

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re <i>Ex Parte</i> Reexamination of:	)	
	)	
U.S. Patent No. 7,904,686	)	
	)	
Issued: March 8, 2011	)	Group Art Unit: To Be Assigned
	)	
Named Inventor: Ranganath G. Iyengar,	)	Examiner: To Be Assigned
	)	
Control Number: To Be Assigned	)	
	)	
Filed: May 13, 2008	)	
	)	
Title: DATA SECURITY FOR USE WITH A	)	<u>VIA EFS-Web</u>
FILE SYSTEM	)	
	)	

Mail Stop *Ex Parte* Reexam

Attn: Central Reexamination Unit
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

REQUEST FOR *EX PARTE* REEXAMINATION

Requester NetApp, Inc. respectfully requests *ex parte* reexamination under 35 U.S.C. § 302 and 37 C.F.R. § 1.510 of U.S. Patent No. 7,904,686, purportedly assigned to Patent Owner Valtrus Innovations Limited, per the assignments at reel/frame: 056157/0492, 055269/0001, and 037079/0001. Prior art not cited or applied during prosecution renders issued claims 1-18 unpatentable under 35 U.S.C. §§ 102 and/or 103.

As this Request demonstrates, prior art references, including at least *Colgrove*, *Merrell*, *Lai*, *Card*, and *Blumenau* raise substantial new questions of patentability (“SNQs”) with respect to claims 1-18. Indeed, this prior art renders these claims unpatentable under 35 U.S.C. §§ 102

and/or 103. NetApp thus respectfully requests that the Office grant this Request for *ex parte* reexamination of the '686 patent and cancel claims 1-18.

In accordance with 37 C.F.R. § 1.510(b), this Request includes the following:

- (1) A statement pointing out each substantial new question of patentability based on prior patents and printed publications.
- (2) An identification of every claim for which reexamination is requested, and a detailed explanation of the pertinency and manner of applying the cited prior art to every claim for which reexamination is requested. For each statement of the patent owner and accompanying information submitted pursuant to [37 C.F.R.] § 1.501(a)(2) which is relied upon in the detailed explanation, the request must explain how that statement is being used to determine the proper meaning of a patent claim in connection with the prior art applied to that claim and how each relevant claim is being interpreted. If appropriate, the party requesting reexamination may also point out how claims distinguish over cited prior art.
- (3) A copy of every patent or printed publication relied upon or referred to in paragraph (b) (1) and (2) of this section accompanied by an English language translation of all the necessary and pertinent parts of any non-English language patent or printed publication.
- (4) A copy of the entire patent including the front face, drawings, and specification/claims (in double column format) for which reexamination is requested, and a copy of any disclaimer, certificate of correction, or reexamination certificate issued in the patent. All copies must have each page plainly written on only one side of a sheet of paper.
- (5) A certification that a copy of the request filed by a person other than the patent owner has been served in its entirety on the patent owner at the address as provided for in [37 C.F.R.] § 1.33(c). The name and address of the party served must be indicated. If service was not possible, a duplicate copy must be supplied to the Office.
- (6) A certification by the third party requester that the statutory estoppel provisions of 35 U.S.C. [§] 315(e)(1) or 35 U.S.C. [§] 325(e)(1) do not prohibit the requester from filing the *ex parte* reexamination request.

Attachments

- (1) Certificate of Service on Patent Owner.
- (2) Information Disclosure Statement Form PTO/SB/08.

Exhibits

- Ex. A: U.S. Patent No. 7,904,686 to Ranganath G. Iyengar (“the ’686 patent”)
- Ex. B: Patent Prosecution History of U.S. Patent No. 7,904,686
- Ex. C: Declaration of Dr. Paul C. Clark.
- Ex. D: Declaration of June Ann Munford.
- Ex. E: File History of U.S. Patent No. 6,377,958
- Ex. F: File History of U.S. Patent No. 8,285,757
- Ex. G: Curriculum Vitae of Dr. Paul C. Clark

Prior Art Exhibits

- PA-1: U.S. Patent No. 7,320,008 to John A. Colgrove (“*Colgrove*”)
- PA-2: U.S. Patent No. 6,907,457 to Alan Ray Merrell et al. (“*Merrell*”)
- PA-3: U.S. Patent Pub. No. 2003/0182564 to Jing-Shiun Lai et al. (“*Lai*”)
- PA-4: Rémy Card et al., “Design and Implementation of the Second Extended Filesystem,” Proceedings of the First Dutch International Symposium on Linux (“*Card*”)
- PA-5: U.S. Patent Application Publication No. 2004/0091114 to Ernst B. Carter et al. (“*Carter*”)
- PA-6: Jim Mauro et al., “Solaris Internals: Core Kernel Components,” Sun Microsystems Press (“*Mauro*”)
- PA-7: U.S. Patent No. 6,282,602 to Steven M. Blumenau et al. (“*Blumenau*”)

Litigation Document Exhibits

- L-1: Plaintiff’s Complaint for Patent Infringement in *Valtrus Innovations Ltd. et al. v. The Home Depot, Inc. et al.*, No. 2:25-cv-00081, Dkt. 1 (E.D. Tex. Jan. 27, 2025)
- L-2: Plaintiff’s Complaint for Patent Infringement in *Valtrus Innovations Ltd. et al. v. NetApp, Inc.*, No. 2:25-cv-00517, Dkt. 1 (E.D. Tex. May 9, 2025)
- L-3: Plaintiff’s Complaint for Patent Infringement in *Valtrus Innovations Ltd. et al. v. FedEx Corp. et al.*, No. 2:25-cv-00516, Dkt. 1 (E.D. Tex. May 9, 2025)
- L-4: Consolidation Order in *Valtrus Innovations Ltd. et al. v. NetApp, Inc.*, No. 2:25-cv-00517, Dkt. 11 (E.D. Tex. July 27, 2025)
- L-5: Declaratory Judgment Complaint in *NetApp, Inc. v. Valtrus Innovations Ltd. et al.*, No. 1:25-cv-06138, Dkt. 1 (S.D.N.Y. July 25, 2025)

Inter Partes Review Document Exhibits

- IPR-1: Petition for *Inter Partes Review*, *FedEx Corp. et al. v. Valtrus Innovations Ltd.*, IPR2025-01491, Paper 2 (PTAB Aug. 2, 2025)

TABLE OF CONTENTS

	<u>Page</u>
I. RELATED PROCEEDINGS.....	1
A. Certification Regarding Statutory Estoppel.....	1
B. Other Proceedings.....	1
C. Request for Expedited Reexamination.....	1
II. IDENTIFICATION OF CLAIMS THAT SHOULD BE REEXAMINED	2
III. BACKGROUND OF FILE SYSTEMS.....	2
IV. OVERVIEW OF THE '686 PATENT AND ITS PROSECUTION HISTORY	4
A. '686 Patent Specification	4
B. Claims 1-18 of the '686 Patent	5
C. '686 Patent Prosecution History	8
D. Level of Ordinary Skill in the Art.....	9
E. Claim Construction for the '686 Patent	9
1. “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” (claim 17).....	9
2. “means for obtaining mapped data block numbers after applying the mapping function” (claim 17).....	10
V. IDENTIFICATION OF PRIOR ART AND SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY	11
A. Identification of Prior Art	11
B. Substantial New Questions of Patentability.....	11
1. <i>Colgrove</i> Raises an SNQ with Respect to Claims 1, 5-9, and 13-18.....	12
2. The Combined Teachings of <i>Colgrove</i> and <i>Lai</i> Raise an SNQ with Respect to Claims 1-3, 5-11, and 13-16.....	17
3. The Combined Teachings of <i>Colgrove</i> and <i>Merrell</i> Raise an SNQ with Respect to Claims 1-3, 5-11, and 13-16.....	19

4.	The Combined Teachings of <i>Lai</i> and <i>Card</i> Raise an SNQ with Respect to Claims 1-3, 5-11, and 13-18.....	22
5.	The Combined Teachings of <i>Lai</i> , <i>Card</i> , and <i>Blumenau</i> Raise an SNQ with Respect to Claims 4 and 12.....	27
VI.	PROPOSED REJECTIONS OF THE '686 PATENT	28
VII.	DETAILED EXPLANATION OF THE PROPOSED REJECTIONS.....	28
A.	<i>Colgrove</i> Anticipates (SNQ 1) or Renders Obvious (SNQ 2) Claims 1, 5-9, and 13-18	28
1.	Overview of <i>Colgrove</i>	29
2.	Independent Claim 1	32
a.	1[pre]: “A method of providing data security for use with a file system, the method comprising:”	32
b.	1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	32
c.	1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	35
3.	Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”	36
4.	Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”	37
5.	Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	38
6.	Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”	38
7.	Independent Claim 9	39
a.	9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”	39

b.	9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”	40
c.	9[b]: “obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device.”	40
8.	Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”	41
9.	Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”	41
10.	Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	41
11.	Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”	41
12.	Independent Claim 17	41
a.	17[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”	41
b.	17[a]: “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	42
c.	17[b]: “means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	42
13.	Independent Claim 18	42
a.	18[a]: “Article of manufacture, comprising: a machine-readable medium having stored thereon instructions to:”	42
b.	18[a]: “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	44

c.	18[b]: “obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	44
B.	SNQ 3: <i>Colgrove</i> in view of <i>Lai</i> Renders Obvious Claims 1-3, 5-11, and 13-16	44
1.	Overview of <i>Lai</i>	44
2.	Reasons to Combine <i>Colgrove</i> and <i>Lai</i>	46
3.	Independent Claim 1	51
a.	1[pre]: “A method of providing data security for use with a file system, the method comprising:”	51
b.	1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	51
c.	1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	52
4.	Claim 2: “The method of claim 1, further comprising: storing a mapping of the data block numbers and the mapped data block numbers.”	52
5.	Claim 3	53
a.	3[pre]: “The method of claim 1, wherein a read operation for the file comprises:”	53
b.	3[a]: “obtaining the data block numbers of the file;	54
c.	3[b]: “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”	56
d.	3[c]: “reading the data in the mapped data block numbers in the storage device.”	56
6.	Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”	57

