one of the inspirations for LBFS, file caching in real time is somewhat different from directory tree mirroring. LBFS thus uses a different algorithm. We discuss the rsync algorithm in more detail and compare it to our approach in Section 3.1.

A number of Unix utilities operate on differences between files. `diff` computes the difference between two text files. `patch` applies the output of `diff` to transform one file into the other. There have been studies of the problem of describing one file in terms of a minimal set of edits to another [22]. Mogul et. al. [17] have investigated transmitting such deltas to save bandwidth when updating cached web pages. The CVS [1] version management system ships patches over the network to bring a user's working copy of a directory tree up to date. Unlike CVS, however, a file system cannot store a complete revision history for all files. Therefore, the LBFS server will typically not have an exact old version from which to compute differences.

3 Design

LBFS is designed to save bandwidth while providing traditional file system semantics. In particular, LBFS provides

close-to-open consistency. After a client has written and closed a file, another client opening the same file will always see the new contents. Moreover, once a file is successfully written and closed, the data resides safely at the server. These semantics are similar to those of AFS. Other work exploring relaxed consistency semantics may apply to LBFS, but we wished to build a file system that could directly substitute for a widely accepted network file system in use today.

To save bandwidth, LBFS uses a large, persistent file cache at the client. LBFS assumes clients will have enough cache to contain a user's entire working set of files (a reasonable assumption given the capacities of cheap IDE disks today). With such aggressive caching, most client-server communication is solely for the purpose of maintaining consistency. When a user modifies a file, the client must transmit the changes to the server (since in our model the client might crash or be cut from the network). Similarly, when a client reads a file last modified by a different client, the server must send it the latest version of the file.

LBFS reduces bandwidth requirements further by exploiting similarities between files. When possible, it reconstitutes files using chunks of existing data in the file system and client cache instead of transmitting those chunks over the network. Of course, not all applications can benefit from this technique. A worst case scenario is when applications encrypt files on disk, since two different encryptions of the same file have no commonality whatsoever. Nonetheless, LBFS provides significant bandwidth reduction for common workloads.

For the remainder of this section, we first discuss the issues involved in indexing chunks of the file system and cache data. We describe the advantages and disadvantages of several approaches, including LBFS's particular solution. Then, we describe the actual LBFS protocol and its use of chunk indexes.

3.1 Indexing

On both the client and server, LBFS must index a set of files to recognize data chunks it can avoid sending over the network. To save chunk transfers, LBFS relies on the collision-resistant properties of the SHA-1 [6] hash function. The probability of two inputs to SHA-1 producing the same output is far lower than the probability of hardware bit errors. Thus, LBFS follows the widely-accepted practice of assuming no hash collisions. If the client and server both have data chunks producing the same SHA-1 hash, they assume the two are really the same chunk and avoid transferring its contents over the network.

The central challenge in indexing file chunks to identify commonality is keeping the index a reasonable size while dealing with shifting offsets. As an example, one could index the hashes of all aligned 8 KByte data blocks in files. To transfer a file, the sender would transmit only hashes of the file's blocks, and the receiver would request only blocks not

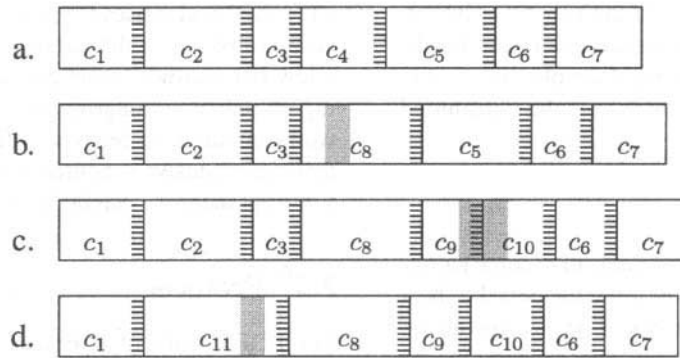


Figure 1: Chunks of a file before and after various edits. Horizontal stripes show 48-byte regions with magic hash values creating chunk boundaries. Gray shading shows regions of the file that were changed by an edit.

already in its database. Unfortunately, a single byte inserted at the start of a large file would shift all the block boundaries, change the hashes of all the file's blocks, and thereby thwart any potential bandwidth savings.

As an alternative, one might index files by the hashes of all (overlapping) 8 KByte blocks at all offsets. Such a scheme would require storage many times the size of the indexed files (almost one index entry per byte of file data). The size in itself might be tolerable given large enough disks, but every file modification might require thousands of index insertions. The cost of performing so many updates to an index in secondary storage would be prohibitive.

Rsync more practically tackles this problem by considering only two files at a time. When transferring file F from machine A to machine B , if B already has a file F' by the same name, rsync guesses the two files may be similar and attempts to exploit that fact. A simplified version of the rsync algorithm proceeds as follows. First, the recipient, B , breaks its file F' into non-overlapping, contiguous, fixed-size blocks. B transmits hashes of those blocks to A . A in turn begins computing the hashes of all (overlapping) blocks of F . If any of those hashes matches one from F' , A avoids sending the corresponding sections of F , instead telling B where to find the data in F' .

A couple of complications arise when trying to apply the rsync algorithm to a file system, however. First, rsync's choice of F' based on filename is too simple. For example, when editing file `foo`, emacs creates an auto-save file named `#foo#`. RCS uses even less suggestive temporary file names such as `_1v22825`. Thus, the recipient would have to choose F' using something other than file names. It might select F' based on a fixed-size "sketch" of F , using Broder's resemblance estimation technique [4]. However, even ignoring the additional cost of this approach, sometimes F can best be reconstructed from chunks of multiple files—consider `ar`, which outputs software libraries containing many object files.

3.1.1 LBFS Solution

In order to use chunks from multiple files on the recipient, LBFS takes a different approach from that of rsync. It considers only non-overlapping chunks of files and avoids sensitivity to shifting file offsets by setting chunk boundaries based on file contents, rather than on position within a file. Insertions and deletions therefore only affect the surrounding chunks. Similar techniques have been used successfully in the past to segment files for the purpose of detecting unauthorized copying [3].

To divide a file into chunks, LBFS examines every (overlapping) 48-byte region of the file and with probability 2^{-13} over each region's contents considers it to be the end of a data chunk. LBFS selects these boundary regions—called *breakpoints*—using Rabin fingerprints [19]. A Rabin fingerprint is the polynomial representation of the data modulo a predetermined irreducible polynomial. We chose fingerprints because they are efficient to compute on a sliding window in a file. When the low-order 13 bits of a region's fingerprint equal a chosen value, the region constitutes a breakpoint. Assuming random data, the expected chunk size is $2^{13} = 8192 = 8$ KBytes (plus the size of the 48-byte breakpoint window). As will be discussed in Section 5.1, we experimented with various window sizes and found that 48 bytes provided good results (though the effect of window size was not huge).

Figure 1 shows how LBFS might divide up a file and what happens to chunk boundaries after a series of edits. **a.** shows the original file, divided into variable length chunks with breakpoints determined by a hash of each 48-byte region. **b.** shows the effects of inserting some text into the file. The text is inserted in chunk c_4 , producing a new, larger chunk c_8 . However, all other chunks remain the same. Thus, one need only send c_8 to transfer the new file to a recipient that already has the old version. Modifying a file can also change the number of chunks. **c.** shows the effects of inserting data that contains a breakpoint. Bytes are inserted in c_5 , splitting that chunk into two new chunks c_9 and c_{10} . Again, the

file can be transferred by sending only the two new chunks. Finally, **d.** shows a modification in which one of the breakpoints is eliminated. Chunks c_2 and c_3 of the old file are now combined into a new chunk, c_{11} , which must be transmitted to compose the new file.

3.1.2 Pathological Cases

Unfortunately, variable-sized chunks can lead to some pathological behavior. If every 48 bytes of a file happened to be a breakpoint, for instance, the index would be as large as the file. Worse yet, hashes of chunks sent over the wire would consume as much bandwidth as just sending the file. Conversely, a file might contain enormous chunks. In particular, the Rabin fingerprint has the property that a long extent of zeros will never contain a breakpoint. As discussed later in Section 3.2, LBFS transmits the contents of a chunk in the body of an RPC message. Having arbitrary size RPC messages would be somewhat inconvenient, since most RPC libraries hold messages in memory to unmarshal them.

To avoid the pathological cases, LBFS imposes a minimum and maximum chunk size. The minimum chunk size is 2K. Any 48-byte region hashing to a magic value in the first 2K after a breakpoint does not constitute a new breakpoint. The maximum chunk size is 64K. If the file contents does not produce a breakpoint every 64K, LBFS will artificially insert chunk boundaries. Such artificial suppression and creation of breakpoints can disrupt the synchronization of file chunks between versions of a file. The risk, if this occurs, is that LBFS will perform no better than an ordinary file system. Fortunately, synchronization problems most often result from stylized files—for instance a long run of zeros, or a few repeated sequences none of which has a breakpoint—and such files do well under conventional compression. Since all LBFS RPC traffic gets conventionally compressed, pathological cases do not necessarily translate into slow file access.

3.1.3 Chunk Database

LBFS uses a database to identify and locate duplicate data chunks. It indexes each chunk by the first 64 bits of its SHA-1 hash. The database maps these 64-bit keys to (file, offset, count) triples. This mapping must be updated whenever a file is modified. Keeping such a database in sync with the file system could potentially incur significant overhead. Moreover, if files exported through LBFS are modified by other means—for instance by a local process on the server—LBFS cannot prevent them from growing inconsistent with the database. Even on the client side, an inopportune crash could potentially corrupt the contents of the disk cache.

To avoid synchronization problems, LBFS never relies on the correctness of the chunk database. It recomputes the SHA-1 hash of any data chunk before using it to reconstruct

a file. LBFS also uses the recomputed SHA-1 value to detect hash collisions in the database, since the 64-bit keys have a low but non-negligible probability of collision. Not relying on database integrity also frees LBFS from the need to worry about crash recovery. That in turn saves LBFS from making expensive synchronous database updates. The worst a corrupt database can do is degrade performance.

3.2 Protocol

The LBFS protocol is based on NFS version 3 [5]. NFS names all files by server-chosen opaque handles. Operations on handles include reading and writing data at specific offsets. LBFS adds extensions to exploit inter-file commonality during reads and writes. Most NFS clients poll the server on file open to check permissions and validate previously cached data. For recently accessed files, LBFS saves this round trip by adding leases to the protocol. Unlike many NFS clients, LBFS also practices aggressive pipelining of RPC calls to tolerate network latency. The system uses an asynchronous RPC library that efficiently supports large numbers of simultaneously outstanding RPCs. Finally, LBFS compresses all RPC traffic using conventional gzip compression.

3.2.1 File Consistency

The LBFS client currently performs whole file caching (though in the future we would like to cache only portions of very large files). When a user opens a file, if the file is not in the local cache or the cached version is not up to date, the client fetches a new version from the server. When a process that has written a file closes it, the client writes the data back to the server.

LBFS uses a three-tiered scheme to determine if a file is up to date. Whenever a client makes any RPC on a file in LBFS, it gets back a read lease on the file. The lease is a commitment on the part of the server to notify the client of any modifications made to that file during the term of the lease (by default one minute, though the duration is server-configurable). When a user opens a file, if the lease on the file has not expired and the version of the file in cache is up to date (meaning the server also has the same version), then the open succeeds immediately with no messages sent to the server.

If a user opens a file and the lease on the file has expired, then the client asks the server for the attributes of the file. This request implicitly grants the client a lease on the file. When the client gets the attributes, if the modification and inode change times are the same as when the file was stored in the cache, then the client uses the version in the cache with no further communication to the server. Finally, if the file times have changed, then the client must transfer the new contents from the server.

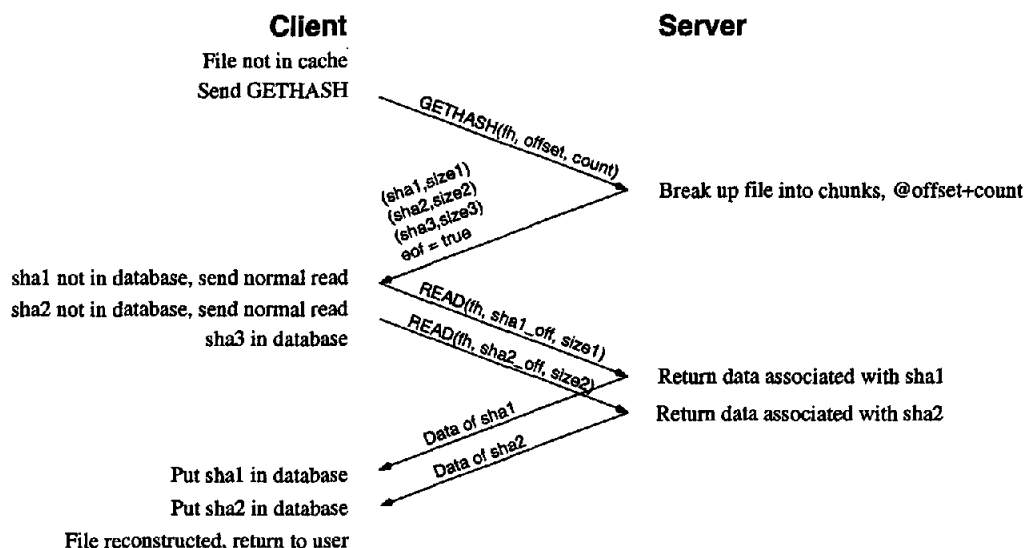


Figure 2: Reading a file using LBFS

Because LBFS only provides close-to-open consistency, a modified file does not need to be written back to the server until it is closed. Thus, LBFS does not need write leases on files—the server never demands back a dirty file. Moreover, when files are written back, they are committed atomically. Thus, if a client crashes or is cut from the network while writing a file, the file will not get corrupted or locked—other clients will simply continue to see the old version. When multiple processes on the same client have the same file open for writing, LBFS writes data back whenever any of the process closes the file. If multiple clients are writing the same file, then the last one to close the file will win and overwrite changes from the others. These semantics are similar to those of AFS.

3.2.2 File Reads

File reads in LBFS make use of one RPC procedure not in the NFS protocol, GETHASH.

GETHASH retrieves the hashes of data chunks in a file, so as to identify any chunks that already exist in the client's cache. GETHASH takes the same arguments as a READ RPC, namely a file handle, offset, and size (though in practice the size is always the maximum possible, because the client practices whole file operations). Instead of returning file data, however, GETHASH returns a vector of (SHA-1 hash, size) pairs.

Figure 2 shows the use of GETHASH. When downloading a file not in its cache, the client first calls GETHASH to obtain hashes of the file's chunks. Then, for any chunks not already in its cache, the client issues regular READ RPCs. Because the READ RPCs are pipelined, downloading a file generally only incurs two network-round trip times plus the cost of downloading any data not in the cache. For files larger

than 1,024 chunks, the client must issue multiple GETHASH calls and may incur multiple round trips. However, network latency can be overlapped with transmission and disk I/O.