7.	Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”	57
8.	Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	58
9.	Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”	58
10.	Independent Claim 9	58
	a. 9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”	58
	b. 9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”	58
	a. 9[b]: “obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device.”	59
11.	Claim 10: “The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.”	60
12.	Claim 11	60
	a. 11[pre]: “The apparatus of claim 9, wherein for a read operation for the file,.....	60
	b. 11[a]: “the block distribution engine obtains the data block numbers of the file from a file system, and”	61
	c. 11[b]: “obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”	61
	d. 11[c]: “wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.”	62
13.	Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”	62

14.	Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”	63
15.	Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	63
16.	Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”	63
C.	SNQ 4: <i>Colgrove</i> in view of <i>Merrell</i> Renders Obvious Claims 1-3, 5-11, and 13-16	63
1.	Overview of <i>Merrell</i>	63
2.	Reasons to Combine <i>Colgrove</i> and <i>Merrell</i>	65
3.	Independent Claim 1	69
a.	1[pre]: “A method of providing data security for use with a file system, the method comprising:”	69
b.	1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	70
c.	1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	70
4.	Claim 2: “The method of claim 1, further comprising: storing a mapping of the data block numbers and the mapped data block numbers.”	71
5.	Claim 3.....	71
a.	3[pre]: “The method of claim 1, wherein a read operation for the file comprises:”	71
b.	3[a]: “obtaining the data block numbers of the file;	74
c.	3[b]: “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”	76

d.	3[c]: “reading the data in the mapped data block numbers in the storage device.”	76
6.	Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”	77
7.	Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”	77
8.	Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	78
9.	Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”	78
10.	Independent Claim 9	78
a.	9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”	78
b.	9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”	78
c.	9[b]: “obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device.”	79
11.	Claim 10: “The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.”	80
12.	Claim 11	80
a.	11[pre]: “The apparatus of claim 9, wherein for a read operation for the file,.....	80
b.	11[a]: “the block distribution engine obtains the data block numbers of the file from a file system, and”	81
c.	11[b]: “obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”	81

d.	11[c]: “wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.”	82
13.	Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”	82
14.	Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”	83
15.	Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	83
16.	Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”	83
D.	SNQ 5: <i>Lai</i> in View of <i>Card</i> Renders Obvious Claims 1-3, 5-11, and 13-18	83
1.	Overview of <i>Card</i>	83
2.	Reasons to Combine <i>Lai</i> and <i>Card</i>	85
3.	Independent Claim 1:	88
a.	1[pre]: “A method of providing data security for use with a file system, the method comprising:”	88
b.	1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	89
c.	1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	91
4.	Claim 2: “The method of claim 1, further comprising: storing a mapping of the data block numbers and the mapped data block numbers.”	92
5.	Claim 3	93

a.	3[pre]: “The method of claim 1, wherein a read operation for the file comprises:”	93
b.	3[a]: “obtaining the data block numbers of the file;	94
c.	3[b]: “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”	94
d.	3[c]: “reading the data in the mapped data block numbers in the storage device.”	94
6.	Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”	95
7.	Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”	95
8.	Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	97
9.	Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”	97
10.	Independent Claim 9	98
a.	9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”	98
b.	9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”	99
c.	9[b]: “obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device.”	99
11.	Claim 10: “The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.”	99
12.	Claim 11	100
a.	11[pre]: “The apparatus of claim 9, wherein for a read operation for the file, ...	100

b.	11[a]: “the block distribution engine obtains the data block numbers of the file from a file system, and”	100
c.	11[b]: “obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”	101
d.	11[c]: “wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.”	101
13.	Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”	103
14.	Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”	103
15.	Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”	103
16.	Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”	103
17.	Independent Claim 17:	103
a.	17[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”	103
b.	17[a]: “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”	104
c.	17[b]: “means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	104
18.	Independent Claim 18:	104
a.	18[pre]: “Article of manufacture, comprising: a machine-readable medium having stored thereon instructions to:”	104
b.	18[a]: “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers	

	are contained in an index node associated with said file; and”	105
c.	18[b]: “obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”	106
E.	SNQ 6: <i>Lai</i> in view of <i>Card</i> and <i>Blumenau</i> Renders Obvious Claims 4 and 12.....	106
1.	Overview of <i>Blumenau</i>	106
2.	Reasons to Combine <i>Lai</i> , <i>Card</i> , and <i>Blumenau</i>	111
3.	Claim 4:.....	116
a.	4[pre]: “The method of claim 1, wherein a write operation for the file comprises:”	116
b.	4[a]: “allocating data block numbers of free data blocks in the storage device;”	117
c.	4[b]: “linking the data block numbers to a file name of the file;”	117
d.	4[c]: “applying the mapping function to the data block numbers in order to obtain the mapped data block numbers; and”	119
e.	4[d]: “writing the data of the file into the mapped data block numbers in the storage device.”	120
4.	Claim 12.....	120
a.	12[pre]: “The apparatus of claim 9, wherein for a write operation for the file,”	120
b.	12[a]: “the block distribution engine is configured to receive allocated data block numbers of free data blocks in the storage device from a file system,”	120
c.	12[b]: “link the data block numbers to a file name of the file, and”	121
d.	12[c]: “apply the mapping function to the data block numbers in order to obtain the mapped data block numbers; and”	121

- e. 12[d]: “wherein a block device driver is configured to write the data of the file into the mapped data block numbers in the storage device.” 122

VIII. CONCLUSION.....	124
-----------------------	-----

I. RELATED PROCEEDINGS

A. Certification Regarding Statutory Estoppel

As required by 37 C.F.R. § 1.510(b)(6), NetApp certifies that the statutory estoppel provisions of 35 U.S.C. § 315(e)(1) and 35 U.S.C. § 325(e)(1) do not prohibit NetApp from filing the instant request for *ex parte* reexamination.

B. Other Proceedings

Valtrus asserted the '686 patent against (1) Home Depot, Inc. and Home Depot U.S.A., Inc. in a lawsuit captioned *Valtrus Innovations Ltd. et al. v. The Home Depot, Inc. et al.*, No. 2:25-cv-00081 (E.D. Tex. Jan. 27, 2025); (2) NetApp in a lawsuit captioned *Valtrus Innovations Ltd. et al. v. NetApp, Inc.*, No. 2:25-cv-00517 (E.D. Tex. May 9, 2025); and (3) FedEx Corporation and FedEx Office and Print Services, Inc. in a lawsuit captioned *Valtrus Innovations Ltd. et al. v. FedEx Corporation et al.*, No. 2:25-cv-00516 (E.D. Tex. May 9, 2025). Ex. L-1; Ex. L-2; Ex. L-3. On June 27, 2025, the cases against NetApp and the FedEx defendants were consolidated. Ex. L-4.

On July 25, 2025, Requester filed a declaratory judgment complaint involving the '686 patent in *NetApp, Inc. v. Valtrus Innovations Ltd. et al.*, 1:25-cv-06138 (S.D.N.Y. July 25, 2025). Ex. L-5.

On August 29, 2025, FedEx Corporation and FedEx Office and Print Services, Inc. filed a petition for *inter partes* review. Ex. IPR-1.

C. Request for Expedited Reexamination

Due to the ongoing nature of the above-identified lawsuits, Requester respectfully urges that, pursuant to 35 U.S.C. § 305, this Request be granted and reexamination be conducted not only with “special dispatch,” but also with “priority over all other cases” in accordance with M.P.E.P. § 2261.

II. IDENTIFICATION OF CLAIMS THAT SHOULD BE REEXAMINED

Under 37 C.F.R. § 1.510(b)(2), NetApp respectfully requests *ex parte* reexamination of claims 1-18 of the '686 patent and the issuance of a reexamination certificate cancelling these claims.

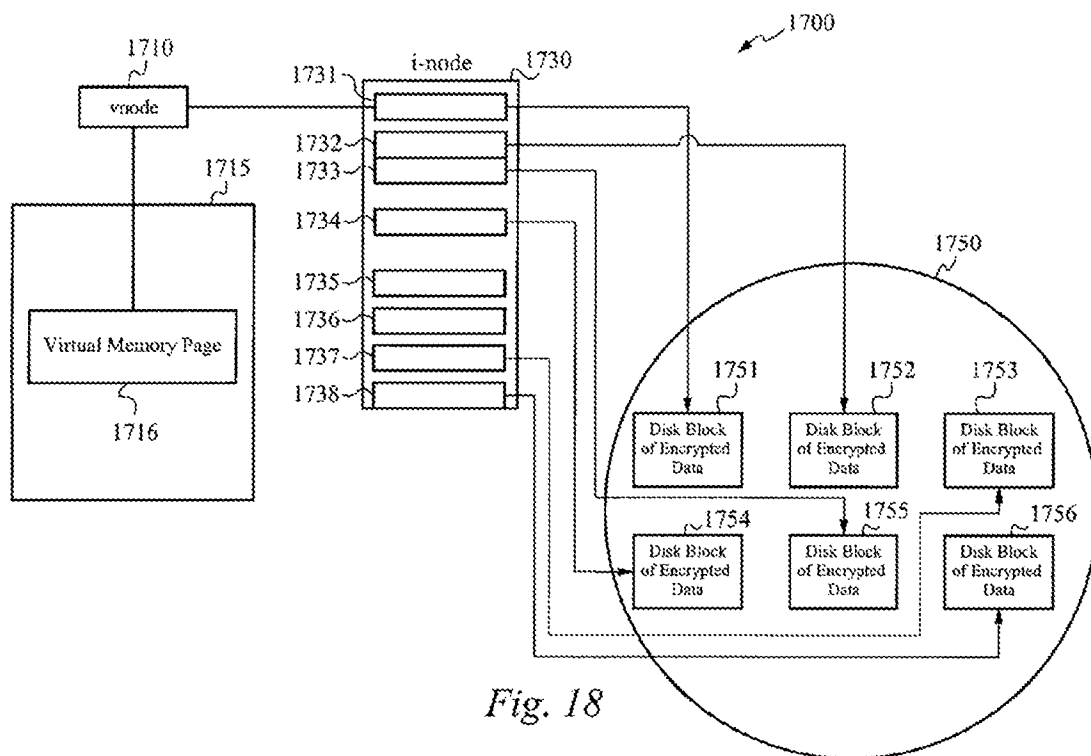
III. BACKGROUND OF FILE SYSTEMS

File systems have existed since before the 1980s, and as the '686 patent admits, Unix-based file systems have long used index nodes, also called inodes, to store information about directories and other files stored in the file system. Ex. C, ¶ 49; Ex. A, 1:30-33, 2:15-28; Ex. PA-1, *Colgrove*, 2:31-3:4; Ex. PA-4, *Card*, 2 (“The addresses of data blocks allocated to a file are stored in its inode.”). In particular, file system software maintains metadata about the files stored in the file system and uses the metadata, including the inodes, “to track where each portion of each file is stored.” Ex. PA-1, *Colgrove*, 2:52-56.

Each file in the file system is represented by a corresponding inode data structure. Ex. PA-4, *Card*, 2; Ex. PA-1, *Colgrove*, 2:57-63. Historically, the inode contained information about the corresponding file, including “[t]he addresses of data blocks allocated to [the] file.” Ex. C, ¶ 50; Ex. PA-4, *Card*, 2; Ex. PA-1, *Colgrove*, 2:57-3:4. “When a user requests an I/O operation on the file, the kernel code [within the operating system] converts the current offset to a block number, uses this number as an index in the block addresses table and reads or writes the physical block.” Ex. PA-4, *Card*, 2; Ex. PA-2, *Merrell*, 4:10-13 (disclosing that a client may request to “read file ‘a,’ offset 10 for 100 bytes”). To read the requested portion of the file, the file system software obtains the inode corresponding to file, uses the offset and length to determine which logical block numbers (block indexes) within the inode are requested, obtains the physical addresses contained within the inode corresponding to the logical block numbers,

and reads the data blocks associated with the file and stored at those physical addresses. Ex. C, ¶ 50; Ex. PA-2, *Merrell*, 6:51-7:27.

Inodes, however, presented a potential data security vulnerability that was known before the '686 patent. Ex. C, ¶ 51. Because each inode corresponds to a file and “contains the locations of the data blocks that contain the data associated with the file,” an unauthorized user that gains access to the inode could “recover the data associated with the file.” Ex. PA-5, *Carter*, [0218]. One known solution to this vulnerability was to encrypt the data blocks of the file on the disk, as illustrated below. Ex. C, ¶ 51. In this system, the inode 1730 contains the addresses 1731-1738 of the encrypted data blocks 1751-1756 of the file. Ex. PA-5, *Carter*, [0222]-[0223]. The file system uses the inode to access the encrypted data blocks, which “can then be decrypted using encryption keys.” Ex. PA-5, *Carter*, [0223].



Ex. PA-5, *Carter*, Fig. 18.

Another known solution was to “encrypt (and decrypt when necessary) one or more elements or levels of data system metadata,” such as the inodes or portions of the inode (e.g., the addresses of the data blocks), “while not encrypting, and thus not decrypting, the data stored in the data system itself.” Ex. C, ¶ 52; Ex. PA-1, *Colgrove*, 4:64-5:3, 5:46-59, 7:49-51. Other known solutions existed as well, including, for example, implementing multiple “mapping layers that map from the logical objects operated upon by the application programs to the particular physical locations in the storage system where the blocks that make up the logical object are stored.” Ex. PA-7, 1:37-41, 1:53-65, 2:8-17.

Each of these concepts was well understood in the art before the application leading to the '686 patent was filed. Ex. C, ¶ 53.

IV. OVERVIEW OF THE '686 PATENT AND ITS PROSECUTION HISTORY

A. '686 Patent Specification

The '686 patent purportedly provides security for use in a file system by applying a mapping function to data block numbers contained in an inode to obtain mapped data block numbers that are addresses of a file's data in a storage device. Ex. A, Abstract, 2:53-3:8. Figure 2, reproduced below, shows an exemplary mapping where mapping function $f(x)$, which can be “any function” (e.g., “ $f(x)=5x$ ”), is applied such that data block numbers X1-X4 in inode 210 are mapped to the mapped data block numbers $f(X1)$ - $f(X4)$. *Id.*, 3:50-4:3, 4:32-34. The '686 patent discloses that the file's data in this embodiment is stored in data blocks 215 identified by the mapped data block numbers $f(X1)$ - $f(X4)$, rather than in data blocks 212 identified by the data block numbers X1-X4. *Id.*, 3:51-4:3, FIG. 2. According to the '686 patent, “[a]n unauthorized user will not be able to obtain the data for a file 205 by simply accessing the inode 210 of the file 205 and copying the data blocks 212, because the actual data of the file 205 are stored in the mapped data blocks 215.” *Id.*, 4:34-38.

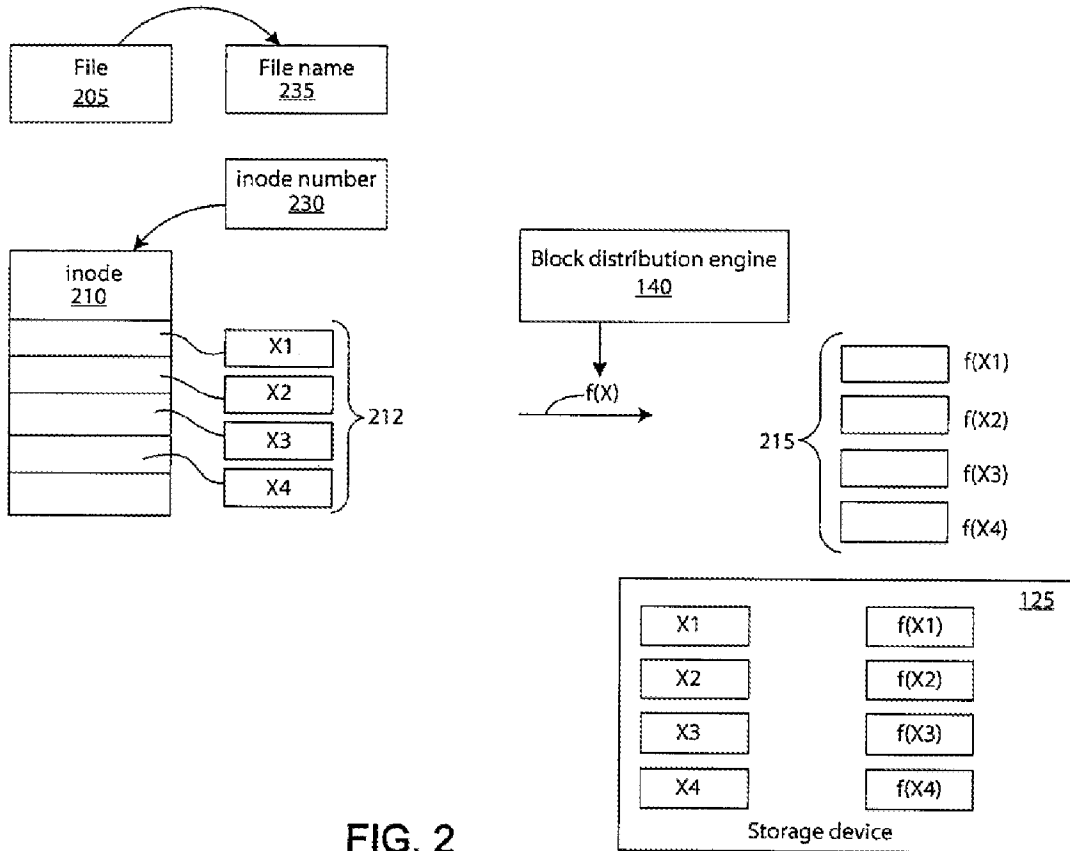


FIG. 2

Id., FIG. 2.

B. Claims 1-18 of the '686 Patent

This Request concerns independent claims 1, 9, 17, and 18; dependent claims 2-8 (which depend from claim 1); and dependent claims 10-16 (which depend from claim 9). These claims are reproduced below:

1. A method of providing data security for use with a file system, the method comprising:

applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and

obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.
2. The method of claim 1, further comprising:

storing a mapping of the data block numbers and the mapped data block numbers.

3. The method of claim 1, wherein a read operation for the file comprises:
 - obtaining the data block numbers of the file;
 - obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and
 - reading the data in the mapped data block numbers in the storage device.
4. The method of claim 1, wherein a write operation for the file comprises:
 - allocating data block numbers of free data blocks in the storage device;
 - linking the data block numbers to a file name of the file;
 - applying the mapping function to the data block numbers in order to obtain the mapped data block numbers; and
 - writing the data of the file into the mapped data block numbers in the storage device.
5. The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.
6. The method of claim 1, further comprising:
 - selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.
7. The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.
8. The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.
9. An apparatus for providing data security for use with a file system, the apparatus comprising:
 - a block distribution engine configured to apply a mapping function to data block numbers that are associated with a

file, wherein the data block numbers are contained in an index node associated with said file, and obtain mapped data block numbers after applying the mapping function;

wherein the mapped data block numbers are addresses of data of the file in a storage device.