3.2.3 File Writes

File writes proceed somewhat differently in LBFS from NFS. While NFS updates files at the server incrementally with each write, LBFS updates them atomically at close time. There are several reasons for using atomic updates. Most importantly, the previous version of a file often has many chunks in common with the current version. Keeping the old version around helps LBFS exploit the commonality. Second, LBFS's file reconstruction protocol can significantly alter the order of writes to a file. Files being written back may have confusing intermediary states (for instance an ASCII file might temporarily contain blocks of 0s). Finally, atomic updates limit the potential damage of simultaneous writes from different clients. Since two clients writing the same file do not see each other's updates, simultaneously changing the same file is a bad idea. When this does occur, however, atomic updates at least ensure that the resulting file contains the coherent contents written by one of the clients, rather than a mishmash of both versions.

LBFS uses temporary files to implement atomic updates. The server first creates a unique temporary file, writes the temporary file, and only then atomically commits the contents to the real file being updated. While writing the temporary file, LBFS uses chunks of existing files to save bandwidth where possible. Four RPCs implement this update protocol: MKTMPFILE, TMPWRITE, CONDWRITE, and COMMITTMP.

MKTMPFILE creates a temporary file for later use in an atomic update. MKTMPFILE takes two arguments: first,

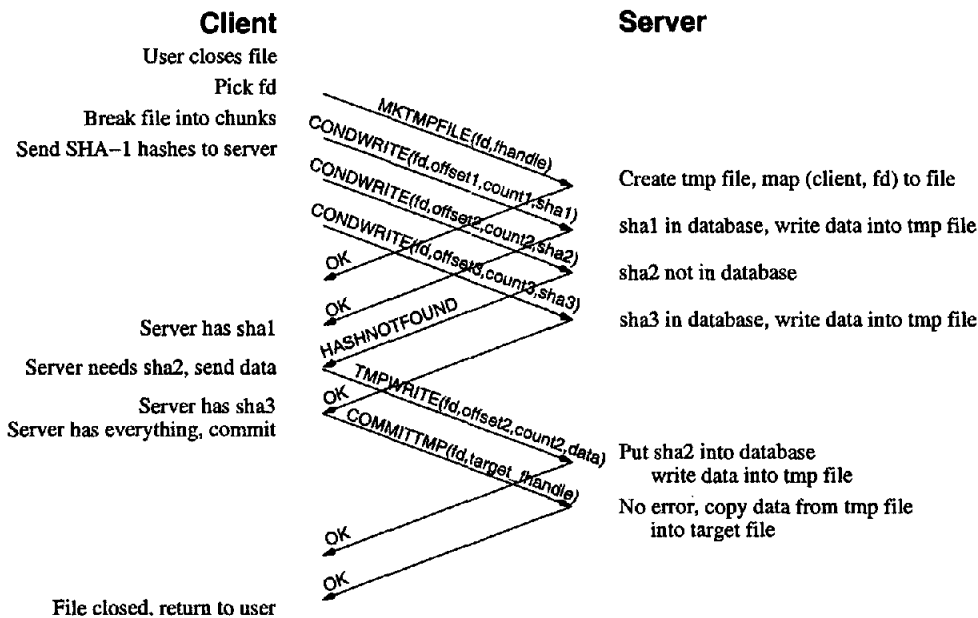


Figure 3: Writing a file using LBFS

the file handle of the file that will eventually be atomically updated, and second, a client-chosen “file descriptor” for the temporary file. After receiving the call, the server creates a temporary file in the same file system as the specified handle and keeps a mapping from the per-client file descriptor to the temporary file. Because clients choose the descriptors for temporary files, they can pipeline operations on temporary files before the MKTMPFILE RPC returns.

TMPWRITE is similar to a **WRITE** RPC. The only difference is that a client-chosen temporary file descriptor replaces the NFS file handle in the arguments. An LBFS client sends **TMPWRITE**s instead of **WRITE**s to update a file created with **MKTMPFILE**.

CONDWRITE is similar to a **TMPWRITE** RPC. The arguments contain a file descriptor, offset, and length. Instead of the actual data to write, however, **CONDWRITE** arguments contain a SHA-1 hash of the data. If the server can find the data specified by the hash somewhere in its file system, it writes the data to the temporary file at the specified offset. If it cannot find the data, but the request would otherwise have completed, **CONDWRITE** returns the special error code **HASHNOTFOUND**.

COMMITTMP commits the contents of a temporary file to a permanent file if no error has occurred. It takes two arguments, a file descriptor for the temporary file, and a file handle for the permanent file. For each temporary file descriptor, the server keeps track of any errors other than **HASHNOTFOUND** that have occurred during **CONDWRITE** and **TMPWRITE** RPCs. If any error has occurred on the file descriptor (e.g., disk full), **COMMITTMP** fails. Otherwise, the server replaces the contents of the target file with that of the

temporary file and updates the chunk database to reflect the file’s new contents. Since LBFS uses TCP, RPCs are delivered in order. Thus, the client can pipeline a **COMMITTMP** operation behind **TMPWRITE** RPCs.

Figure 3 shows the file write protocol in action. When a user closes a file that the client must write back, the client picks a file descriptor and issues a **MKTMPFILE** RPC with the handle of the closed file. In response, the server creates a temporary file handle and maps it to the specified file descriptor. The client then makes **CONDWRITE** RPCs for all data chunks in the file it is writing back. For any **CONDWRITE**s returning **HASHNOTFOUND**, the client also issues **TMPWRITE** calls. Finally, the client issues a **COMMITTMP**.

Pipelining of writes occurs in two stages. First, the client pipelines a series of **CONDWRITE** requests behind a **MKTMPFILE** RPC. Second, as the **CONDWRITE** replies come back, the client turns around and issues **TMPWRITE** RPCs for any **HASHNOTFOUND** responses. It pipelines the **COMMITTMP** immediately behind the last **TMPWRITE**. The communication overhead is therefore generally two round trip latencies, plus the transmission times of the RPCs. For large files, the client has a maximum limit on the number of outstanding **CONDWRITE** and **TMPWRITE** calls so as not to spend too much time sending calls when it can process replies. However, the extra network round trips will generally overlap with the transmission time of RPC calls.

3.2.4 Security Considerations

Because LBFS performs well over a wider range of networks than most file systems, the protocol must resist a

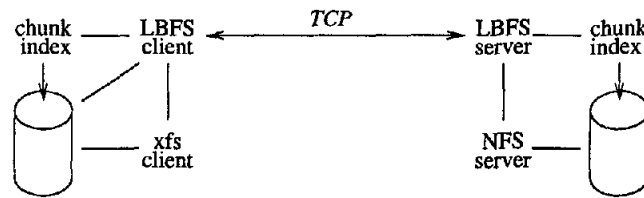


Figure 4: Overview of the LBFS implementation.

wider range of attacks. LBFS uses the security infrastructure from SFS [16]. Every server has a public key, which the client administrator specifies on the command line when mounting the server. In the future, we intend to embed public keys in pathnames as SFS does and to integrate LBFS into SFS's auto-mounting system so that unprivileged users on clients can access any server. The entire LBFS protocol, RPC headers and all, is passed through gzip compression, tagged with a message authentication code, and then encrypted. At mount time, the client and server negotiate a session key, the server authenticates itself to the user, and the user authenticates herself to the client, all using public key cryptography.

Finally, we note that LBFS may raise some non-network security issues. When several users share the same file system, LBFS could leak information about files a user is not allowed to read. Specifically, through careful use of CONDWRITE, a user can check whether the file system contains a particular chunk of data, even if the data resides in a read-protected file. Though CONDWRITE will fail on chunks the user cannot read, subtle timing differences may still let the user infer that the database contained the hash of the chunk. Nonetheless, LBFS should provide more than adequate security for most purposes, particularly given how widely users accept file systems that do not even encrypt network traffic.

4 Implementation

Figure 4 shows the architecture of the LBFS implementation. Both the client and server run at user-level. The client implements the file system using *xfs*, a device driver bundled with the ARLA [24] file system. The server accesses files through NFS. The client and server communicate over TCP, using Sun RPC. We used the asynchronous RPC library from the SFS toolkit [15] both for the server's NFS client and for LBFS client-server communication. The RPC library already had support for authenticating and encrypting traffic between a client and server. We added support for compression.

4.1 Chunk Index

The LBFS client and server both maintain chunk indexes, the server indexing file system contents and the client its local cache. The two share the same indexing code, imple-

mented using the B-tree from SleepyCat software's BerkeleyDB package. Since LBFS never relies on chunk database correctness, it also does not concern itself with crash recoverability. LBFS avoids any synchronous database updates, and the server always replies to clients before inserting new chunks in its database. If the database loses a few hashes, clients will simply use more bandwidth until the database comes back up to date. There is a utility, *mkdb*, which builds a file system's database from scratch. However, if an LBFS server is run without a database, the server simply creates the database and populates it as users access files.

The one database operation on the critical path for clients is the lookup done as part of a CONDWRITE RPC. However, for all but the smallest files, CONDWrites are pipelined deeply enough to overlap database lookups with the transmission of any write data not found in the chunk index. For 8 KByte or smaller files, LBFS avoids CONDWrites and simply writes the files directly to the server in a single RPC. The overhead of multiple round trip times overshadows any potential bandwidth savings on such small files.

4.2 Server Implementation

Our main goal for the LBFS server implementation, other than saving bandwidth and providing acceptable performance, was to build a system that could unobtrusively be installed on an already running file system. This both isolates LBFS's benefits from physical file system layout and lets users take immediate advantage of LBFS on existing files without dedicating a disk or partition to it.

The LBFS server accesses the file system by pretending to be an NFS client, effectively translating LBFS requests into NFS. Building the LBFS server as an NFS client lets LBFS serve any file system for which an NFS server exists, which includes most file systems on most Unix operating systems.

Of course, the server might alternatively have been implemented using regular system calls to access the file system. However, NFS offers several advantages over the traditional system call interface. First, it simplifies the implementation, since the LBFS protocol is based on NFS. Second, NFS saved the LBFS server from the need to implement access control. The server simply maps LBFS requests to user IDs and tags the resulting NFS requests with those IDs, letting the NFS server decide whether or not to grant access. Fi-

nally, NFS allows the chunk index to be more resilient to outside file system changes. When a file is renamed, its NFS file handle remains the same and thus the chunk index does not need to be updated.

The LBFS server creates a “trash directory,” `.lbfs.trash`, in the root directory of every file system it exports. The trash directory contains temporary files created by MK-TMPFILE RPCs. As explained below, after a COMMIT-TMP RPC, the LBFS server does not delete the committed temporary file. Rather, if space is needed, it garbage-collects a random file in the trash directory. A background thread purges the database of pointers to deleted files.

4.2.1 Static i-number Problem

The one major disadvantage to using NFS is the lack of low-level control over file system data structures. In particular, Unix file system semantics dictate that a file’s i-number not change when the file is overwritten. Thus, when the server commits a temporary file to a target file, it has to copy the contents of the temporary file onto the target file rather than simply rename the temporary file into place, so as to preserve the target file’s i-number. Not only is this gratuitously inefficient, but during the copy other clients cannot access the target file. Worse yet, a server crash will leave the file in an inconsistent state (though the client will restart the file transfer after the COMMIT fails).

A related problem occurs with file truncation. Applications often truncate files and then immediately overwrite them with similar versions. Thus, LBFS would benefit from having the previous contents of a truncated file when reconstructing the new contents. The obvious solution is to move truncated files into the trash directory and replace them with new, zero-length files. Unfortunately, the NFS interface will not let the server do this without changing the truncated file’s i-number. To avoid losing the contents of truncated files, then, LBFS delays the deletion of temporary files after COMMIT-TMP RPCs. Thus, many truncated files will still have copies in the trash directory, and new versions can be reconstituted from those copies.

It is worth noting that the static i-number problem could be solved given a file system operation that truncates a file *A* to zero length and then atomically replaces the contents of a second file *B* with the previous contents of *A*. We can even afford to lose the original contents of *A* after an inopportune crash. In the case of COMMIT-TMP, the lost data will be resent by the client. Such a “truncate and update” operation would be efficient and easy to implement for most Unix physical file system layouts. It might in other situations serve as a more efficient alternative to the rename operation for atomically updating files. Unfortunately, the current LBFS server must make do without such an operation.

4.3 Client Implementation

The LBFS client uses the `xfs` device driver. `xfs` lets user-level programs implement a file system by passing messages to the kernel through a device node in `/dev`. We chose `xfs` for its suitability to whole-file caching. The driver notifies the LBFS client whenever it needs the contents of a file a user has opened, or whenever a file is closed and must be written back to the server. The LBFS client is responsible for fetching remote files and storing them in the local cache. It informs `xfs` of the bindings between files users have opened and files in the local cache. `xfs` then satisfies read and write requests directly from the cache, without the need to call into user-level code each time.

5 Evaluation

This section evaluates LBFS using several experiments. First, we examine the behavior of LBFS’s content-based breakpoint chunking on static file sets. Next, we measure the bandwidth consumption and network utilization of LBFS under several common workloads and compare it to that of CIFS, NFS version 3 and AFS. Finally, we show that LBFS can improve end-to-end application performance when compared with AFS, CIFS, and NFS.

Our experiments were conducted on identical 1.4 GHz Athlon computers, each with 256 MBytes of RAM and a 7,200 RPM, 8.9 ms Seagate ST320414A IDE drive. The IDE drives are slower than common SCSI drives, which penalizes LBFS for performing more disk operations than other file systems. Except where otherwise noted, all file system clients ran on OpenBSD 2.9 and servers on FreeBSD 4.3. The AFS client was the version of ARLA bundled with BSD, configured with a 512 MByte cache. The AFS server was `openafs 1.1.1` running on Linux 2.4.3. For the Microsoft Word experiments, we ran Office 2000 on a 900 MHz IBM ThinkPad T22 laptop with 256 MBytes of RAM, Windows 98, and `openafs 1.1.1` with a 400 MByte cache.

The clients and servers in our experiments were connected by full-duplex 100 Mbit Ethernet through the Click [11] modular router, which can be configured to measure traffic and impose bandwidth limitations, delay, and loss. Click ran on a Linux 2.2.18 Athlon machine.

5.1 Repeated Data in Files

LBFS’s content-based breakpoint chunking scheme reduces bandwidth only if different files or versions of the same file share common data. Fortunately, this occurs relatively frequently in practice. Table 1 summarizes the amount of commonality found between various files.