10. The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.
11. The apparatus of claim 9, wherein for a read operation for the file, the block distribution engine obtains the data block numbers of the file from a file system, and obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.
12. The apparatus of claim 9, wherein for a write operation for the file, the block distribution engine is configured to receive allocated data block numbers of free data blocks in the storage device from a file system, link the data block numbers to a file name of the file, and apply the mapping function to the data block numbers in order to obtain the mapped data block numbers; and wherein a block device driver is configured to write the data of the file into the mapped data block numbers in the storage device.
13. The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.
14. The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.
15. The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.
16. The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.
17. An apparatus for providing data security for use with a file system, the apparatus comprising:

means for applying a mapping function to data block numbers that are associated with a file, wherein the data block

numbers are contained in an index node associated with said file; and

means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.

18. Article of manufacture, comprising:

a machine-readable medium having stored thereon instructions to:

apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and

obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.

Id. at 6:12-8:22.

C. '686 Patent Prosecution History

During prosecution, the Examiner rejected all pending claims as obvious over U.S. Patent Pub. No. 2008/0034264 (“Kaabouch”) “in view of well-known features of which Official Notice [was] taken.” Ex. B, 41. The Examiner asserted it was “known and commonly practiced in the pertinent art that memory systems such as that of Kaabouch are configured for storing data files having associated data block numbers.” *Id.*

Applicant argued the Examiner’s use of official notice without evidentiary support was improper, amended all independent claims to add “wherein the data block numbers are contained in an index node associated with said file,” and argued that Kaabouch lacked this limitation. *Id.*, 75-84. A Notice of Allowance followed. *Id.*, 90-95.

As discussed below, the features relied on to gain allowance are taught by the references cited in this Request (Section VII), none of which were considered by the Examiner.

D. Level of Ordinary Skill in the Art

A person of ordinary skill in the art (“POSITA”) would have had at least a bachelor’s degree in electrical engineering, computer science, or a similar field and two to three years of experience designing or analyzing computer storage systems. Ex. C, ¶ 41. Additional relevant work experience can compensate for less education, and vice versa. *Id.*

E. Claim Construction for the ’686 Patent

With the exception of the following two terms, no other term in the ’686 patent requires a construction. Requester reserves the right to argue in an appropriate forum that certain limitations in the challenged claims are indefinite, lack written description support, or are invalid for other reasons not at issue here.

1. “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” (claim 17)

This term recites the word “means” and should be presumptively construed as means-plus-function under pre-AIA 35 U.S.C. § 112, ¶ 6, with the following function and corresponding structure:

- **Function:** “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file”;
- **Structure:** a block distribution engine or equivalents. *See* Ex. A, 3:57-4:13, 4:49-53, 5:30-35.

This structure is consistent with the claims and specification. Ex. C, ¶¶ 44-45. Claim 9 recites “a block distribution engine” that performs this function. Ex. A, 6:55-58. Accordingly, the

'686 patent “clearly associate[s]” a “block distribution engine” with the recited function. *See Cardiac Pacemakers, Inc. v. St. Jude Med., Inc.*, 296 F.3d 1106, 1113 (Fed. Cir. 2002). And the '686 patent discloses that “inode 210 of file 205 points to four (4) data block numbers X1-X4,” and that “block distribution engine 140 maps the data block numbers X1-X4 to different data block numbers $f(X1)$ - $f(X4)$, respectively, by applying the equation $f(x)$ to the data block numbers $x=\{X1, X2, X3, X4\}$,” where $f(x)$ is any “mapping function” applied to the data block numbers. Ex. A, 3:50-64, 4:25-29.

2. “means for obtaining mapped data block numbers after applying the mapping function” (claim 17)

This term recites the word “means” and should be presumptively construed as means-plus-function under pre-AIA 35 U.S.C. § 112, ¶ 6, with the following function and corresponding structure:

- Function: “obtaining mapped data block numbers after applying the mapping function”;
- Structure: a block distribution engine, I/O subsystem of the operating system, block device driver, or equivalents, as well as hardware executing the same. *See* Ex. A, 3:28-37, 4:49-62, 4:67-5:3, 5:36-42.

This structure is consistent with the claims and specification. Ex. C, ¶¶ 46-47. Claim 9 recites “a block distribution engine” that performs the same function. Ex. A, 6:55-60.

Accordingly, the '686 patent “clearly associate[s]” a “block distribution engine” with the recited function. *See Cardiac Pacemakers*, 296 F.3d at 1113. Moreover, in the specification, the block distribution engine 140 obtains the mapped data block numbers after applying the mapping function because it “converts (maps) the data block numbers ... to the mapped data block numbers.” Ex. A, 2:62-3:1, 5:30-35. Block distribution engine 140 then returns the mapped data

block numbers to the I/O subsystem, which passes them to the block device driver. *Id.*, 3:34-37, 5:36-39.

V. IDENTIFICATION OF PRIOR ART AND SUBSTANTIAL NEW QUESTIONS OF PATENTABILITY

A. Identification of Prior Art

The earliest possible priority date for claims 1-18 of the '686 patent is July 31, 2007.

Ex. A, Cover Page, (30). Reexamination is requested in view of the following prior art references:

- PA-1: U.S. Patent No. 7,320,008 to John A. Colgrove, filed Dec. 20, 2004, and issued Jan. 15, 2008 ("*Colgrove*").
- PA-2: U.S. Patent No. 6,907,457 to Alan Ray Merrell et al., filed Jan. 25, 2001, and issued June 14, 2005 ("*Merrell*").
- PA-3: U.S. Patent Pub. No. 2003/0182564 to Jing-Shiun Lai et al., filed Mar. 19, 2002, and published Sep. 25, 2003 ("*Lai*").
- PA-4: Rémy Card et al., "Design and Implementation of the Second Extended Filesystem," Proceedings of the First Dutch International Symposium on Linux, published no later than January 18, 2000 ("*Card*").
- PA-7: U.S. Patent No. 6,282,602 to Steven M. Blumenau et al., filed June 30, 1998, and issued Aug. 28, 2001 ("*Blumenau*").

These prior art documents are listed on a form PTO/SB/08 filed concurrently herewith. Copies of all the listed references are attached.

B. Substantial New Questions of Patentability

For an SNQ to be present, "it is only necessary that: (A) an item of information raises a[n] SNQ regarding at least one claim, i.e., the teaching of the item of information is such that a reasonable examiner would consider the teaching to be important in deciding whether or not the claim is patentable; and (B) the same question of patentability . . . has not been decided by the Office in an earlier concluded examination or review of the patent, decided in a final holding of invalidity (after all appeals) by a federal court in a decision on the merits involving the claim, or raised to or by the Office in a pending reexamination or supplemental examination of the patent."

M.P.E.P. § 2816.02(I); *see also In re Vivint, Inc.*, 14 F.4th 1342, 1349 (Fed. Cir. 2021). “For example, the same question of patentability may already have been decided if the Office has previously considered, in an earlier concluded examination or review of the patent, the same question of patentability based on the same prior art.” *See* M.P.E.P. § 2816.02(I) (citing *In re Recreative Techs. Corp.*, 83 F.3d 1394 (Fed. Cir. 1996)).

Each proposed rejection in this Request presents a distinct SNQ of patentability. First, a reasonable Examiner would find the teachings of the prior art submitted in this Request (as also discussed in detail below) important in deciding whether the claims are patentable. Second, the same question of patentability has not been decided by the Office in an earlier concluded review or examination of the ’686 patent. During prosecution, the Examiner did not consider any of the prior art relied upon in this Request. *See* Section IV.C. Third, the same question of patentability has not been decided in a final holding of invalidity by a federal court. Though the ’686 patent is currently involved in co-pending district court litigations, those litigations have not resolved any question of validity regarding the ’686 claims. *See supra* Section I.B. And finally, the same question of patentability has not been raised to or by the Office in a pending reexamination or supplemental examination. To NetApp’s best knowledge, there has been no previous Office proceedings, *inter partes* review or *ex parte* reexamination, involving the claims of the ’686 patent, much less any involving the specific combination of prior art that NetApp has raised below. While there is a pending petition for *inter partes* review of the ’686 patent, that petition relies on entirely different and distinct art not cited or relied on in this Request. *See* Ex. IPR-1.

1. Colgrove Raises an SNQ with Respect to Claims 1, 5-9, and 13-18

Colgrove’s teachings were not before the Office during prosecution (whether on an IDS or considered and addressed during prosecution). Ex. A, Cover Page (56). And *Colgrove* teaches the very claim features that the Examiner thought were missing from the prior art, *supra* Section

IV.C, anticipating or alone rendering obvious the independent claims of the '686 patent, *infra* Section VII.A. As such, *Colgrove* presents an SNQ for the patentability of claims 1, 5-9, and 13-18.

Independent claim 1 is directed to “[a] method of providing data security for use with a file system.” Ex. A, 6:12-31. The claimed method includes, in part, “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file.” *Id.*, 6:14-17. Independent claims 9 and 17 are directed to “[a]n apparatus for providing data security for use with a file system” and contain substantially similar limitations to claim 1. *Id.*, 6:53-62 (independent claim 9 reciting “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and obtain mapped data block numbers after applying the mapping function”), 8:3-12 (independent claim 17 reciting “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file”). Independent claim 18 is directed to an “[a]rticle of manufacture” and contains substantially similar limitations to claim 1. *Id.*, 8:13-22 (independent claim 18 reciting “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file”). Claims 5-8 depend from claim 1, and claims 13-15 depend from claim 9.