We examined `emacs` to see how much commonality there is between files under a software development workload. The `emacs 20.7` source tree is 52.1 MBytes. However, if a client already has source for `emacs 20.6` in its cache, it

Data	Given	Data size	New data	Overlap
emacs 20.7 source	emacs 20.6	52.1 MB	12.6 MB	76%
Build tree of emacs 20.7	—	20.2 MB	12.5 MB	38%
emacs 20.7 + printf executable	emacs 20.7	6.4 MB	2.9 MB	55%
emacs 20.7 executable	emacs 20.6	6.4 MB	5.1 MB	21%
Installation of emacs 20.7	emacs 20.6	43.8 MB	16.9 MB	61%
Elisp doc. + new page	original postscript	4.1 MB	0.4 MB	90%
MSWord doc. + edits	original MSWord	1.4 MB	0.4 MB	68%

Table 1: Amount of new data in a file or directory, given an older version.

only needs to download 12.6 MBytes to reconstitute the 20.7 source tree—a 76% savings. An emacs 20.7 build tree consumes 20.2 MBytes of disk space, but only contains 12.5 MBytes of unique chunks. Thus, writing a build tree, LBFS will save 38% even if the server starts with an empty chunk database. When adding a debugging printf to emacs 20.7 and recompiling, changing the size of the executable, the new binary has 55% in common with the old one. Between emacs-20.6 and 20.7, the two executables have 21% commonality. A full emacs 20.7 installation consumes 43.8 MBytes. However, 61% of this would not need to be transferred to a client that already had emacs 20.6 in its cache.

We also examined two document preparation workloads. When adding a page to the front of the emacs lisp manual, the new postscript file had 90% in common with the previous one. Unfortunately, if we added two pages, it changed the page numbering for more than the first chapter, and the commonality disappeared. From this we conclude that LBFS is suitable for postscript document previewing—for instance tuning a T_EX document to get an equation to look right—but that between substantial revisions of a document there will be little commonality (unless the pages are numbered by chapter). We also used Microsoft Word to sprinkle references to LBFS in a paper about Windows 2000 disk performance, and found that the new version had 68% overlap with the original.

To investigate the behavior of LBFS's chunking algorithm, we ran *mkdb* on the server's */usr/local* directory, using an 8 KByte chunk size and 48-byte moving window. */usr/local* contained 354 MBytes of data in 10,702 files. *mkdb* broke the files into 42,466 chunks. 6% of the chunks appeared in 2 or more files. The generated database consumed 4.7 MBytes of space, or 1.3% the size of the directory. It took 9 minutes to generate the database. Figure 5 shows the distribution of chunk sizes. The median is 5.8K, and the mean 8,570 bytes, close to the expected value of 8,240 bytes. 11,379 breakpoints were suppressed by the 2K minimum chunk size requirement, while 75 breakpoints were inserted because of the 64K maximum chunk size limit. Note that the database does contain chunks shorter than 2K. These chunks come from files that are shorter than 2K and from the ends of larger files (since an end of file is always a chunk

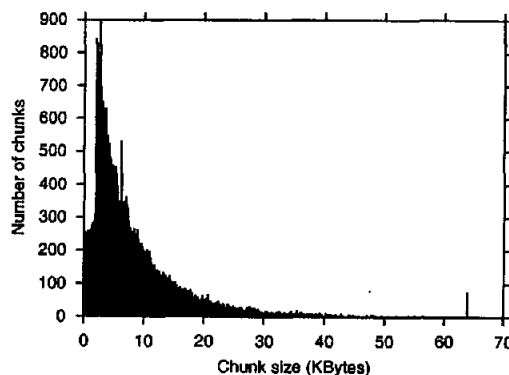


Figure 5: Distribution of chunk sizes in the */usr/local* database. X-axis represents chunk sizes, in KBytes. Y-axis shows the number of chunks having that size.

Exp chunk size	% of data in shared chunks	
	24 B window	48 B window
2 KB	21.33%	21.30%
4 KB	19.29%	19.65%
8 KB	17.01%	18.01%

Table 2: Percentage of bytes in shared chunks in */usr/local* for various chunk and window sizes. Minimum chunk size was always 1/4 the expected chunk size.

boundary).

Table 2 shows the amount of data in */usr/local* that appears in shared chunks for various expected chunk sizes and breakpoint window sizes. As expected, smaller chunks yield somewhat greater commonality, as smaller common segments between files can be isolated. However, the increased cost of GETHASH and CONWRITE traffic associated with smaller chunks outweighed the increased bandwidth savings in tests we performed. Window size does not appear to have a large effect on commonality.

5.2 Practical Workloads

We use three workloads to evaluate LBFS's ability to reduce bandwidth. In the first workload, MSWord, we open a 1.4 MByte Microsoft Word document, make the same ed-

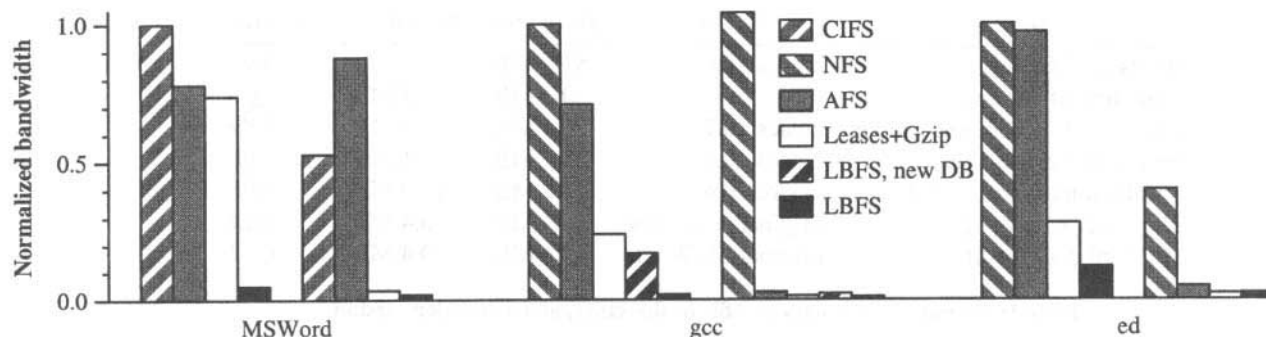


Figure 6: Normalized bandwidth consumed by three workloads. The first four bars of each workload show upstream bandwidth, the second four downstream bandwidth. The results are normalized against the upstream bandwidth of CIFS or NFS.

its as for Table 1, then measure the cost to save and close the file.¹ For the second workload, gcc, we simply recompile emacs 20.7 from source. The third workload, ed, involves making a series of changes to the perl 5.6.0 source tree to transform it into perl 5.6.1. Using the expect language, we scripted the ed text editor based on the output of diff. The benchmark saves files after text insertion operations if there have been 40 or more lines of editing. (For reference, emacs creates auto-save files every 300 characters typed.) These workloads reflect the common activities of document editing and software development.

We ran each benchmark over the client's native network file system—CIFS for Windows and NFS UDP for Unix.² We also ran all three benchmarks over AFS. To isolate the benefits of exploiting file commonality, we additionally measured a “Leases+Gzip” file system that uses LBFS's file caching, leases, and data compression, but not its chunking scheme. Finally, we ran the three workloads over LBFS.

For the MSWord benchmark, because LBFS only runs on Unix, we ran a Samba server on the OpenBSD LBFS client, re-exporting the LBFS file system with CIFS. We saved the Word document from the Windows 98 client to the Samba server over 100 Mbit Ethernet, and measured the traffic going through the Click router between the Samba server and the LBFS server. All MSWord experiments were conducted with a warm cache; the original files had been written on the same client through the file system under test. However, the LBFS server did not have output files from previous runs of the benchmark in its database.

For the gcc benchmark, the emacs sources had been unpacked through the file system under test. Emacs had also previously been compiled on the same file system. The intent was to simulate what happens when one modifies a header file that requires an entire project to be recompiled. Though two successive compilations of emacs do not produce the same executable, there was substantial commonality

between object files created by the benchmark and ones in the server's trash directory. To isolate this benefit, we also measured a compilation of emacs when the LBFS server was started with a new database not containing chunks from any previous compiles.

Because of a bug in expect on OpenBSD, we used a FreeBSD client for all instances of the ed benchmark. Like the MSWord benchmark, we ran ed with a warm client cache to a server that had not previously seen the output files. We also ran ed over an ssh remote login connection, to compare using a distributed file system to running a text editor remotely. To simulate some type-ahead, the benchmark sends one line at a time and waits for the line to echo.

5.3 Bandwidth Utilization

Figure 6 shows the bandwidth consumed by the client writing to and reading from the server under each of the three workloads. The bandwidth numbers are obtained from byte counters in the Click router. For this experiment, the router did not impose any delay, loss, or bandwidth limitations. (tcp reported TCP throughput of 89 Mbit/sec between the client and server, and ping reported a round-trip time of 0.2 ms.) In each case, we separately report first the upstream traffic from client to server, then the downstream traffic from server to client. The numbers are normalized to the upstream (client to server) bandwidth of the native file system, CIFS on Windows and NFS on Unix.

Because AFS, Leases+Gzip, and LBFS all have large, on-disk caches, all three systems reduce the amount of downstream bandwidth from server to client when compared to the native file systems. For upstream bandwidth, the drops from CIFS and NFS bandwidth to AFS bandwidth represent savings gained from deferring writes to close time and eliding overwrites of the same data. The drops from AFS bandwidth to Leases+Gzip bandwidth represent savings from compression. Finally, the drops from Leases+Gzip bandwidth to LBFS bandwidth represent savings gained from the chunking scheme.

For the MSWord workload, the savings provided by the

¹We did not enable Word's “fast saves” feature, as it neither reduced write bandwidth nor improved running time.

²NFS TCP performs worse than NFS UDP, probably because it has not been as extensively tested and tuned.

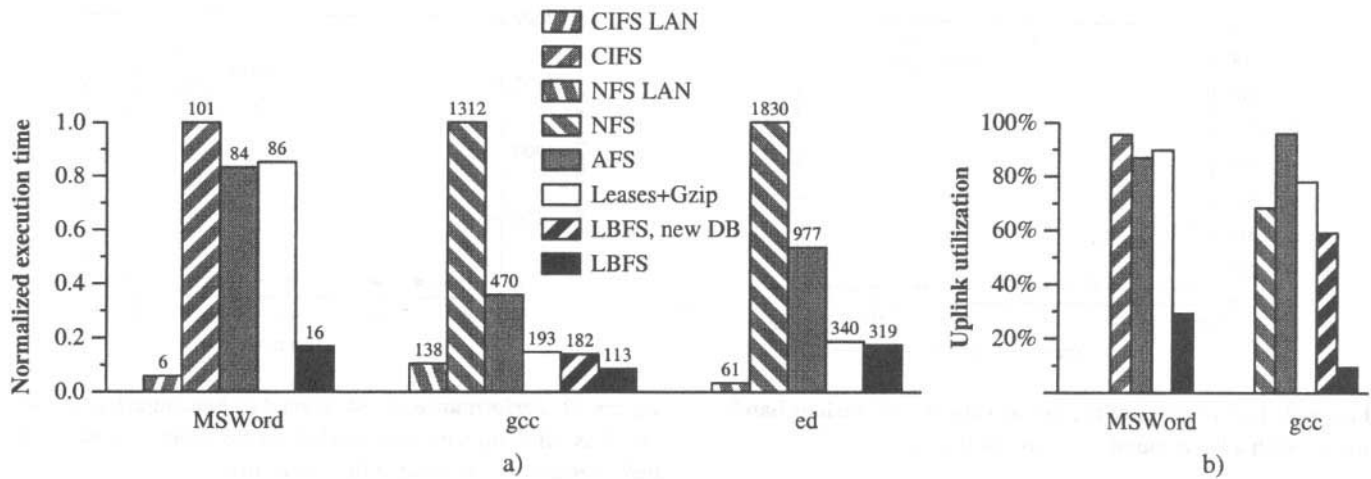


Figure 7: a) Normalized application performance on top of several file systems over a cable modem link with 384 Kbit/sec uplink and 1.5 Mbit/sec downlink. Execution times are normalized against CIFS or NFS results. Execution times in seconds appear on top of the bars. b) Uplink bandwidth utilization of the MSWord and gcc benchmarks.

chunking scheme come not only from commonality between the old and new versions of the document, but also from commonality with large temporary files that Word creates during saves. LBFS is able to reduce the upstream bandwidth by 15 times over Leases+Gzip, 16 times over AFS, and 20 times over CIFS. More careful analysis reveals that the Unix Samba server closes then reopens temporary files, requiring them to be transferred multiple times. These multiple transfers largely negate the benefits of gzip compression in Leases+Gzip. In contrast, LBFS exploits the files' common contents from one close to the next, consuming very little unnecessary traffic. AFS uses only slightly more bore bandwidth than Leases+Gzip, either because the extra closes are an artifact of the Unix Samba server, or perhaps because the Windows AFS implementation performs partial file caching.

For the gcc benchmark, the savings provided by the chunking scheme come from the fact that many of the compiled object files, libraries, and executables are similar or identical to files in the server's trash directory. Chunks only need to be written to the server where object files differ or files have been evicted from the trash directory. In this case, LBFS was able to reduce the upstream bandwidth by 15 times over Leases+Gzip, 46 times over AFS, and more than 64 times over NFS. Even without the benefit of old object files in the database, LBFS still reduces upstream bandwidth utilization because many object files, libraries, and executables share common data. When started with a new and empty chunk database, LBFS still used 30% less upstream bandwidth than Leases+Gzip.

In the ed case, the savings provided by the chunking scheme come from writing versions of files that share common chunks with older revisions. LBFS was able to reduce the upstream bandwidth by more than a factor of 2 over Leases+Gzip and 8 over AFS and NFS.

5.4 Application Performance

Figure 7a shows the normalized end-to-end application performance of the three workloads on a simulated cable modem link, with 1.5 Mbit/sec downstream bandwidth from server to client, 384 Kbit/sec upstream bandwidth from client to server, and 30 ms of round-trip latency. The execution times are normalized against CIFS or NFS results. For comparison, we also show the execution times of the native file system on a 100 Mbit/sec full-duplex LAN.

For the MSWord workload, LBFS was able to reduce the execution times from a potentially unusable 101 seconds with CIFS to a much more tolerable 16 seconds, more than 6 times faster. In fact, AFS takes 16 seconds to run the benchmark on a LAN, though CIFS takes only 6 seconds. The gcc workload took 113 seconds under LBFS with a populated database, 1.7 times faster than Leases+Gzip, 4 times faster than AFS, almost 12 times faster than NFS, and 18% faster than NFS on a LAN. With a new server database, LBFS still reduces the execution time by 6% over Leases+Gzip, though it is 32% slower than NFS on a LAN.

For both the MSWord and gcc workloads, Figure 7b shows that LBFS reduces network utilization, or the percentage of available bandwidth used by the file system. Over LBFS, gcc used only used only 9.5% of the 384 Kbit per second upstream link. In contrast, gcc under NFS used 68% and under AFS used 96%. For the MSWord benchmarks, LBFS was able to reduce the upstream network utilization from 87% and 96% with AFS and CIFS to 29%.

Figure 8 examines the effects of available network bandwidth on the performance of the gcc workload over LBFS, Leases+Gzip, and AFS. In these experiments, the simulated network has a fixed round trip time of 10 ms. This graph shows that LBFS is least affected by a reduction in available network bandwidth, because LBFS reduces the read and

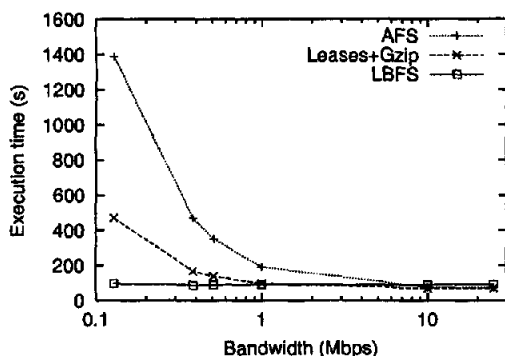


Figure 8: Performance of the gcc workload over various bandwidths with a fixed round-trip time of 10 ms.

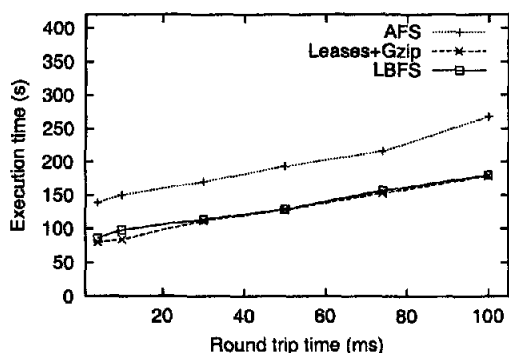


Figure 9: Performance of the gcc workload over a range of round-trip times with fixed 1.5 Mbit/sec symmetric links.

write bandwidth required by the workload to the point where CPU and network latency, not bandwidth, become the limiting factors.

Figure 9 examines the effects of network latency on LBFS, Leases+Gzip, and AFS performance. In these experiments, the simulated network has symmetric 1.5 Mbit per second links. Although the gcc workload uses more bandwidth over Leases+Gzip, the performance of the workload over LBFS and Leases+Gzip are roughly the same because the available network bandwidth is high enough. On the other hand, because gcc over AFS uses significantly more bandwidth, it performs worse than both LBFS and Leases+Gzip. This graph shows that the execution time of the gcc workload degrades similarly on all three file systems as latency increases.

Figure 7a also shows LBFS's performance on the ed benchmark, a 6% improvement over Leases+Gzip, 67% over AFS, and 83% over NFS. However, execution time is not the best measure of performance for interactive workloads. Users care about delays of over a second, but cannot differentiate much smaller ones that nonetheless affect the runtime of a scripted benchmark. Long delays are most often caused by TCP entering the backoff state. We therefore ran a shortened version of the ed benchmark over a network with

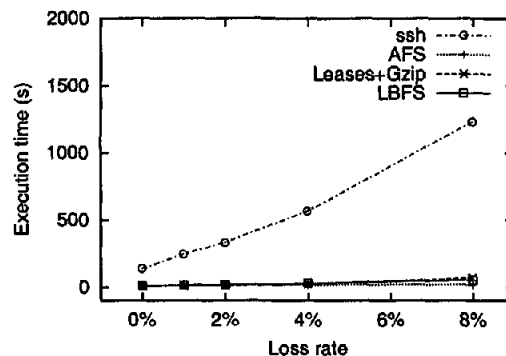


Figure 10: Performance of a shortened ed benchmark over various loss rates, on a network with fixed 1.5 Mbit/sec symmetric links and a fixed round-trip time of 10 ms.

simulated packet loss, comparing the performance of the network file systems with the ssh remote login program.

Figure 10 compares file systems to ssh under various loss rates. With no packet loss, ssh is slower than any file system, but the difference would not affect performance at the rate users type. However, as the loss rate increases, delays are imposed by TCP's backoff mechanism. As ssh never has more than a few packets in flight, every lost packet puts TCP into backoff, imposing a delay of one or more seconds while the user waits for typed characters to echo. The file systems outperform ssh for several reasons. First, LBFS and Leases+Gzip experience fewer losses by sending fewer total packets than ssh; the file systems both consume less bandwidth and send more data per packet. Second, when file systems transfer large files, TCP can get four or more packets in flight, allowing it to recover from a single loss with fast retransmission and avoid backoff. AFS uses UDP rather than TCP, and does not appear to reduce its sending rate as precipitously as TCP in the face of packet loss. We conclude that for the kind of editing in this benchmark, it is far preferable to use a network file system then to run an editor remotely.

6 Summary

LBFS is a network file system that saves bandwidth by taking advantage of commonality between files. LBFS breaks files into chunks based on contents, using the value of a hash function on small regions of the file to determine chunk boundaries. It indexes file chunks by their hash values, and subsequently looks up chunks to reconstruct files that contain the same data without sending that data over the network.

Under common operations such as editing documents and compiling software, LBFS can consume over an order of magnitude less bandwidth than traditional file systems. Such a dramatic savings in bandwidth makes LBFS practical for situations where other file systems cannot be used. In many situations, LBFS makes transparent remote file access a vi-

able and less frustrating alternative to running interactive programs on remote machines.

7 Acknowledgments

We thank Chuck Blake, Frans Kaashoek, Butler Lampson, Robert Morris, Marc Waldman, and the anonymous reviewers for their feedback and suggestions. We thank Chuck Blake and Frank Dabek for help with hardware, Nickolai Zeldovich for setting up AFS, and Chuck Blake, Frank Dabek, and Bryan Ford for assistance with xfs.

References

- [1] Brian Berliner. CVS II: Parellizing software development. In *Proceedings of the Winter 1990 USENIX Technical Conference*, Colorado Springs, CO, 1990.
- [2] David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Westley Weimer, Christopher Wells, Ben Zhao, and John Kubiawicz. Oceanstore: An extremely wide-area storage system. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–201, Boston, MA, November 2000.
- [3] Sergey Brin, James Davis, and Hector Garcia-Molina. Copy detection mechanisms for digital documents. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 398–409, San Jose, CA, May 1995.
- [4] Andrei Broder. On the resemblance and containment of documents. *Compression and Complexity of Sequences*, pages 21–29, 1997.
- [5] B. Callaghan, B. Pawlowski, and P. Staubach. NFS version 3 protocol specification. RFC 1813, Network Working Group, June 1995.
- [6] FIPS 180-1. *Secure Hash Standard*. U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, VA, April 1995.
- [7] Cary G. Gray and David R. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, pages 202–210, Litchfield Park, AZ, December 1989.
- [8] Björn Grönvall, Assar Westerlund, and Stephen Pink. The design of a multicast-based distributed file system. In *Proceedings of the Third Symposium on Operating System Design and Implementation*, pages 251–264, New Orleans, LA, February 1999.
- [9] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.
- [10] James J. Kistler and M. Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.
- [11] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(4):263–297, November 2000.
- [12] Yui-Wah Lee, Kwong-Sak Leung, and M. Satyanarayanan. Operation-based update propagation in a mobile file system. In *Proceedings of the 1999 USENIX Technical Conference*, Monterey, CA, June 1999.
- [13] Udi Manber. Finding similar files in a large file system. In *Proceedings of the Winter 1994 USENIX Technical Conference*, San Francisco, CA, January 1994.
- [14] Timothy Mann, Andrew D. Birrell, Andy Hisgen, Chuck Jerian, and Garret Swart. A coherent distributed file cache with directory write-behind. *ACM Transactions on Computer Systems*, 12(2):123–164, May 1994.
- [15] David Mazières. A toolkit for user-level file systems. In *Proceedings of the 2001 USENIX Technical Conference*, Boston, MA, June 2001.
- [16] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 124–139, Kiawa Island, SC, 1999.
- [17] Jeffrey C. Mogul, Fred Douglass, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *Proceedings of the 1997 ACM SIGCOMM Conference*, pages 181–194, Cannes, France, September 1997.
- [18] Karin Petersen, Mike J. Spreitzer, and Douglas B. Terry. Flexible update propagation for weakly consistent replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 288–301, Saint-Malo, France, 1997.
- [19] Michael O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.
- [20] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. NFS version 4 protocol. RFC 3010, Network Working Group, December 2000.
- [21] Neil T. Spring and David Wetherall. A protocol independent technique for eliminating redundant network traffic. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 87–95, Stockholm, Sweden, August 2000.
- [22] Walter F. Tichy. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems*, 2(4):309–321, November 1984.
- [23] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, April 2000.
- [24] Assar Westerlund and Johan Danielsson. Arla—a free AFS client. In *Proceedings of the 1998 USENIX, Freenix track*, New Orleans, LA, June 1998. USENIX.
- [25] Haifeng Yu and Amin Vahdat. Design and evaluation of a continuous consistency model for replicated services. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, San Diego, CA, 2000.

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

Categories and Subject Descriptors

D [4]: 3—*Distributed file systems*

General Terms

Design, reliability, performance, measurement

Keywords

Fault tolerance, scalability, data storage, clustered storage

*The authors can be reached at the following addresses: {sanjay,hgobioff,shuntak}@google.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA.
Copyright 2003 ACM 1-58113-757-5/03/0010 ...\$5.00.

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Second, files are huge by traditional standards. Multi-GB files are common. Each file typically contains many application objects such as web documents. When we are regularly working with fast growing data sets of many TBs comprising billions of objects, it is unwieldy to manage billions of approximately KB-sized files even when the file system could support it. As a result, design assumptions and parameters such as I/O operation and block sizes have to be revisited.

Third, most files are mutated by appending new data rather than overwriting existing data. Random writes within a file are practically non-existent. Once written, the files are only read, and often only sequentially. A variety of data share these characteristics. Some may constitute large repositories that data analysis programs scan through. Some may be data streams continuously generated by running applications. Some may be archival data. Some may be intermediate results produced on one machine and processed on another, whether simultaneously or later in time. Given this access pattern on huge files, appending becomes the focus of performance optimization and atomicity guarantees, while caching data blocks in the client loses its appeal.

Fourth, co-designing the applications and the file system API benefits the overall system by increasing our flexibility.

For example, we have relaxed GFS's consistency model to vastly simplify the file system without imposing an onerous burden on the applications. We have also introduced an atomic append operation so that multiple clients can append concurrently to a file without extra synchronization between them. These will be discussed in more details later in the paper.

Multiple GFS clusters are currently deployed for different purposes. The largest ones have over 1000 storage nodes, over 300 TB of disk storage, and are heavily accessed by hundreds of clients on distinct machines on a continuous basis.

2. DESIGN OVERVIEW

2.1 Assumptions

In designing a file system for our needs, we have been guided by assumptions that offer both challenges and opportunities. We alluded to some key observations earlier and now lay out our assumptions in more details.

- The system is built from many inexpensive commodity components that often fail. It must constantly monitor itself and detect, tolerate, and recover promptly from component failures on a routine basis.
- The system stores a modest number of large files. We expect a few million files, each typically 100 MB or larger in size. Multi-GB files are the common case and should be managed efficiently. Small files must be supported, but we need not optimize for them.
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads. In large streaming reads, individual operations typically read hundreds of KBs, more commonly 1 MB or more. Successive operations from the same client often read through a contiguous region of a file. A small random read typically reads a few KBs at some arbitrary offset. Performance-conscious applications often batch and sort their small reads to advance steadily through the file rather than go back and forth.
- The workloads also have many large, sequential writes that append data to files. Typical operation sizes are similar to those for reads. Once written, files are seldom modified again. Small writes at arbitrary positions in a file are supported but do not have to be efficient.
- The system must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file. Our files are often used as producer-consumer queues or for many-way merging. Hundreds of producers, running one per machine, will concurrently append to a file. Atomicity with minimal synchronization overhead is essential. The file may be read later, or a consumer may be reading through the file simultaneously.
- High sustained bandwidth is more important than low latency. Most of our target applications place a premium on processing data in bulk at a high rate, while few have stringent response time requirements for an individual read or write.

2.2 Interface

GFS provides a familiar file system interface, though it does not implement a standard API such as POSIX. Files are organized hierarchically in directories and identified by pathnames. We support the usual operations to *create*, *delete*, *open*, *close*, *read*, and *write* files.

Moreover, GFS has *snapshot* and *record append* operations. Snapshot creates a copy of a file or a directory tree at low cost. Record append allows multiple clients to append data to the same file concurrently while guaranteeing the atomicity of each individual client's append. It is useful for implementing multi-way merge results and producer-consumer queues that many clients can simultaneously append to without additional locking. We have found these types of files to be invaluable in building large distributed applications. Snapshot and record append are discussed further in Sections 3.4 and 3.3 respectively.

2.3 Architecture

A GFS cluster consists of a single *master* and multiple *chunkservers* and is accessed by multiple *clients*, as shown in Figure 1. Each of these is typically a commodity Linux machine running a user-level server process. It is easy to run both a chunkserver and a client on the same machine, as long as machine resources permit and the lower reliability caused by running possibly flaky application code is acceptable.

Files are divided into fixed-size *chunks*. Each chunk is identified by an immutable and globally unique 64 bit *chunk handle* assigned by the master at the time of chunk creation. Chunkservers store chunks on local disks as Linux files and read or write chunk data specified by a chunk handle and byte range. For reliability, each chunk is replicated on multiple chunkservers. By default, we store three replicas, though users can designate different replication levels for different regions of the file namespace.

The master maintains all file system metadata. This includes the namespace, access control information, the mapping from files to chunks, and the current locations of chunks. It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunkservers. The master periodically communicates with each chunkserver in *HeartBeat* messages to give it instructions and collect its state.

GFS client code linked into each application implements the file system API and communicates with the master and chunkservers to read or write data on behalf of the application. Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunkservers. We do not provide the POSIX API and therefore need not hook into the Linux vnode layer.

Neither the client nor the chunkserver caches file data. Client caches offer little benefit because most applications stream through huge files or have working sets too large to be cached. Not having them simplifies the client and the overall system by eliminating cache coherence issues. (Clients do cache metadata, however.) Chunkservers need not cache file data because chunks are stored as local files and so Linux's buffer cache already keeps frequently accessed data in memory.

2.4 Single Master

Having a single master vastly simplifies our design and enables the master to make sophisticated chunk placement

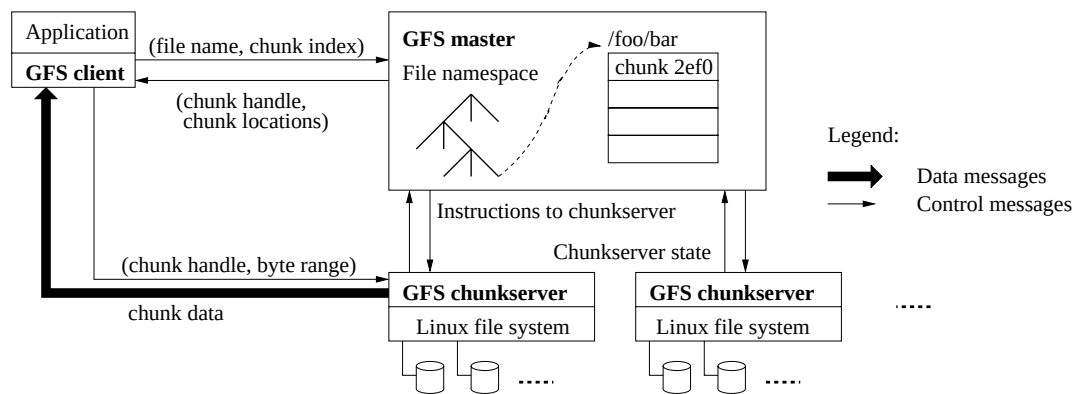


Figure 1: GFS Architecture

and replication decisions using global knowledge. However, we must minimize its involvement in reads and writes so that it does not become a bottleneck. Clients never read and write file data through the master. Instead, a client asks the master which chunkserver it should contact. It caches this information for a limited time and interacts with the chunkservers directly for many subsequent operations.