During prosecution, the Examiner initially rejected all claims as obvious over Kaabouch “in view of well-known features,” of which the Examiner took official notice. Ex. B, 41-42. In response, the Applicant amended the independent claims to require that “the data block numbers are contained in an index node associated with said file.” *Id.*, 76-79. The Applicant argued that

the cited prior art did not “suggest or teach and is silent in regards to” this feature. *Id.*, 81-84.

The Examiner then allowed the claims, stating that “none of the prior art of record teaches or fairly suggests the claimed” method, apparatus, or article of manufacture with the “specific combination” of claimed features. *Id.*, 94. But the concepts claimed in the ’686 patent were known before the earliest claimed priority date of the ’686 patent, as demonstrated by *Colgrove*. Indeed, *Colgrove* discloses the only feature that the Examiner believed was missing from the prior art.

For example, *Colgrove* discloses providing data security for use with a file system because it relates to “data protection mechanisms” for data systems,” where a “data system” refers to “any system that stores and manages access to data (e.g. files, records, etc.) typically stored or maintained on some type of logical or physical storage devices.” Ex. PA-1, *Colgrove*, 1:6-13. *Colgrove* discloses that a file system is one type of data system that uses metadata “to track where each portion of each file is stored.” *Id.*, 2:22-37, 2:41-56. The file system metadata includes inodes, which are data structures that include information about a particular file in the file system, including “the addresses of the file’s blocks on disk.” *Id.*, 2:57-3:4.

Colgrove discloses that “the metadata may be necessary to locate, access, reconstruct and/or understand the data itself” because a file system’s data is “stored on the storage device(s) in such a way that the data is essentially unintelligible without having the metadata to locate and reconstruct the data elements into files.” *Id.*, 5:7-16. Accordingly, instead of encrypting the data itself, *Colgrove* discloses a data protection mechanism that encrypts portions of metadata data structures in a data system, making it “difficult or impossible to reconstruct the stored data elements into data structures such as files.” *Id.*, 4:62-5:3, 5:17-21, 5:46-59.

In one embodiment, a file's data is "stored in blocks" and "accessible as files via the metadata for the file system 310." *Id.*, 7:31-34. *Colgrove* discloses that the portion of the inodes "that allows access to data (e.g. the block maps in inodes) may be encrypted." *Id.*, 5:56-59, 7:47-51. *Colgrove* discloses "the particular encryption algorithm used is not significant" and "a particular encryption algorithm may be selected to match the security and performance requirements of the file system 310." *Id.*, 7:51-56.

Data system software (e.g., file system software 300) includes a data protection mechanism (e.g., data protection mechanism 306), which encrypts and, when necessary, decrypts portions of the metadata, and data system functionality (e.g., file system functionality 302) that use unencrypted metadata to access and manage elements of the stored data, including performing operations like "storing, creating, deleting, copying, moving, and modifying elements" of the data. *Id.*, 4:64-5:3, 6:5-7, 6:17-22, 6:40-53, 7:40-45. In one embodiment, depicted below, the data system software "decrypt[s] the encrypted metadata from the data system to generate a decrypted copy of the metadata," "cache[s] the decrypted copy of the metadata in memory," and "access[es] the data in the data system in accordance with the decrypted copy of the metadata cached in memory." *Id.*, 11:51-58.

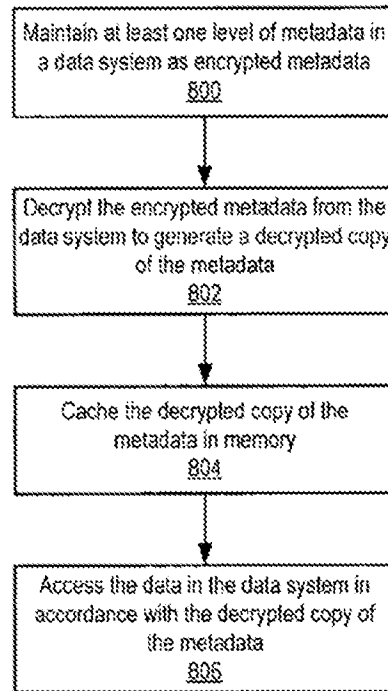


FIG. 9

Id., FIG. 9.

Decrypting the encrypted addresses of the file's blocks on disk teaches applying a mapping function to data block numbers that are associated with a file. And the encrypted address are contained in an inode associated with the file, which teaches that the data block numbers are contained in an index node associated with said file. This was the only limitation that the Examiner incorrectly believed was missing from the prior art.

Colgrove's teachings were not previously considered or addressed during the prosecution of the application resulting in the '686 patent. *Colgrove* was also not cited on any information disclosure statement or considered by the Examiner at any point during prosecution. *See* Ex. A, Cover Page (56). Additionally, each of the grounds raised in the pending IPR petition relies on different prior art with a different mapping. *See* Ex. IPR-1. For example, none of the grounds

raised in the IPR petition rely on decrypting the encrypted addresses of the file's blocks in an inode. *Id.* Because *Colgrove* teaches the claim features that the Examiner thought were missing from the prior art, and the Examiner did not consider *Colgrove*, it raises an SNQ for the patentability of claims 1, 5-9, and 13-18. Section VII.A, below, explains how each limitation of claims 1, 5-9, and 13-18 of the '686 patent were either disclosed or rendered obvious by *Colgrove*.

2. The Combined Teachings of *Colgrove* and *Lai* Raise an SNQ with Respect to Claims 1-3, 5-11, and 13-16

As stated in Section V.B.1, the teachings of *Colgrove* raise an SNQ with respect to claims 1, 5-9, and 13-18. Dependent claim 2 depends from claim 1 and further recites "storing a mapping of the data block numbers and the mapped data block numbers." Ex. A, 6:21-23. Dependent claim 10 is an apparatus claim that contains substantially similar limitations. *Id.*, 6:63-65. The '686 patent specification discloses an embodiment where "[t]he mapping 145 is a data structure mapping of the blocks in an inode of the file 205 to an [*sic*] algorithm-specific data blocks." *Id.*, 3:19-21.

Dependent claim 3 also depends from claim 1 and further recites "a read operation for the file." *Id.*, 6:24-31. The read operation comprises "obtaining the data block numbers of the file," "obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers," and "reading the data in the mapped data block numbers in the storage device." *Id.* Dependent claim 11 is an apparatus claim that contains substantially similar limitations. *Id.*, 6:66-7:5.

The combination of *Colgrove* and *Lai* raises substantial new questions of patentability regarding 1-3, 5-11, and 13-16. *Lai* contains disclosures related to a "method for protecting digital information" using a data protection system that maps default addresses into re-mapped

addresses within a storage device's protected zone. Ex. PA-3, *Lai*, Abstract, [0002], [0031]-[0046]. Figures 4 and 5, below, depict sample re-mapping tables based on two different re-mapping rules. Ex. PA-3, *Lai*, [0010]-[0011].

default address	0	1	2	3	4	5	6	7	...	996	997	998	999	1000	↖ 70
re-mapped address	1000	999	998	997	996	995	994	993	...	4	3	2	1	0	↖ 75

FIG. 4

address	a				f				k				9				(unused)	(unused)	↖ conversion key
conversion key	97				49				75				57				128	128	
default address	0	1	...	96	97	...	145	146	...	220	221	...	277	278	...	...	...	499	↖ 70
re-mapped address	96	95	...	0	145	...	97	220	...	146	277	...	221	499	...	...	...	496	↖ 75

FIG. 5

Ex. PA-3, *Lai*, Figs. 4-5.

To read data within the protected zone, the data protection system re-maps the default address sequence for the data to the corresponding re-mapped address sequence, reads the ciphertext data from the storage device using the re-mapped address sequence, and decrypts the ciphertext. Ex. PA-3, *Lai*, [0039]-[0046]. *Lai* teaches storing “a protected zone address re-mapping table 65” that contains the mapping between the default address sequence 70 and the protected zone re-mapped address sequence 75 based on a re-mapping rule 60. Ex. PA-3, *Lai*, [0006], [0010]-[0011], [0017]-[0018], [0024], [0037], [0041], claim 3, Figs. 1-5 & 7. This table 65 “can be used for look-up to convert the protected zone default address sequence 70 to the protected zone re-mapped address sequence 75.” *Id.*, [0017]. Accordingly, when obtaining the re-mapped address sequence from the default address sequence during the read operation, the system may use the address re-mapping rule 60, the protected zone address re-mapping table 65, or a mixture of the two. *Id.*, [0039]-[0046], claim 4. Figure 7, below, depicts using the re-mapping table 65 to convert the default address sequence 80 [1, 2, 4, 6, 7, 996] to the re-mapped

address sequence 85 [999, 998, 996, 994, 993, 4]. Ex. PA-3, *Lai*, [0013], [0036]-[0037], [0041]-[0042].

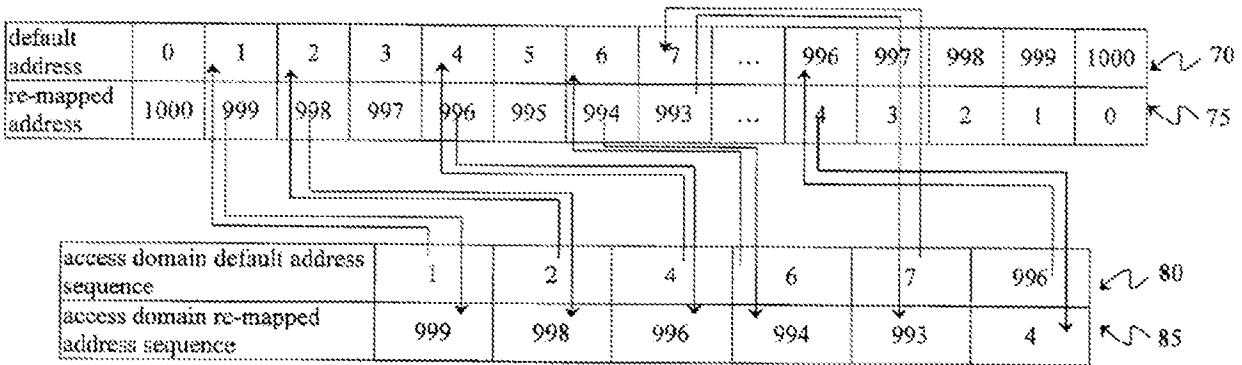


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

Colgrove and *Lai*'s teachings were not previously considered or addressed during the prosecution of the application resulting in the '686 patent. Because *Lai* provides additional teachings beyond those that already provide an SNQ and, in combination with *Colgrove*, further teaches claim features, including claim features in dependent claims 2-3 and 10-11, the teachings of *Colgrove* and *Lai* raise an SNQ for the patentability of 1-3, 5-11, and 13-16. Section VII.B, below, explains how each limitation of claims 1-3, 5-9, and 13-18 of the '686 patent were either disclosed or rendered obvious by the combination of *Colgrove* and *Lai*.

3. The Combined Teachings of *Colgrove* and *Merrell* Raise an SNQ with Respect to Claims 1-3, 5-11, and 13-16

As stated in Section V.B.1, the teachings of *Colgrove* raise an SNQ with respect to claims 1, 5-9, and 13-18. Dependent claim 2 depends from claim 1 and further recites "storing a mapping of the data block numbers and the mapped data block numbers." Ex. A, 6:21-23. Dependent claim 10 is an apparatus claim that contains substantially similar limitations. *Id.*, 6:63-65. The '686 patent specification discloses an embodiment where "[t]he mapping 145 is a

data structure mapping of the blocks in an inode of the file 205 to an [*sic*] algorithm-specific data blocks.” *Id.*, 3:19-21.

Dependent claim 3 also depends from claim 1 and further recites “a read operation for the file.” *Id.*, 6:24-31. The read operation comprises “obtaining the data block numbers of the file,” “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers,” and “reading the data in the mapped data block numbers in the storage device.” *Id.* Dependent claim 11 is an apparatus claim that contains substantially similar limitations. *Id.*, 6:66-7:5.

The combination of *Colgrove* and *Merrell* raises substantial new questions of patentability regarding 1-3, 5-11, and 13-16. *Merrell* contains disclosures that “relate[] to the management of access to shared files managed by file systems.” Ex. PA-2, *Merrell*, 1:7-10. *Merrell* discloses a storage area network (“SAN”) connecting “mass storage systems in a communication network,” a file server managing “access to storage according to a file system architecture,” and an intermediate device processing read/write commands from clients and the file server. *Id.*, 1:22-27, 2:31-60. *Merrell* discloses that processing these commands in the intermediate device improves data security because they lack “the physical addressing parameters” of the file system’s files and, therefore, cannot bypass the file system’s “logical file management processes.” *Id.*, 2:60-64.

Figure 1, below, illustrates clients 10-12 connected to a SAN including file system server 13, intermediate device 15, and storage resources 20-22. *Id.*, 4:37-50. The file system resides in the file server or the intermediate device. *Id.*, 3:1-4.

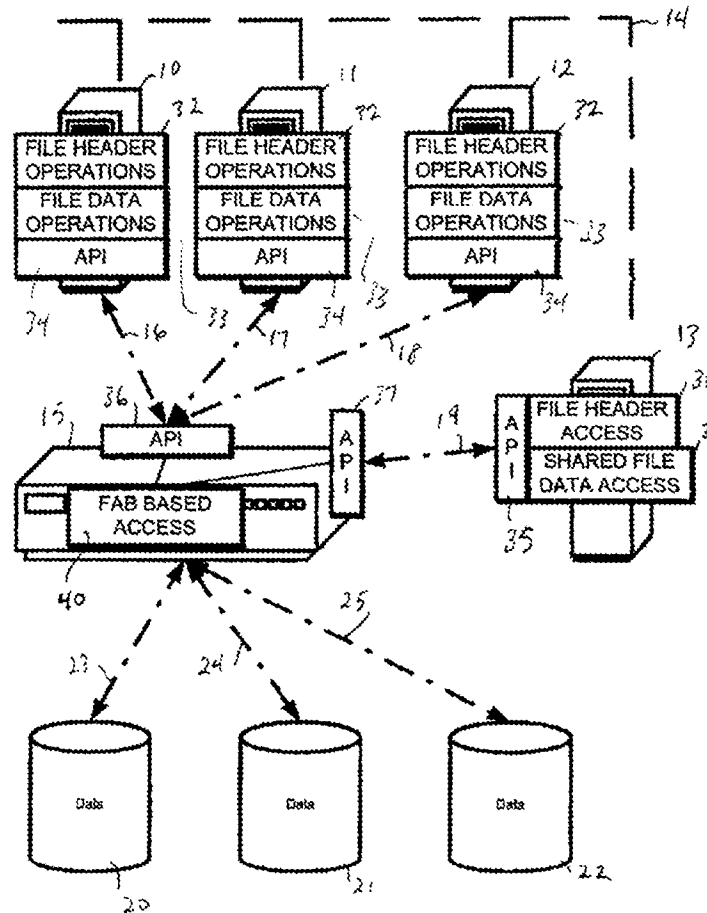


FIG. 1

Id., FIG. 1.

The clients and file server produce read and write commands that “express file access parameters using the logical addressing of the file system, such as using file access block based addressing in which an INODE defines the extent of a file.” *Id.*, 6:51-61. The intermediate device matches “the file access block based file system parameters” to “physical parameters of the storage area network.” *Id.*, 6:51-61, 7:2-8, FIG. 3. To perform this matching, the intermediate device uses “a BMAP function and tables” to find the physical address in storage corresponding “to the logical address within a given file.” *Id.*, 7:15-18. The intermediate device obtains the inode for the requested file and “the logical offset and transfer length needed to satisfy the read or write process,” “comp[li]etes the logical to physical address transformation using this

information,” and uses the resulting physical address to satisfy the read/write command. *Id.*, 7:15-29. This allows a client to make a logical request (e.g., “read file ‘a,’ offset 10 for 100 bytes”) “directly to the SAN fabric.” *Id.*, 4:6-16.

Colgrove and *Merrell*’s teachings were not previously considered or addressed during the prosecution of the application resulting in the ’686 patent. Because *Merrell* provides additional teachings beyond those that already provide an SNQ and, in combination with *Colgrove*, further teaches claim features, including claim features in dependent claims 2-3 and 10-11, the teachings of *Colgrove* and *Merrell* raise an SNQ for the patentability of 1-3, 5-11, and 13-16. Section VII.C, below, explains how each limitation of claims 1-3, 5-9, and 13-18 of the ’686 patent were either disclosed or rendered obvious by the combination of *Colgrove* and *Merrell*.

4. The Combined Teachings of *Lai* and *Card* Raise an SNQ with Respect to Claims 1-3, 5-11, and 13-18

Independent claim 1 is directed to “[a] method of providing data security for use with a file system.” Ex. A, 6:12-31. The claimed method includes, in part, “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file.” *Id.*, 6:14-17. Independent claims 9 and 17 are directed to “[a]n apparatus for providing data security for use with a file system” and contain substantially similar limitations to claim 1. *Id.*, 6:53-62 (independent claim 9 reciting “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and obtain mapped data block numbers after applying the mapping function”), 8:3-12 (independent claim 17 reciting “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file”). Independent claim 18 is directed to an “[a]rticle of manufacture”

and contains substantially similar limitations to claim 1. *Id.*, 8:13-22 (independent claim 18 reciting “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file”). Claims 5-8 depend from claim 1, and claims 13-15 depend from claim 9.

As discussed above for the SNQ based on *Colgrove* (SNQ 1), during prosecution, the Examiner initially rejected all claims as obvious over Kaabouch “in view of well-known features,” of which the Examiner took official notice. Ex. B, 41-42. In response, the Applicant amended the independent claims to require that “the data block numbers are contained in an index node associated with said file.” *Id.*, 76-79. The Applicant argued that the cited prior art did not “suggest or teach and is silent in regards to” this feature. *Id.*, 81-84. The Examiner then allowed the claims, stating that “none of the prior art of record teaches or fairly suggests the claimed” method, apparatus, or article of manufacture with the “specific combination” of claimed features. *Id.*, 94. But the concepts claimed in the ’686 patent were known before the earliest claimed priority date of the ’686 patent, as demonstrated by the combined teachings of *Lai* and *Card*. Indeed, *Card* shows that index nodes were well known and used in common file systems long before the ’686 patent, and it would have been obvious to apply *Lai*’s mapping function to data block numbers in an index node, as demonstrated further below.

Lai relates to a “method for protecting digital information” using a data protection system that maps default addresses into re-mapped addresses within a storage device’s protected zone. Ex. PA-3, *Lai*, Abstract, [0002], [0031]-[0046]. To write data within the protected zone, the data protection system encrypts the data to be written and re-maps the default address sequence for the data, as “designed by the base computer system,” to a re-mapped address sequence. Ex. PA-3, *Lai*, [0031]-[0037]. The data protection system then stores the data as ciphertext into the

re-mapped address sequence. Ex. PA-3, *Lai*, [0038]. Similarly, to read data within the protected zone, the data protection system re-maps the default address sequence for the data to the corresponding re-mapped address sequence, reads the ciphertext data from the storage device using the re-mapped address sequence, and decrypts the ciphertext. Ex. PA-3, *Lai*, [0039]-[0046].