Let us explain the interactions for a simple read with reference to Figure 1. First, using the fixed chunk size, the client translates the file name and byte offset specified by the application into a chunk index within the file. Then, it sends the master a request containing the file name and chunk index. The master replies with the corresponding chunk handle and locations of the replicas. The client caches this information using the file name and chunk index as the key.

The client then sends a request to one of the replicas, most likely the closest one. The request specifies the chunk handle and a byte range within that chunk. Further reads of the same chunk require no more client-master interaction until the cached information expires or the file is reopened. In fact, the client typically asks for multiple chunks in the same request and the master can also include the information for chunks immediately following those requested. This extra information sidesteps several future client-master interactions at practically no extra cost.

2.5 Chunk Size

Chunk size is one of the key design parameters. We have chosen 64 MB, which is much larger than typical file system block sizes. Each chunk replica is stored as a plain Linux file on a chunkserver and is extended only as needed. Lazy space allocation avoids wasting space due to internal fragmentation, perhaps the greatest objection against such a large chunk size.

A large chunk size offers several important advantages. First, it reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information. The reduction is especially significant for our workloads because applications mostly read and write large files sequentially. Even for small random reads, the client can comfortably cache all the chunk location information for a multi-TB working set. Second, since on a large chunk, a client is more likely to perform many operations on a given chunk, it can reduce network overhead by keeping a persis-

tent TCP connection to the chunkserver over an extended period of time. Third, it reduces the size of the metadata stored on the master. This allows us to keep the metadata in memory, which in turn brings other advantages that we will discuss in Section 2.6.1.

On the other hand, a large chunk size, even with lazy space allocation, has its disadvantages. A small file consists of a small number of chunks, perhaps just one. The chunkservers storing those chunks may become hot spots if many clients are accessing the same file. In practice, hot spots have not been a major issue because our applications mostly read large multi-chunk files sequentially.

However, hot spots did develop when GFS was first used by a batch-queue system: an executable was written to GFS as a single-chunk file and then started on hundreds of machines at the same time. The few chunkservers storing this executable were overloaded by hundreds of simultaneous requests. We fixed this problem by storing such executables with a higher replication factor and by making the batch-queue system stagger application start times. A potential long-term solution is to allow clients to read data from other clients in such situations.

2.6 Metadata

The master stores three major types of metadata: the file and chunk namespaces, the mapping from files to chunks, and the locations of each chunk's replicas. All metadata is kept in the master's memory. The first two types (namespaces and file-to-chunk mapping) are also kept persistent by logging mutations to an *operation log* stored on the master's local disk and replicated on remote machines. Using a log allows us to update the master state simply, reliably, and without risking inconsistencies in the event of a master crash. The master does not store chunk location information persistently. Instead, it asks each chunkserver about its chunks at master startup and whenever a chunkserver joins the cluster.

2.6.1 In-Memory Data Structures

Since metadata is stored in memory, master operations are fast. Furthermore, it is easy and efficient for the master to periodically scan through its entire state in the background. This periodic scanning is used to implement chunk garbage collection, re-replication in the presence of chunkserver failures, and chunk migration to balance load and disk space

usage across chunkservers. Sections 4.3 and 4.4 will discuss these activities further.

One potential concern for this memory-only approach is that the number of chunks and hence the capacity of the whole system is limited by how much memory the master has. This is not a serious limitation in practice. The master maintains less than 64 bytes of metadata for each 64 MB chunk. Most chunks are full because most files contain many chunks, only the last of which may be partially filled. Similarly, the file namespace data typically requires less than 64 bytes per file because it stores file names compactly using prefix compression.

If necessary to support even larger file systems, the cost of adding extra memory to the master is a small price to pay for the simplicity, reliability, performance, and flexibility we gain by storing the metadata in memory.

2.6.2 Chunk Locations

The master does not keep a persistent record of which chunkservers have a replica of a given chunk. It simply polls chunkservers for that information at startup. The master can keep itself up-to-date thereafter because it controls all chunk placement and monitors chunkserver status with regular *HeartBeat* messages.

We initially attempted to keep chunk location information persistently at the master, but we decided that it was much simpler to request the data from chunkservers at startup, and periodically thereafter. This eliminated the problem of keeping the master and chunkservers in sync as chunkservers join and leave the cluster, change names, fail, restart, and so on. In a cluster with hundreds of servers, these events happen all too often.

Another way to understand this design decision is to realize that a chunkserver has the final word over what chunks it does or does not have on its own disks. There is no point in trying to maintain a consistent view of this information on the master because errors on a chunkserver may cause chunks to vanish spontaneously (e.g., a disk may go bad and be disabled) or an operator may rename a chunkserver.

2.6.3 Operation Log

The operation log contains a historical record of critical metadata changes. It is central to GFS. Not only is it the only persistent record of metadata, but it also serves as a logical time line that defines the order of concurrent operations. Files and chunks, as well as their versions (see Section 4.5), are all uniquely and eternally identified by the logical times at which they were created.

Since the operation log is critical, we must store it reliably and not make changes visible to clients until metadata changes are made persistent. Otherwise, we effectively lose the whole file system or recent client operations even if the chunks themselves survive. Therefore, we replicate it on multiple remote machines and respond to a client operation only after flushing the corresponding log record to disk both locally and remotely. The master batches several log records together before flushing thereby reducing the impact of flushing and replication on overall system throughput.

The master recovers its file system state by replaying the operation log. To minimize startup time, we must keep the log small. The master checkpoints its state whenever the log grows beyond a certain size so that it can recover by loading the latest checkpoint from local disk and replaying only the

	Write	Record Append
Serial success	<i>defined</i>	<i>defined interspersed with inconsistent</i>
Concurrent successes	<i>consistent but undefined</i>	
Failure	<i>inconsistent</i>	

Table 1: File Region State After Mutation

limited number of log records after that. The checkpoint is in a compact B-tree like form that can be directly mapped into memory and used for namespace lookup without extra parsing. This further speeds up recovery and improves availability.

Because building a checkpoint can take a while, the master's internal state is structured in such a way that a new checkpoint can be created without delaying incoming mutations. The master switches to a new log file and creates the new checkpoint in a separate thread. The new checkpoint includes all mutations before the switch. It can be created in a minute or so for a cluster with a few million files. When completed, it is written to disk both locally and remotely.

Recovery needs only the latest complete checkpoint and subsequent log files. Older checkpoints and log files can be freely deleted, though we keep a few around to guard against catastrophes. A failure during checkpointing does not affect correctness because the recovery code detects and skips incomplete checkpoints.

2.7 Consistency Model

GFS has a relaxed consistency model that supports our highly distributed applications well but remains relatively simple and efficient to implement. We now discuss GFS's guarantees and what they mean to applications. We also highlight how GFS maintains these guarantees but leave the details to other parts of the paper.

2.7.1 Guarantees by GFS

File namespace mutations (e.g., file creation) are atomic. They are handled exclusively by the master: namespace locking guarantees atomicity and correctness (Section 4.1); the master's operation log defines a global total order of these operations (Section 2.6.3).

The state of a file region after a data mutation depends on the type of mutation, whether it succeeds or fails, and whether there are concurrent mutations. Table 1 summarizes the result. A file region is *consistent* if all clients will always see the same data, regardless of which replicas they read from. A region is *defined* after a file data mutation if it is consistent and clients will see what the mutation writes in its entirety. When a mutation succeeds without interference from concurrent writers, the affected region is defined (and by implication consistent): all clients will always see what the mutation has written. Concurrent successful mutations leave the region undefined but consistent: all clients see the same data, but it may not reflect what any one mutation has written. Typically, it consists of mingled fragments from multiple mutations. A failed mutation makes the region inconsistent (hence also undefined): different clients may see different data at different times. We describe below how our applications can distinguish defined regions from undefined

regions. The applications do not need to further distinguish between different kinds of undefined regions.

Data mutations may be *writes* or *record appends*. A write causes data to be written at an application-specified file offset. A record append causes data (the “record”) to be appended *atomically at least once* even in the presence of concurrent mutations, but at an offset of GFS’s choosing (Section 3.3). (In contrast, a “regular” append is merely a write at an offset that the client believes to be the current end of file.) The offset is returned to the client and marks the beginning of a defined region that contains the record. In addition, GFS may insert padding or record duplicates in between. They occupy regions considered to be inconsistent and are typically dwarfed by the amount of user data.

After a sequence of successful mutations, the mutated file region is guaranteed to be defined and contain the data written by the last mutation. GFS achieves this by (a) applying mutations to a chunk in the same order on all its replicas (Section 3.1), and (b) using chunk version numbers to detect any replica that has become stale because it has missed mutations while its chunkserver was down (Section 4.5). Stale replicas will never be involved in a mutation or given to clients asking the master for chunk locations. They are garbage collected at the earliest opportunity.

Since clients cache chunk locations, they may read from a stale replica before that information is refreshed. This window is limited by the cache entry’s timeout and the next open of the file, which purges from the cache all chunk information for that file. Moreover, as most of our files are append-only, a stale replica usually returns a premature end of chunk rather than outdated data. When a reader retries and contacts the master, it will immediately get current chunk locations.

Long after a successful mutation, component failures can of course still corrupt or destroy data. GFS identifies failed chunkservers by regular handshakes between master and all chunkservers and detects data corruption by checksumming (Section 5.2). Once a problem surfaces, the data is restored from valid replicas as soon as possible (Section 4.3). A chunk is lost irreversibly only if all its replicas are lost before GFS can react, typically within minutes. Even in this case, it becomes unavailable, not corrupted: applications receive clear errors rather than corrupt data.

2.7.2 Implications for Applications

GFS applications can accommodate the relaxed consistency model with a few simple techniques already needed for other purposes: relying on appends rather than overwrites, checkpointing, and writing self-validating, self-identifying records.

Practically all our applications mutate files by appending rather than overwriting. In one typical use, a writer generates a file from beginning to end. It atomically renames the file to a permanent name after writing all the data, or periodically checkpoints how much has been successfully written. Checkpoints may also include application-level checksums. Readers verify and process only the file region up to the last checkpoint, which is known to be in the defined state. Regardless of consistency and concurrency issues, this approach has served us well. Appending is far more efficient and more resilient to application failures than random writes. Checkpointing allows writers to restart incrementally and keeps readers from processing successfully written

file data that is still incomplete from the application’s perspective.

In the other typical use, many writers concurrently append to a file for merged results or as a producer-consumer queue. Record append’s append-at-least-once semantics preserves each writer’s output. Readers deal with the occasional padding and duplicates as follows. Each record prepared by the writer contains extra information like checksums so that its validity can be verified. A reader can identify and discard extra padding and record fragments using the checksums. If it cannot tolerate the occasional duplicates (e.g., if they would trigger non-idempotent operations), it can filter them out using unique identifiers in the records, which are often needed anyway to name corresponding application entities such as web documents. These functionalities for record I/O (except duplicate removal) are in library code shared by our applications and applicable to other file interface implementations at Google. With that, the same sequence of records, plus rare duplicates, is always delivered to the record reader.

3. SYSTEM INTERACTIONS

We designed the system to minimize the master’s involvement in all operations. With that background, we now describe how the client, master, and chunkservers interact to implement data mutations, atomic record append, and snapshot.

3.1 Leases and Mutation Order

A mutation is an operation that changes the contents or metadata of a chunk such as a write or an append operation. Each mutation is performed at all the chunk’s replicas. We use leases to maintain a consistent mutation order across replicas. The master grants a chunk lease to one of the replicas, which we call the *primary*. The primary picks a serial order for all mutations to the chunk. All replicas follow this order when applying mutations. Thus, the global mutation order is defined first by the lease grant order chosen by the master, and within a lease by the serial numbers assigned by the primary.

The lease mechanism is designed to minimize management overhead at the master. A lease has an initial timeout of 60 seconds. However, as long as the chunk is being mutated, the primary can request and typically receive extensions from the master indefinitely. These extension requests and grants are piggybacked on the *HeartBeat* messages regularly exchanged between the master and all chunkservers. The master may sometimes try to revoke a lease before it expires (e.g., when the master wants to disable mutations on a file that is being renamed). Even if the master loses communication with a primary, it can safely grant a new lease to another replica after the old lease expires.

In Figure 2, we illustrate this process by following the control flow of a write through these numbered steps.

1. The client asks the master which chunkserver holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a replica it chooses (not shown).
2. The master replies with the identity of the primary and the locations of the other (*secondary*) replicas. The client caches this data for future mutations. It needs to contact the master again only when the primary

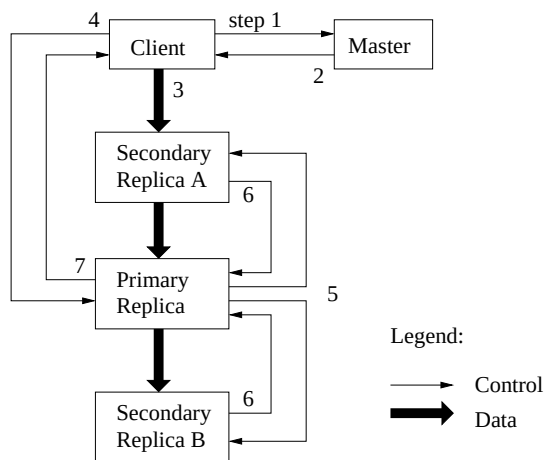


Figure 2: Write Control and Data Flow

becomes unreachable or replies that it no longer holds a lease.

3. The client pushes the data to all the replicas. A client can do so in any order. Each chunkserver will store the data in an internal LRU buffer cache until the data is used or aged out. By decoupling the data flow from the control flow, we can improve performance by scheduling the expensive data flow based on the network topology regardless of which chunkserver is the primary. Section 3.2 discusses this further.
4. Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The request identifies the data pushed earlier to all of the replicas. The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients, which provides the necessary serialization. It applies the mutation to its own local state in serial number order.
5. The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.
6. The secondaries all reply to the primary indicating that they have completed the operation.
7. The primary replies to the client. Any errors encountered at any of the replicas are reported to the client. In case of errors, the write may have succeeded at the primary and an arbitrary subset of the secondary replicas. (If it had failed at the primary, it would not have been assigned a serial number and forwarded.) The client request is considered to have failed, and the modified region is left in an inconsistent state. Our client code handles such errors by retrying the failed mutation. It will make a few attempts at steps (3) through (7) before falling back to a retry from the beginning of the write.