Lai teaches storing “a protected zone address re-mapping table 65” that contains the mapping between the default address sequence 70 and the protected zone re-mapped address sequence 75 based on a re-mapping rule 60. Ex. PA-3, *Lai*, [0006], [0010]-[0011], [0017]-[0018], [0024], [0037], [0041], claim 3, Figs. 1-5 & 7. *Lai* discloses that one example of the re-mapping rule 60 is “ $f(x)=1000-x$.” *Id.*, [0026]. This fits the ’686 patent’s preferred characteristics of a mapping function because every number x maps to another unique number $f(x)$, the mapped address $f(x)$ maps to only one address x that was requested, and the addresses on the storage device are not wasted. *See* Ex. A, 4:15-24.

Lai discloses that the protected zone address re-mapping table 65 “can be used for look-up to convert the protected zone default address sequence 70 to the protected zone re-mapped address sequence 75.” Ex. PA-3, *Lai*, [0017]. Accordingly, when obtaining the re-mapped address sequence from the default address sequence during the read or write operations, the system may use the address re-mapping rule 60, the protected zone address re-mapping table 65, or a mixture of the two. *Id.*, [0031]-[0046], claim 4. Figure 7, below, depicts using the re-mapping table 65 to convert the default address sequence 80 [1, 2, 4, 6, 7, 996] to the re-mapped address sequence 85 [999, 998, 996, 994, 993, 4]. Ex. PA-3, *Lai*, [0013], [0036]-[0037], [0041]-[0042].

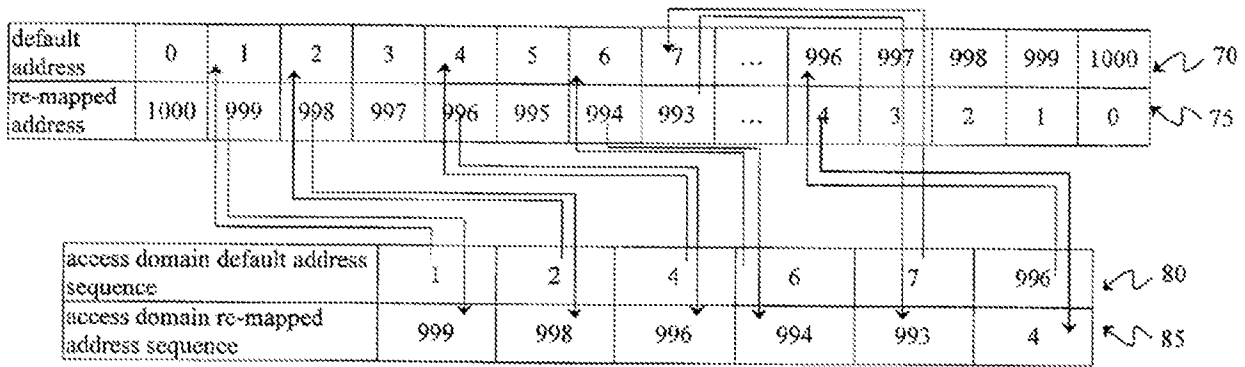
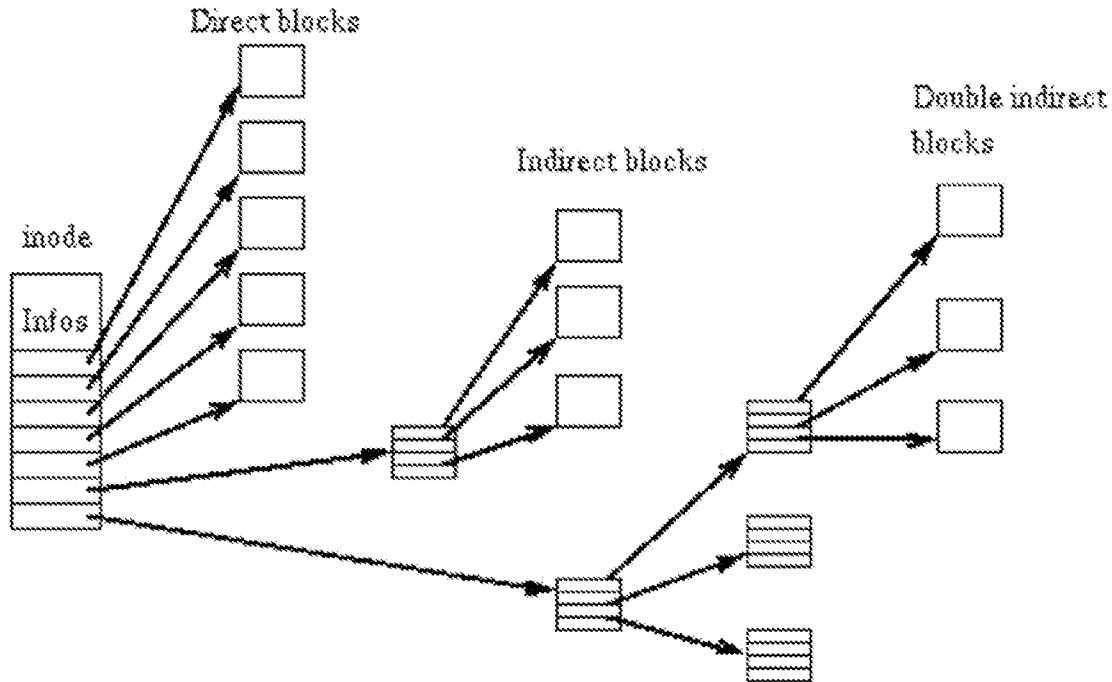


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

While *Lai* does not explicitly discuss inodes, inodes were well-known data structures used in file systems, as evidenced by references like *Card*.¹ *Card* is entitled “Design and Implementation of the Second Extended Filesystem” and relates to the history and implementation of the efs2 filesystem that was widely used with the Linux operating system. See generally Ex. PA-4, *Card*; Ex. C, ¶ 69. *Card* summarizes the basic concepts of Linux filesystems, including their inodes. Ex. PA-4, *Card*, 2-3. Each file in the filesystem is represented by an inode containing information about the file, including “[t]he addresses of data blocks allocated to a file.” Ex. PA-4, *Card*, 2. “When a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table[,] and reads or writes the physical block.” Ex. PA-4, *Card*, 2. The structure of an inode is depicted below.

¹ Citations to *Card* refer to the PDF page number added to the bottom of each page.



Ex. PA-4, *Card*, 2.

Card teaches additional implementation details, including allocating disk blocks to files for storage on disk; deallocating blocks to free them; and creating links between a file's filename and inode number, and thus the corresponding data block numbers. Ex. PA-4, *Card*, 2-5; Ex. C, ¶ 70.

Lai's and *Card*'s teachings were not previously considered or addressed during the prosecution of the application resulting in the '686 patent. *Lai* and *Card* were also not cited on any information disclosure statement or considered by the Examiner at any point during prosecution. See Ex. A, Cover Page (56). Additionally, each of the grounds raised in the pending IPR petition relies on different prior art with a different mapping. See Ex. IPR-1. For example, none of the grounds raised in the IPR petition rely on applying a mapping function like *Lai*'s to data block numbers contained in an inode, as taught by *Card*. *Id.* Because the combination of *Lai* and *Card* teaches the claim features the Examiner thought were missing from the prior art, and the Examiner did not consider *Lai* and *Card*, it raises an SNQ for the patentability of claims 1-3,

5-9, and 13-18. Section VII.D, below, explains how each limitation of claims 1-3, 5-9, and 13-18 of the '686 patent were either disclosed or rendered obvious by the combination of *Lai* and *Card*.

5. The Combined Teachings of *Lai*, *Card*, and *Blumenau* Raise an SNQ with Respect to Claims 4 and 12

As stated in Section V.B.4, the teachings of *Lai* and *Card* raise an SNQ with respect to claims 1-3, 5-11, and 13-18. Dependent claim 4 depends from claim 1 and further recites “a write operation for the file” that comprises “allocating data block numbers of free data blocks in the storage device”; “linking the data block numbers to a file name of the file”; “applying the mapping function to the data block numbers in order to obtain the mapped data block numbers”; and “writing the data of the file into the mapped data block numbers in the storage device.” Ex. A, 6:24-31. Dependent claim 12 is an apparatus claim that contains substantially similar limitations. *Id.*, 7:6-15.

The combination of *Lai*, *Card*, and *Blumenau* raises substantial new questions of patentability regarding claims 4 and 12. *Blumenau* discloses several examples of write operations for writing data to the disk. *E.g.*, Ex. PA-7, *Blumenau*, 4:59-6:24, 10:42-12:50. In *Blumenau*, after the file system creates a metadata entry for the file to be written, the file system accesses the metadata entry of the file, determines that “the size and location fields are empty,” allocates “one or more blocks from free space 330 (e.g., logical block 0 or “the requisite number of blocks from the available blocks in free-space 330”) to store this data, “updates the metadata entry” for the file to reflect the size and location, “call[s] a mapping routine” that “returns the [file] in which is perceived by the host computer to be physical space,” and writes the file data to the allocated blocks. *Id.*, 5:23-6:24, 11:22-12:50.