If a write by the application is large or straddles a chunk boundary, GFS client code breaks it down into multiple write operations. They all follow the control flow described above but may be interleaved with and overwritten by concurrent operations from other clients. Therefore, the shared

file region may end up containing fragments from different clients, although the replicas will be identical because the individual operations are completed successfully in the same order on all replicas. This leaves the file region in consistent but undefined state as noted in Section 2.7.

3.2 Data Flow

We decouple the flow of data from the flow of control to use the network efficiently. While control flows from the client to the primary and then to all secondaries, data is pushed linearly along a carefully picked chain of chunkservers in a pipelined fashion. Our goals are to fully utilize each machine's network bandwidth, avoid network bottlenecks and high-latency links, and minimize the latency to push through all the data.

To fully utilize each machine's network bandwidth, the data is pushed linearly along a chain of chunkservers rather than distributed in some other topology (e.g., tree). Thus, each machine's full outbound bandwidth is used to transfer the data as fast as possible rather than divided among multiple recipients.

To avoid network bottlenecks and high-latency links (e.g., inter-switch links are often both) as much as possible, each machine forwards the data to the "closest" machine in the network topology that has not received it. Suppose the client is pushing data to chunkservers S1 through S4. It sends the data to the closest chunkserver, say S1. S1 forwards it to the closest chunkserver S2 through S4 closest to S1, say S2. Similarly, S2 forwards it to S3 or S4, whichever is closer to S2, and so on. Our network topology is simple enough that "distances" can be accurately estimated from IP addresses.

Finally, we minimize latency by pipelining the data transfer over TCP connections. Once a chunkserver receives some data, it starts forwarding immediately. Pipelining is especially helpful to us because we use a switched network with full-duplex links. Sending the data immediately does not reduce the receive rate. Without network congestion, the ideal elapsed time for transferring B bytes to R replicas is $B/T + RL$ where T is the network throughput and L is latency to transfer bytes between two machines. Our network links are typically 100 Mbps (T), and L is far below 1 ms. Therefore, 1 MB can ideally be distributed in about 80 ms.

3.3 Atomic Record Appends

GFS provides an atomic append operation called *record append*. In a traditional write, the client specifies the offset at which data is to be written. Concurrent writes to the same region are not serializable: the region may end up containing data fragments from multiple clients. In a record append, however, the client specifies only the data. GFS appends it to the file at least once atomically (i.e., as one continuous sequence of bytes) at an offset of GFS's choosing and returns that offset to the client. This is similar to writing to a file opened in `O_APPEND` mode in Unix without the race conditions when multiple writers do so concurrently.

Record append is heavily used by our distributed applications in which many clients on different machines append to the same file concurrently. Clients would need additional complicated and expensive synchronization, for example through a distributed lock manager, if they do so with traditional writes. In our workloads, such files often

serve as multiple-producer/single-consumer queues or contain merged results from many different clients.

Record append is a kind of mutation and follows the control flow in Section 3.1 with only a little extra logic at the primary. The client pushes the data to all replicas of the last chunk of the file. Then, it sends its request to the primary. The primary checks to see if appending the record to the current chunk would cause the chunk to exceed the maximum size (64 MB). If so, it pads the chunk to the maximum size, tells secondaries to do the same, and replies to the client indicating that the operation should be retried on the next chunk. (Record append is restricted to be at most one-fourth of the maximum chunk size to keep worst-case fragmentation at an acceptable level.) If the record fits within the maximum size, which is the common case, the primary appends the data to its replica, tells the secondaries to write the data at the exact offset where it has, and finally replies success to the client.

If a record append fails at any replica, the client retries the operation. As a result, replicas of the same chunk may contain different data possibly including duplicates of the same record in whole or in part. GFS does not guarantee that all replicas are bitwise identical. It only guarantees that the data is written at least once as an atomic unit. This property follows readily from the simple observation that for the operation to report success, the data must have been written at the same offset on all replicas of some chunk. Furthermore, after this, all replicas are at least as long as the end of record and therefore any future record will be assigned a higher offset or a different chunk even if a different replica later becomes the primary. In terms of our consistency guarantees, the regions in which successful record append operations have written their data are defined (hence consistent), whereas intervening regions are inconsistent (hence undefined). Our applications can deal with inconsistent regions as we discussed in Section 2.7.2.

3.4 Snapshot

The snapshot operation makes a copy of a file or a directory tree (the “source”) almost instantaneously, while minimizing any interruptions of ongoing mutations. Our users use it to quickly create branch copies of huge data sets (and often copies of those copies, recursively), or to checkpoint the current state before experimenting with changes that can later be committed or rolled back easily.

Like AFS [5], we use standard copy-on-write techniques to implement snapshots. When the master receives a snapshot request, it first revokes any outstanding leases on the chunks in the files it is about to snapshot. This ensures that any subsequent writes to these chunks will require an interaction with the master to find the lease holder. This will give the master an opportunity to create a new copy of the chunk first.

After the leases have been revoked or have expired, the master logs the operation to disk. It then applies this log record to its in-memory state by duplicating the metadata for the source file or directory tree. The newly created snapshot files point to the same chunks as the source files.

The first time a client wants to write to a chunk *C* after the snapshot operation, it sends a request to the master to find the current lease holder. The master notices that the reference count for chunk *C* is greater than one. It defers replying to the client request and instead picks a new chunk

handle *C'*. It then asks each chunkserver that has a current replica of *C* to create a new chunk called *C'*. By creating the new chunk on the same chunkservers as the original, we ensure that the data can be copied locally, not over the network (our disks are about three times as fast as our 100 Mb Ethernet links). From this point, request handling is no different from that for any chunk: the master grants one of the replicas a lease on the new chunk *C'* and replies to the client, which can write the chunk normally, not knowing that it has just been created from an existing chunk.

4. MASTER OPERATION

The master executes all namespace operations. In addition, it manages chunk replicas throughout the system: it makes placement decisions, creates new chunks and hence replicas, and coordinates various system-wide activities to keep chunks fully replicated, to balance load across all the chunkservers, and to reclaim unused storage. We now discuss each of these topics.

4.1 Namespace Management and Locking

Many master operations can take a long time: for example, a snapshot operation has to revoke chunkserver leases on all chunks covered by the snapshot. We do not want to delay other master operations while they are running. Therefore, we allow multiple operations to be active and use locks over regions of the namespace to ensure proper serialization.

Unlike many traditional file systems, GFS does not have a per-directory data structure that lists all the files in that directory. Nor does it support aliases for the same file or directory (i.e., hard or symbolic links in Unix terms). GFS logically represents its namespace as a lookup table mapping full pathnames to metadata. With prefix compression, this table can be efficiently represented in memory. Each node in the namespace tree (either an absolute file name or an absolute directory name) has an associated read-write lock.

Each master operation acquires a set of locks before it runs. Typically, if it involves `/d1/d2/.../dn/leaf`, it will acquire read-locks on the directory names `/d1`, `/d1/d2`, ..., `/d1/d2/.../dn`, and either a read lock or a write lock on the full pathname `/d1/d2/.../dn/leaf`. Note that `leaf` may be a file or directory depending on the operation.

We now illustrate how this locking mechanism can prevent a file `/home/user/foo` from being created while `/home/user` is being snapshot to `/save/user`. The snapshot operation acquires read locks on `/home` and `/save`, and write locks on `/home/user` and `/save/user`. The file creation acquires read locks on `/home` and `/home/user`, and a write lock on `/home/user/foo`. The two operations will be serialized properly because they try to obtain conflicting locks on `/home/user`. File creation does not require a write lock on the parent directory because there is no “directory”, or *inode*-like, data structure to be protected from modification. The read lock on the name is sufficient to protect the parent directory from deletion.

One nice property of this locking scheme is that it allows concurrent mutations in the same directory. For example, multiple file creations can be executed concurrently in the same directory: each acquires a read lock on the directory name and a write lock on the file name. The read lock on the directory name suffices to prevent the directory from being deleted, renamed, or snapshot. The write locks on

file names serialize attempts to create a file with the same name twice.

Since the namespace can have many nodes, read-write lock objects are allocated lazily and deleted once they are not in use. Also, locks are acquired in a consistent total order to prevent deadlock: they are first ordered by level in the namespace tree and lexicographically within the same level.

4.2 Replica Placement

A GFS cluster is highly distributed at more levels than one. It typically has hundreds of chunkservers spread across many machine racks. These chunkservers in turn may be accessed from hundreds of clients from the same or different racks. Communication between two machines on different racks may cross one or more network switches. Additionally, bandwidth into or out of a rack may be less than the aggregate bandwidth of all the machines within the rack. Multi-level distribution presents a unique challenge to distribute data for scalability, reliability, and availability.

The chunk replica placement policy serves two purposes: maximize data reliability and availability, and maximize network bandwidth utilization. For both, it is not enough to spread replicas across machines, which only guards against disk or machine failures and fully utilizes each machine's network bandwidth. We must also spread chunk replicas across racks. This ensures that some replicas of a chunk will survive and remain available even if an entire rack is damaged or offline (for example, due to failure of a shared resource like a network switch or power circuit). It also means that traffic, especially reads, for a chunk can exploit the aggregate bandwidth of multiple racks. On the other hand, write traffic has to flow through multiple racks, a tradeoff we make willingly.

4.3 Creation, Re-replication, Rebalancing

Chunk replicas are created for three reasons: chunk creation, re-replication, and rebalancing.

When the master *creates* a chunk, it chooses where to place the initially empty replicas. It considers several factors. (1) We want to place new replicas on chunkservers with below-average disk space utilization. Over time this will equalize disk utilization across chunkservers. (2) We want to limit the number of “recent” creations on each chunkserver. Although creation itself is cheap, it reliably predicts imminent heavy write traffic because chunks are created when demanded by writes, and in our append-once-read-many workload they typically become practically read-only once they have been completely written. (3) As discussed above, we want to spread replicas of a chunk across racks.

The master *re-replicates* a chunk as soon as the number of available replicas falls below a user-specified goal. This could happen for various reasons: a chunkserver becomes unavailable, it reports that its replica may be corrupted, one of its disks is disabled because of errors, or the replication goal is increased. Each chunk that needs to be re-replicated is prioritized based on several factors. One is how far it is from its replication goal. For example, we give higher priority to a chunk that has lost two replicas than to a chunk that has lost only one. In addition, we prefer to first re-replicate chunks for live files as opposed to chunks that belong to recently deleted files (see Section 4.4). Finally, to minimize the impact of failures on running applications, we boost the priority of any chunk that is blocking client progress.

The master picks the highest priority chunk and “clones” it by instructing some chunkserver to copy the chunk data directly from an existing valid replica. The new replica is placed with goals similar to those for creation: equalizing disk space utilization, limiting active clone operations on any single chunkserver, and spreading replicas across racks. To keep cloning traffic from overwhelming client traffic, the master limits the numbers of active clone operations both for the cluster and for each chunkserver. Additionally, each chunkserver limits the amount of bandwidth it spends on each clone operation by throttling its read requests to the source chunkserver.

Finally, the master *rebalances* replicas periodically: it examines the current replica distribution and moves replicas for better disk space and load balancing. Also through this process, the master gradually fills up a new chunkserver rather than instantly swamps it with new chunks and the heavy write traffic that comes with them. The placement criteria for the new replica are similar to those discussed above. In addition, the master must also choose which existing replica to remove. In general, it prefers to remove those on chunkservers with below-average free space so as to equalize disk space usage.

4.4 Garbage Collection

After a file is deleted, GFS does not immediately reclaim the available physical storage. It does so only lazily during regular garbage collection at both the file and chunk levels. We find that this approach makes the system much simpler and more reliable.

4.4.1 Mechanism

When a file is deleted by the application, the master logs the deletion immediately just like other changes. However instead of reclaiming resources immediately, the file is just renamed to a hidden name that includes the deletion timestamp. During the master's regular scan of the file system namespace, it removes any such hidden files if they have existed for more than three days (the interval is configurable). Until then, the file can still be read under the new, special name and can be undeleted by renaming it back to normal. When the hidden file is removed from the namespace, its in-memory metadata is erased. This effectively severs its links to all its chunks.

In a similar regular scan of the chunk namespace, the master identifies orphaned chunks (i.e., those not reachable from any file) and erases the metadata for those chunks. In a *HeartBeat* message regularly exchanged with the master, each chunkserver reports a subset of the chunks it has, and the master replies with the identity of all chunks that are no longer present in the master's metadata. The chunkserver is free to delete its replicas of such chunks.

4.4.2 Discussion

Although distributed garbage collection is a hard problem that demands complicated solutions in the context of programming languages, it is quite simple in our case. We can easily identify all references to chunks: they are in the file-to-chunk mappings maintained exclusively by the master. We can also easily identify all the chunk replicas: they are Linux files under designated directories on each chunkserver. Any such replica not known to the master is “garbage.”

The garbage collection approach to storage reclamation offers several advantages over eager deletion. First, it is simple and reliable in a large-scale distributed system where component failures are common. Chunk creation may succeed on some chunkservers but not others, leaving replicas that the master does not know exist. Replica deletion messages may be lost, and the master has to remember to resend them across failures, both its own and the chunkserver's. Garbage collection provides a uniform and dependable way to clean up any replicas not known to be useful. Second, it merges storage reclamation into the regular background activities of the master, such as the regular scans of namespaces and handshakes with chunkservers. Thus, it is done in batches and the cost is amortized. Moreover, it is done only when the master is relatively free. The master can respond more promptly to client requests that demand timely attention. Third, the delay in reclaiming storage provides a safety net against accidental, irreversible deletion.

In our experience, the main disadvantage is that the delay sometimes hinders user effort to fine tune usage when storage is tight. Applications that repeatedly create and delete temporary files may not be able to reuse the storage right away. We address these issues by expediting storage reclamation if a deleted file is explicitly deleted again. We also allow users to apply different replication and reclamation policies to different parts of the namespace. For example, users can specify that all the chunks in the files within some directory tree are to be stored without replication, and any deleted files are immediately and irrevocably removed from the file system state.

4.5 Stale Replica Detection

Chunk replicas may become stale if a chunkserver fails and misses mutations to the chunk while it is down. For each chunk, the master maintains a *chunk version number* to distinguish between up-to-date and stale replicas.

Whenever the master grants a new lease on a chunk, it increases the chunk version number and informs the up-to-date replicas. The master and these replicas all record the new version number in their persistent state. This occurs before any client is notified and therefore before it can start writing to the chunk. If another replica is currently unavailable, its chunk version number will not be advanced. The master will detect that this chunkserver has a stale replica when the chunkserver restarts and reports its set of chunks and their associated version numbers. If the master sees a version number greater than the one in its records, the master assumes that it failed when granting the lease and so takes the higher version to be up-to-date.

The master removes stale replicas in its regular garbage collection. Before that, it effectively considers a stale replica not to exist at all when it replies to client requests for chunk information. As another safeguard, the master includes the chunk version number when it informs clients which chunkserver holds a lease on a chunk or when it instructs a chunkserver to read the chunk from another chunkserver in a cloning operation. The client or the chunkserver verifies the version number when it performs the operation so that it is always accessing up-to-date data.

5. FAULT TOLERANCE AND DIAGNOSIS

One of our greatest challenges in designing the system is dealing with frequent component failures. The quality and

quantity of components together make these problems more the norm than the exception: we cannot completely trust the machines, nor can we completely trust the disks. Component failures can result in an unavailable system or, worse, corrupted data. We discuss how we meet these challenges and the tools we have built into the system to diagnose problems when they inevitably occur.

5.1 High Availability

Among hundreds of servers in a GFS cluster, some are bound to be unavailable at any given time. We keep the overall system highly available with two simple yet effective strategies: fast recovery and replication.

5.1.1 Fast Recovery

Both the master and the chunkserver are designed to restore their state and start in seconds no matter how they terminated. In fact, we do not distinguish between normal and abnormal termination; servers are routinely shut down just by killing the process. Clients and other servers experience a minor hiccup as they time out on their outstanding requests, reconnect to the restarted server, and retry. Section 6.2.2 reports observed startup times.

5.1.2 Chunk Replication

As discussed earlier, each chunk is replicated on multiple chunkservers on different racks. Users can specify different replication levels for different parts of the file namespace. The default is three. The master clones existing replicas as needed to keep each chunk fully replicated as chunkservers go offline or detect corrupted replicas through checksum verification (see Section 5.2). Although replication has served us well, we are exploring other forms of cross-server redundancy such as parity or erasure codes for our increasing read-only storage requirements. We expect that it is challenging but manageable to implement these more complicated redundancy schemes in our very loosely coupled system because our traffic is dominated by appends and reads rather than small random writes.

5.1.3 Master Replication

The master state is replicated for reliability. Its operation log and checkpoints are replicated on multiple machines. A mutation to the state is considered committed only after its log record has been flushed to disk locally and on all master replicas. For simplicity, one master process remains in charge of all mutations as well as background activities such as garbage collection that change the system internally. When it fails, it can restart almost instantly. If its machine or disk fails, monitoring infrastructure outside GFS starts a new master process elsewhere with the replicated operation log. Clients use only the canonical name of the master (e.g. `gfs-test`), which is a DNS alias that can be changed if the master is relocated to another machine.

Moreover, “shadow” masters provide read-only access to the file system even when the primary master is down. They are shadows, not mirrors, in that they may lag the primary slightly, typically fractions of a second. They enhance read availability for files that are not being actively mutated or applications that do not mind getting slightly stale results. In fact, since file content is read from chunkservers, applications do not observe stale file content. What could be

stale within short windows is file metadata, like directory contents or access control information.

To keep itself informed, a shadow master reads a replica of the growing operation log and applies the same sequence of changes to its data structures exactly as the primary does. Like the primary, it polls chunkservers at startup (and infrequently thereafter) to locate chunk replicas and exchanges frequent handshake messages with them to monitor their status. It depends on the primary master only for replica location updates resulting from the primary's decisions to create and delete replicas.

5.2 Data Integrity

Each chunkserver uses checksumming to detect corruption of stored data. Given that a GFS cluster often has thousands of disks on hundreds of machines, it regularly experiences disk failures that cause data corruption or loss on both the read and write paths. (See Section 7 for one cause.) We can recover from corruption using other chunk replicas, but it would be impractical to detect corruption by comparing replicas across chunkservers. Moreover, divergent replicas may be legal: the semantics of GFS mutations, in particular atomic record append as discussed earlier, does not guarantee identical replicas. Therefore, each chunkserver must independently verify the integrity of its own copy by maintaining checksums.

A chunk is broken up into 64 KB blocks. Each has a corresponding 32 bit checksum. Like other metadata, checksums are kept in memory and stored persistently with logging, separate from user data.

For reads, the chunkserver verifies the checksum of data blocks that overlap the read range before returning any data to the requester, whether a client or another chunkserver. Therefore chunkservers will not propagate corruptions to other machines. If a block does not match the recorded checksum, the chunkserver returns an error to the requestor and reports the mismatch to the master. In response, the requestor will read from other replicas, while the master will clone the chunk from another replica. After a valid new replica is in place, the master instructs the chunkserver that reported the mismatch to delete its replica.

Checksumming has little effect on read performance for several reasons. Since most of our reads span at least a few blocks, we need to read and checksum only a relatively small amount of extra data for verification. GFS client code further reduces this overhead by trying to align reads at checksum block boundaries. Moreover, checksum lookups and comparison on the chunkserver are done without any I/O, and checksum calculation can often be overlapped with I/Os.

Checksum computation is heavily optimized for writes that append to the end of a chunk (as opposed to writes that overwrite existing data) because they are dominant in our workloads. We just incrementally update the checksum for the last partial checksum block, and compute new checksums for any brand new checksum blocks filled by the append. Even if the last partial checksum block is already corrupted and we fail to detect it now, the new checksum value will not match the stored data, and the corruption will be detected as usual when the block is next read.

In contrast, if a write overwrites an existing range of the chunk, we must read and verify the first and last blocks of the range being overwritten, then perform the write, and

finally compute and record the new checksums. If we do not verify the first and last blocks before overwriting them partially, the new checksums may hide corruption that exists in the regions not being overwritten.

During idle periods, chunkservers can scan and verify the contents of inactive chunks. This allows us to detect corruption in chunks that are rarely read. Once the corruption is detected, the master can create a new uncorrupted replica and delete the corrupted replica. This prevents an inactive but corrupted chunk replica from fooling the master into thinking that it has enough valid replicas of a chunk.

5.3 Diagnostic Tools

Extensive and detailed diagnostic logging has helped immeasurably in problem isolation, debugging, and performance analysis, while incurring only a minimal cost. Without logs, it is hard to understand transient, non-repeatable interactions between machines. GFS servers generate diagnostic logs that record many significant events (such as chunkservers going up and down) and all RPC requests and replies. These diagnostic logs can be freely deleted without affecting the correctness of the system. However, we try to keep these logs around as far as space permits.

The RPC logs include the exact requests and responses sent on the wire, except for the file data being read or written. By matching requests with replies and collating RPC records on different machines, we can reconstruct the entire interaction history to diagnose a problem. The logs also serve as traces for load testing and performance analysis.

The performance impact of logging is minimal (and far outweighed by the benefits) because these logs are written sequentially and asynchronously. The most recent events are also kept in memory and available for continuous online monitoring.

6. MEASUREMENTS

In this section we present a few micro-benchmarks to illustrate the bottlenecks inherent in the GFS architecture and implementation, and also some numbers from real clusters in use at Google.

6.1 Micro-benchmarks

We measured performance on a GFS cluster consisting of one master, two master replicas, 16 chunkservers, and 16 clients. Note that this configuration was set up for ease of testing. Typical clusters have hundreds of chunkservers and hundreds of clients.

All the machines are configured with dual 1.4 GHz PIII processors, 2 GB of memory, two 80 GB 5400 rpm disks, and a 100 Mbps full-duplex Ethernet connection to an HP 2524 switch. All 19 GFS server machines are connected to one switch, and all 16 client machines to the other. The two switches are connected with a 1 Gbps link.

6.1.1 Reads

N clients read simultaneously from the file system. Each client reads a randomly selected 4 MB region from a 320 GB file set. This is repeated 256 times so that each client ends up reading 1 GB of data. The chunkservers taken together have only 32 GB of memory, so we expect at most a 10% hit rate in the Linux buffer cache. Our results should be close to cold cache results.

Figure 3(a) shows the aggregate read rate for N clients and its theoretical limit. The limit peaks at an aggregate of 125 MB/s when the 1 Gbps link between the two switches is saturated, or 12.5 MB/s per client when its 100 Mbps network interface gets saturated, whichever applies. The observed read rate is 10 MB/s, or 80% of the per-client limit, when just one client is reading. The aggregate read rate reaches 94 MB/s, about 75% of the 125 MB/s link limit, for 16 readers, or 6 MB/s per client. The efficiency drops from 80% to 75% because as the number of readers increases, so does the probability that multiple readers simultaneously read from the same chunkserver.

6.1.2 Writes

N clients write simultaneously to N distinct files. Each client writes 1 GB of data to a new file in a series of 1 MB writes. The aggregate write rate and its theoretical limit are shown in Figure 3(b). The limit plateaus at 67 MB/s because we need to write each byte to 3 of the 16 chunkservers, each with a 12.5 MB/s input connection.

The write rate for one client is 6.3 MB/s, about half of the limit. The main culprit for this is our network stack. It does not interact very well with the pipelining scheme we use for pushing data to chunk replicas. Delays in propagating data from one replica to another reduce the overall write rate.

Aggregate write rate reaches 35 MB/s for 16 clients (or 2.2 MB/s per client), about half the theoretical limit. As in the case of reads, it becomes more likely that multiple clients write concurrently to the same chunkserver as the number of clients increases. Moreover, collision is more likely for 16 writers than for 16 readers because each write involves three different replicas.

Writes are slower than we would like. In practice this has not been a major problem because even though it increases the latencies as seen by individual clients, it does not significantly affect the aggregate write bandwidth delivered by the system to a large number of clients.

6.1.3 Record Appends

Figure 3(c) shows record append performance. N clients append simultaneously to a single file. Performance is limited by the network bandwidth of the chunkservers that store the last chunk of the file, independent of the number of clients. It starts at 6.0 MB/s for one client and drops to 4.8 MB/s for 16 clients, mostly due to congestion and variances in network transfer rates seen by different clients.

Our applications tend to produce multiple such files concurrently. In other words, N clients append to M shared files simultaneously where both N and M are in the dozens or hundreds. Therefore, the chunkserver network congestion in our experiment is not a significant issue in practice because a client can make progress on writing one file while the chunkservers for another file are busy.

6.2 Real World Clusters

We now examine two clusters in use within Google that are representative of several others like them. Cluster A is used regularly for research and development by over a hundred engineers. A typical task is initiated by a human user and runs up to several hours. It reads through a few MBs to a few TBs of data, transforms or analyzes the data, and writes the results back to the cluster. Cluster B is primarily used for production data processing. The tasks last much

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

Table 2: Characteristics of two GFS clusters

longer and continuously generate and process multi-TB data sets with only occasional human intervention. In both cases, a single “task” consists of many processes on many machines reading and writing many files simultaneously.

6.2.1 Storage

As shown by the first five entries in the table, both clusters have hundreds of chunkservers, support many TBs of disk space, and are fairly but not completely full. “Used space” includes all chunk replicas. Virtually all files are replicated three times. Therefore, the clusters store 18 TB and 52 TB of file data respectively.

The two clusters have similar numbers of files, though B has a larger proportion of dead files, namely files which were deleted or replaced by a new version but whose storage have not yet been reclaimed. It also has more chunks because its files tend to be larger.

6.2.2 Metadata

The chunkservers in aggregate store tens of GBs of metadata, mostly the checksums for 64 KB blocks of user data. The only other metadata kept at the chunkservers is the chunk version number discussed in Section 4.5.

The metadata kept at the master is much smaller, only tens of MBs, or about 100 bytes per file on average. This agrees with our assumption that the size of the master’s memory does not limit the system’s capacity in practice. Most of the per-file metadata is the file names stored in a prefix-compressed form. Other metadata includes file ownership and permissions, mapping from files to chunks, and each chunk’s current version. In addition, for each chunk we store the current replica locations and a reference count for implementing copy-on-write.

Each individual server, both chunkservers and the master, has only 50 to 100 MB of metadata. Therefore recovery is fast: it takes only a few seconds to read this metadata from disk before the server is able to answer queries. However, the master is somewhat hobbled for a period – typically 30 to 60 seconds – until it has fetched chunk location information from all chunkservers.

6.2.3 Read and Write Rates

Table 3 shows read and write rates for various time periods. Both clusters had been up for about one week when these measurements were taken. (The clusters had been restarted recently to upgrade to a new version of GFS.)

The average write rate was less than 30 MB/s since the restart. When we took these measurements, B was in the middle of a burst of write activity generating about 100 MB/s of data, which produced a 300 MB/s network load because writes are propagated to three replicas.

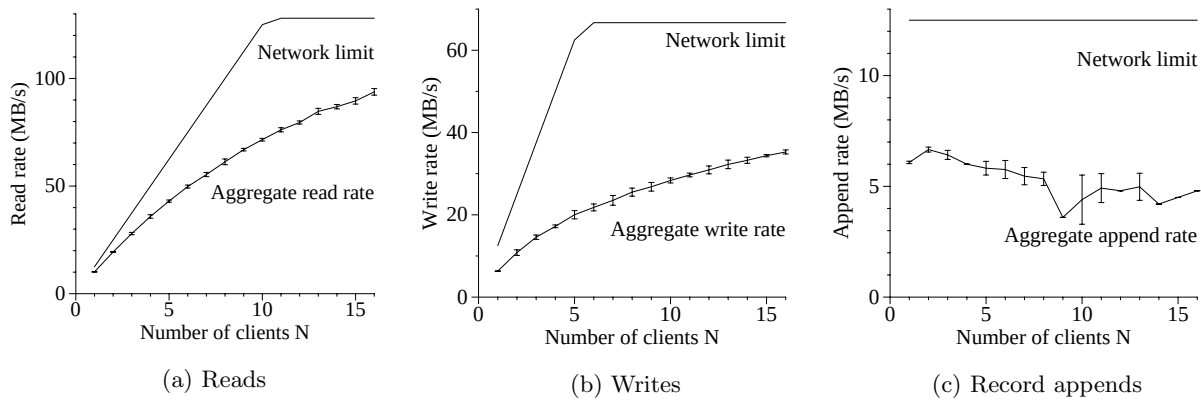


Figure 3: Aggregate Throughputs. Top curves show theoretical limits imposed by our network topology. Bottom curves show measured throughputs. They have error bars that show 95% confidence intervals, which are illegible in some cases because of low variance in measurements.

Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s

Table 3: Performance Metrics for Two GFS Clusters

The read rates were much higher than the write rates. The total workload consists of more reads than writes as we have assumed. Both clusters were in the middle of heavy read activity. In particular, A had been sustaining a read rate of 580 MB/s for the preceding week. Its network configuration can support 750 MB/s, so it was using its resources efficiently. Cluster B can support peak read rates of 1300 MB/s, but its applications were using just 380 MB/s.

6.2.4 Master Load

Table 3 also shows that the rate of operations sent to the master was around 200 to 500 operations per second. The master can easily keep up with this rate, and therefore is not a bottleneck for these workloads.

In an earlier version of GFS, the master was occasionally a bottleneck for some workloads. It spent most of its time sequentially scanning through large directories (which contained hundreds of thousands of files) looking for particular files. We have since changed the master data structures to allow efficient binary searches through the namespace. It can now easily support many thousands of file accesses per second. If necessary, we could speed it up further by placing name lookup caches in front of the namespace data structures.

6.2.5 Recovery Time

After a chunkserver fails, some chunks will become under-replicated and must be cloned to restore their replication levels. The time it takes to restore all such chunks depends on the amount of resources. In one experiment, we killed a single chunkserver in cluster B. The chunkserver had about

15,000 chunks containing 600 GB of data. To limit the impact on running applications and provide leeway for scheduling decisions, our default parameters limit this cluster to 91 concurrent clonings (40% of the number of chunkservers) where each clone operation is allowed to consume at most 6.25 MB/s (50 Mbps). All chunks were restored in 23.2 minutes, at an effective replication rate of 440 MB/s.

In another experiment, we killed two chunkservers each with roughly 16,000 chunks and 660 GB of data. This double failure reduced 266 chunks to having a single replica. These 266 chunks were cloned at a higher priority, and were all restored to at least 2x replication within 2 minutes, thus putting the cluster in a state where it could tolerate another chunkserver failure without data loss.

6.3 Workload Breakdown

In this section, we present a detailed breakdown of the workloads on two GFS clusters comparable but not identical to those in Section 6.2. Cluster X is for research and development while cluster Y is for production data processing.

6.3.1 Methodology and Caveats

These results include only client originated requests so that they reflect the workload generated by our applications for the file system as a whole. They do not include inter-server requests to carry out client requests or internal background activities, such as forwarded writes or rebalancing.

Statistics on I/O operations are based on information heuristically reconstructed from actual RPC requests logged by GFS servers. For example, GFS client code may break a read into multiple RPCs to increase parallelism, from which we infer the original read. Since our access patterns are highly stylized, we expect any error to be in the noise. Explicit logging by applications might have provided slightly more accurate data, but it is logistically impossible to recompile and restart thousands of running clients to do so and cumbersome to collect the results from as many machines.

One should be careful not to overly generalize from our workload. Since Google completely controls both GFS and its applications, the applications tend to be tuned for GFS, and conversely GFS is designed for these applications. Such mutual influence may also exist between general applications

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
0K	0.4	2.6	0	0	0	0
1B..1K	0.1	4.1	6.6	4.9	0.2	9.2
1K..8K	65.2	38.5	0.4	1.0	18.9	15.2
8K..64K	29.9	45.1	17.8	43.0	78.0	2.8
64K..128K	0.1	0.7	2.3	1.9	< .1	4.3
128K..256K	0.2	0.3	31.6	0.4	< .1	10.6
256K..512K	0.1	0.1	4.2	7.7	< .1	31.2
512K..1M	3.9	6.9	35.5	28.7	2.2	25.5
1M..inf	0.1	1.8	1.5	12.3	0.7	2.2

Table 4: Operations Breakdown by Size (%). For reads, the size is the amount of data actually read and transferred, rather than the amount requested.

and file systems, but the effect is likely more pronounced in our case.

6.3.2 Chunkserver Workload

Table 4 shows the distribution of operations by size. Read sizes exhibit a bimodal distribution. The small reads (under 64 KB) come from seek-intensive clients that look up small pieces of data within huge files. The large reads (over 512 KB) come from long sequential reads through entire files.

A significant number of reads return no data at all in cluster Y. Our applications, especially those in the production systems, often use files as producer-consumer queues. Producers append concurrently to a file while a consumer reads the end of file. Occasionally, no data is returned when the consumer outpaces the producers. Cluster X shows this less often because it is usually used for short-lived data analysis tasks rather than long-lived distributed applications.

Write sizes also exhibit a bimodal distribution. The large writes (over 256 KB) typically result from significant buffering within the writers. Writers that buffer less data, checkpoint or synchronize more often, or simply generate less data account for the smaller writes (under 64 KB).

As for record appends, cluster Y sees a much higher percentage of large record appends than cluster X does because our production systems, which use cluster Y, are more aggressively tuned for GFS.

Table 5 shows the total amount of data transferred in operations of various sizes. For all kinds of operations, the larger operations (over 256 KB) generally account for most of the bytes transferred. Small reads (under 64 KB) do transfer a small but significant portion of the read data because of the random seek workload.

6.3.3 Appends versus Writes

Record appends are heavily used especially in our production systems. For cluster X, the ratio of writes to record appends is 108:1 by bytes transferred and 8:1 by operation counts. For cluster Y, used by the production systems, the ratios are 3.7:1 and 2.5:1 respectively. Moreover, these ratios suggest that for both clusters record appends tend to be larger than writes. For cluster X, however, the overall usage of record append during the measured period is fairly low and so the results are likely skewed by one or two applications with particular buffer size choices.

As expected, our data mutation workload is dominated by appending rather than overwriting. We measured the amount of data overwritten on primary replicas. This ap-

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
1B..1K	< .1	< .1	< .1	< .1	< .1	< .1
1K..8K	13.8	3.9	< .1	< .1	< .1	0.1
8K..64K	11.4	9.3	2.4	5.9	2.3	0.3
64K..128K	0.3	0.7	0.3	0.3	22.7	1.2
128K..256K	0.8	0.6	16.5	0.2	< .1	5.8
256K..512K	1.4	0.3	3.4	7.7	< .1	38.4
512K..1M	65.9	55.1	74.1	58.0	.1	46.8
1M..inf	6.4	30.1	3.3	28.0	53.9	7.4

Table 5: Bytes Transferred Breakdown by Operation Size (%). For reads, the size is the amount of data actually read and transferred, rather than the amount requested. The two may differ if the read attempts to read beyond end of file, which by design is not uncommon in our workloads.

Cluster	X	Y
Open	26.1	16.3
Delete	0.7	1.5
FindLocation	64.3	65.8
FindLeaseHolder	7.8	13.4
FindMatchingFiles	0.6	2.2
All other combined	0.5	0.8

Table 6: Master Requests Breakdown by Type (%)

proximates the case where a client deliberately overwrites previous written data rather than appends new data. For cluster X, overwriting accounts for under 0.0001% of bytes mutated and under 0.0003% of mutation operations. For cluster Y, the ratios are both 0.05%. Although this is minute, it is still higher than we expected. It turns out that most of these overwrites came from client retries due to errors or timeouts. They are not part of the workload *per se* but a consequence of the retry mechanism.

6.3.4 Master Workload

Table 6 shows the breakdown by type of requests to the master. Most requests ask for chunk locations (*FindLocation*) for reads and lease holder information (*FindLeaseLocker*) for data mutations.

Clusters X and Y see significantly different numbers of *Delete* requests because cluster Y stores production data sets that are regularly regenerated and replaced with newer versions. Some of this difference is further hidden in the difference in *Open* requests because an old version of a file may be implicitly deleted by being opened for write from scratch (mode “w” in Unix open terminology).

FindMatchingFiles is a pattern matching request that supports “ls” and similar file system operations. Unlike other requests for the master, it may process a large part of the namespace and so may be expensive. Cluster Y sees it much more often because automated data processing tasks tend to examine parts of the file system to understand global application state. In contrast, cluster X’s applications are under more explicit user control and usually know the names of all needed files in advance.

7. EXPERIENCES

In the process of building and deploying GFS, we have experienced a variety of issues, some operational and some technical.

Initially, GFS was conceived as the backend file system for our production systems. Over time, the usage evolved to include research and development tasks. It started with little support for things like permissions and quotas but now includes rudimentary forms of these. While production systems are well disciplined and controlled, users sometimes are not. More infrastructure is required to keep users from interfering with one another.

Some of our biggest problems were disk and Linux related. Many of our disks claimed to be the Linux driver that they supported a range of IDE protocol versions but in fact responded reliably only to the more recent ones. Since the protocol versions are very similar, these drives mostly worked, but occasionally the mismatches would cause the drive and the kernel to disagree about the drive's state. This would corrupt data silently due to problems in the kernel. This problem motivated our use of checksums to detect data corruption, while concurrently we modified the kernel to handle these protocol mismatches.

Earlier we had some problems with Linux 2.2 kernels due to the cost of `fsync()`. Its cost is proportional to the size of the file rather than the size of the modified portion. This was a problem for our large operation logs especially before we implemented checkpointing. We worked around this for a time by using synchronous writes and eventually migrated to Linux 2.4.

Another Linux problem was a single reader-writer lock which any thread in an address space must hold when it pages in from disk (reader lock) or modifies the address space in an `mmap()` call (writer lock). We saw transient timeouts in our system under light load and looked hard for resource bottlenecks or sporadic hardware failures. Eventually, we found that this single lock blocked the primary network thread from mapping new data into memory while the disk threads were paging in previously mapped data. Since we are mainly limited by the network interface rather than by memory copy bandwidth, we worked around this by replacing `mmap()` with `pread()` at the cost of an extra copy.

Despite occasional problems, the availability of Linux code has helped us time and again to explore and understand system behavior. When appropriate, we improve the kernel and share the changes with the open source community.

8. RELATED WORK

Like other large distributed file systems such as AFS [5], GFS provides a location independent namespace which enables data to be moved transparently for load balance or fault tolerance. Unlike AFS, GFS spreads a file's data across storage servers in a way more akin to xFS [1] and Swift [3] in order to deliver aggregate performance and increased fault tolerance.

As disks are relatively cheap and replication is simpler than more sophisticated RAID [9] approaches, GFS currently uses only replication for redundancy and so consumes more raw storage than xFS or Swift.

In contrast to systems like AFS, xFS, Frangipani [12], and Intermezzo [6], GFS does not provide any caching below the file system interface. Our target workloads have little reuse within a single application run because they either stream through a large data set or randomly seek within it and read small amounts of data each time.

Some distributed file systems like Frangipani, xFS, Minnesota's GFS[11] and GPFS [10] remove the centralized server

and rely on distributed algorithms for consistency and management. We opt for the centralized approach in order to simplify the design, increase its reliability, and gain flexibility. In particular, a centralized master makes it much easier to implement sophisticated chunk placement and replication policies since the master already has most of the relevant information and controls how it changes. We address fault tolerance by keeping the master state small and fully replicated on other machines. Scalability and high availability (for reads) are currently provided by our shadow master mechanism. Updates to the master state are made persistent by appending to a write-ahead log. Therefore we could adapt a primary-copy scheme like the one in Harp [7] to provide high availability with stronger consistency guarantees than our current scheme.

We are addressing a problem similar to Lustre [8] in terms of delivering aggregate performance to a large number of clients. However, we have simplified the problem significantly by focusing on the needs of our applications rather than building a POSIX-compliant file system. Additionally, GFS assumes large number of unreliable components and so fault tolerance is central to our design.

GFS most closely resembles the NASD architecture [4]. While the NASD architecture is based on network-attached disk drives, GFS uses commodity machines as chunkservers, as done in the NASD prototype. Unlike the NASD work, our chunkservers use lazily allocated fixed-size chunks rather than variable-length objects. Additionally, GFS implements features such as rebalancing, replication, and recovery that are required in a production environment.

Unlike Minnesota's GFS and NASD, we do not seek to alter the model of the storage device. We focus on addressing day-to-day data processing needs for complicated distributed systems with existing commodity components.

The producer-consumer queues enabled by atomic record appends address a similar problem as the distributed queues in River [2]. While River uses memory-based queues distributed across machines and careful data flow control, GFS uses a persistent file that can be appended to concurrently by many producers. The River model supports m-to-n distributed queues but lacks the fault tolerance that comes with persistent storage, while GFS only supports m-to-1 queues efficiently. Multiple consumers can read the same file, but they must coordinate to partition the incoming load.

9. CONCLUSIONS

The Google File System demonstrates the qualities essential for supporting large-scale data processing workloads on commodity hardware. While some design decisions are specific to our unique setting, many may apply to data processing tasks of a similar magnitude and cost consciousness.

We started by reexamining traditional file system assumptions in light of our current and anticipated application workloads and technological environment. Our observations have led to radically different points in the design space. We treat component failures as the norm rather than the exception, optimize for huge files that are mostly appended to (perhaps concurrently) and then read (usually sequentially), and both extend and relax the standard file system interface to improve the overall system.

Our system provides fault tolerance by constant monitoring, replicating crucial data, and fast and automatic recovery. Chunk replication allows us to tolerate chunkserver

failures. The frequency of these failures motivated a novel online repair mechanism that regularly and transparently repairs the damage and compensates for lost replicas as soon as possible. Additionally, we use checksumming to detect data corruption at the disk or IDE subsystem level, which becomes all too common given the number of disks in the system.

Our design delivers high aggregate throughput to many concurrent readers and writers performing a variety of tasks. We achieve this by separating file system control, which passes through the master, from data transfer, which passes directly between chunkservers and clients. Master involvement in common operations is minimized by a large chunk size and by chunk leases, which delegates authority to primary replicas in data mutations. This makes possible a simple, centralized master that does not become a bottleneck. We believe that improvements in our networking stack will lift the current limitation on the write throughput seen by an individual client.

GFS has successfully met our storage needs and is widely used within Google as the storage platform for research and development as well as production data processing. It is an important tool that enables us to continue to innovate and attack problems on the scale of the entire web.

ACKNOWLEDGMENTS

We wish to thank the following people for their contributions to the system or the paper. Brain Bershad (our shepherd) and the anonymous reviewers gave us valuable comments and suggestions. Anurag Acharya, Jeff Dean, and David desJardins contributed to the early design. Fay Chang worked on comparison of replicas across chunkservers. Guy Edjlali worked on storage quota. Markus Gutschke worked on a testing framework and security enhancements. David Kramer worked on performance enhancements. Fay Chang, Urs Hoelzle, Max Ibel, Sharon Perl, Rob Pike, and Debby Wallach commented on earlier drafts of the paper. Many of our colleagues at Google bravely trusted their data to a new file system and gave us useful feedback. Yoshka helped with early testing.

REFERENCES

- [1] Thomas Anderson, Michael Dahlin, Jeanna Neefe, David Patterson, Drew Roselli, and Randolph Wang. Serverless network file systems. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, pages 109–126, Copper Mountain Resort, Colorado, December 1995.
- [2] Remzi H. Arpaci-Dusseau, Eric Anderson, Noah Treuhaft, David E. Culler, Joseph M. Hellerstein, David Patterson, and Kathy Yelick. Cluster I/O with River: Making the fast case common. In *Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems (IOPADS '99)*, pages 10–22, Atlanta, Georgia, May 1999.
- [3] Luis-Felipe Cabrera and Darrell D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computer Systems*, 4(4):405–436, 1991.
- [4] Garth A. Gibson, David F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th Architectural Support for Programming Languages and Operating Systems*, pages 92–103, San Jose, California, October 1998.
- [5] John Howard, Michael Kazar, Sherri Menees, David Nichols, Mahadev Satyanarayanan, Robert Sidebotham, and Michael West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.
- [6] InterMezzo. <http://www.inter-mezzo.org>, 2003.
- [7] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, Liuba Shrira, and Michael Williams. Replication in the Harp file system. In *13th Symposium on Operating System Principles*, pages 226–238, Pacific Grove, CA, October 1991.
- [8] Lustre. <http://www.lustre.org>, 2003.
- [9] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109–116, Chicago, Illinois, September 1988.
- [10] Frank Schmuck and Roger Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the First USENIX Conference on File and Storage Technologies*, pages 231–244, Monterey, California, January 2002.
- [11] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O’Keefe. The Gobal File System. In *Proceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, College Park, Maryland, September 1996.
- [12] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, pages 224–237, Saint-Malo, France, October 1997.