Lai, *Card*, and *Blumenau*’s teachings were not previously considered or addressed during the prosecution of the application resulting in the '686 patent. Because *Blumenau* provides

additional teachings beyond those that already provide an SNQ and, in combination with *Lai* and *Card*, further teaches claim features, including claim features in dependent claims 4 and 12, the teachings of *Lai*, *Card*, and *Blumenau* raise an SNQ for the patentability of 4 and 12. Section VII.E, below, explains how each limitation of claims 4 and 12 of the '686 patent were either disclosed or rendered obvious by the combination of *Lai*, *Card*, and *Blumenau*.

VI. PROPOSED REJECTIONS OF THE '686 PATENT

The following rejections are proposed in this Request for claims 1-3, 5-11, and 13-18 of the '686 patent:

- SNQ 1: Claims 1, 5-9, and 13-18 are unpatentable under 35 U.S.C. § 102 as anticipated by *Colgrove*.
- SNQ 2: Claims 1, 5-9, and 13-18 are unpatentable under 35 U.S.C. § 103 as being obvious over *Colgrove*.
- SNQ 3: Claims 1-3, 5-11, and 13-16 are unpatentable under 35 U.S.C. § 103 as being obvious over *Colgrove* in view of *Lai*.
- SNQ 4: Claims 1-3, 5-11, and 13-16 are unpatentable under 35 U.S.C. § 103 as being obvious over *Colgrove* in view of *Merrell*.
- SNQ 5: Claims 1-3, 5-11, 13-18 are unpatentable under 35 U.S.C. § 103 as being obvious over *Lai* in view of *Card*.
- SNQ 6: Claims 4 and 12 are unpatentable under 35 U.S.C. § 103 as being obvious over *Lai* in view of *Card* and *Blumenau*.

VII. DETAILED EXPLANATION OF THE PROPOSED REJECTIONS

A. *Colgrove* Anticipates (SNQ 1) or Renders Obvious (SNQ 2) Claims 1, 5-9, and 13-18

As explained below, *Colgrove* anticipates or renders obvious claims 1, 5-9, and 13-18.

1. Overview of *Colgrove*

Colgrove relates to “data protection mechanisms for data systems.” Ex. PA-1, *Colgrove*, 1:6-9. In *Colgrove*, “data system” refers to “any system that stores and manages access to data (e.g. files, records, etc.) typically stored or maintained on some type of logical or physical storage devices.” *Id.*, 1:10-13. A file system is one type of data system that is “a collection of files and file system metadata.” *Id.*, 2:32-37. File systems use metadata “to track where each portion of each file is stored.” *Id.*, 2:41-56. The metadata includes inodes, data structures that each include information about a particular file, including “the addresses of the file’s blocks on disk.” *Id.*, 2:57-3:4.

“[T]he metadata may be necessary to locate, access, reconstruct and/or understand the data itself” because a file system’s data is “stored on the storage device(s) in such a way that the data is essentially unintelligible without having the metadata to locate and reconstruct the data elements into files.” *Id.*, 5:7-16. Accordingly, instead of encrypting the data itself, *Colgrove* discloses a data protection mechanism that encrypts one or more metadata data structures or portions of metadata data structures in a data system, making it “difficult or impossible to reconstruct the stored data elements into data structures such as files.” *Id.*, 4:62-5:3, 5:17-21, 5:46-59.

Figure 4, below, illustrates an example file system 310 “with two layers of metadata (Inode list 320 and Inodes 322A and 322B) between file system software 300 and data 314.” *Id.*, 7:28-31.

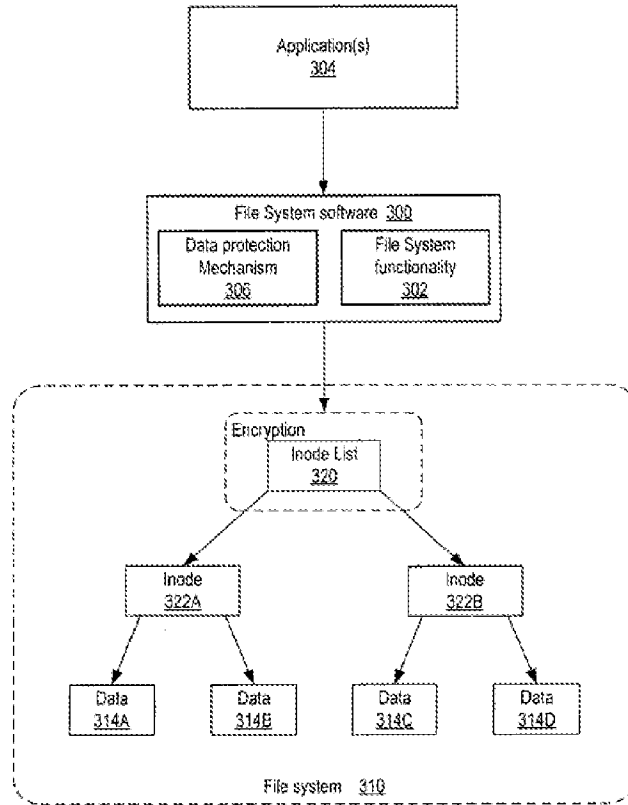


FIG. 4

Id., FIG. 4. “Data 314 may represent files or portions of files” that are “stored in blocks” and “accessible as files via the metadata for the file system 310.” *Id.*, 7:31-34. Inode list 320 includes “information for locating Inodes 322A and 322B,” and Inodes 322A and 322B “include information for locating various elements of data 314 in the file system 310.” *Id.*, 7:34-40. While Figure 4’s Inode list 320 is encrypted, *Colgrove* discloses that, alternatively, the inodes or the portion of the inodes “that allows access to data (e.g. the block maps in inodes) may be encrypted.” *Id.*, 5:56-59, 7:47-51. *Colgrove* discloses “the particular encryption algorithm used is not significant” and “a particular encryption algorithm may be selected to match the security and performance requirements of the file system 310.” *Id.*, 7:51-56.

Data system software (e.g., file system software 300) includes a data protection mechanism (e.g., data protection mechanism 306), which encrypts and, when necessary, decrypts portions of the metadata, and data system functionality (e.g., file system functionality 302) that uses unencrypted metadata to access and manage elements of the stored data, including performing operations like “storing, creating, deleting, copying, moving, and modifying elements” of the data. *Id.*, 4:64-5:3, 6:5-7, 6:17-22, 6:40-53, 7:40-45. In one embodiment, depicted below, the data system software “decrypt[s] the encrypted metadata from the data system to generate a decrypted copy of the metadata,” “cache[s] the decrypted copy of the metadata in memory,” and “access[es] the data in the data system in accordance with the decrypted copy of the metadata cached in memory.” *Id.*, 11:51-58.

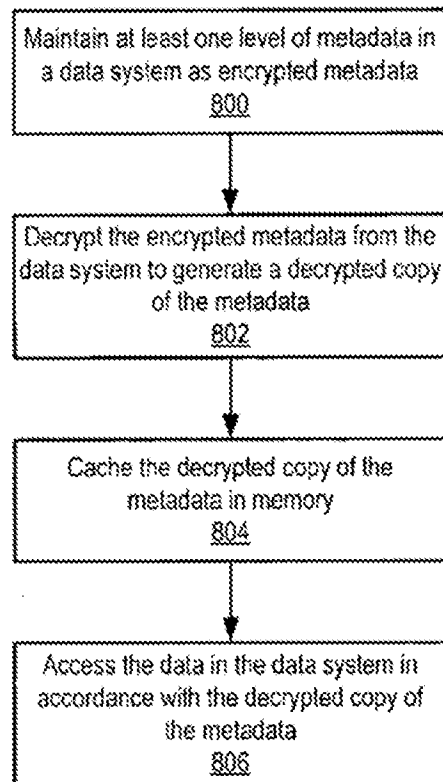


FIG. 9

Id., FIG. 9.

Colgrove is analogous art because it is in the same field of endeavor as the '686 patent—protecting digital information in storage devices. *Compare id.*, Title, Abstract, with Ex. A, Title, Abstract, Claims 1, 9, 17, 18; Ex. C, ¶ 60. *Colgrove* is also analogous art because it is reasonably pertinent to a problem discussed in the '686 patent because it deals with preventing users from reconstructing files stored in a file system using metadata. *Compare* Ex. PA-1, *Colgrove*, 4:64-5:21, with Ex. A, 1:25-32, 4:34-38; Ex. C, ¶ 60.

2. Independent Claim 1

a. 1[pre]: “A method of providing data security for use with a file system, the method comprising:”

If the preamble is limiting, *Colgrove* discloses or suggests 1[pre]. Ex. C, ¶¶ 80-81.

Colgrove discloses “protecting data systems such as file systems” by encrypting “one or more elements or levels of metadata.” Ex. PA-1, *Colgrove*, Abstract; Ex. C, ¶ 81. *Colgrove* discloses this encryption protects the data in the file system “because, without access to the unencrypted metadata, it may be difficult or impossible to reconstruct the stored data elements into data structures such as files or records.” Ex. PA-1, *Colgrove*, 5:17-21. *Colgrove* also discloses that, “instead of encrypting the entire metadata data structure, only a portion of the metadata data structure may be encrypted.” Ex. PA-1, *Colgrove*, 3:23-25, 5:46-59.

b. 1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”

Colgrove discloses or suggests 1[a]. Ex. C, ¶¶ 82-85.

First, *Colgrove* discloses “data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file.” *Colgrove* discloses file systems that use inodes (i.e., “index node[s]”) that “hold information about files in a file

system,” where each inode corresponds to a particular file in the file system. Ex. PA-1, *Colgrove*, 2:57-63; Ex. C, ¶ 83. *Colgrove* discloses that each inode includes “the addresses of the file’s blocks on disk (and/or pointers to indirect blocks that reference the file blocks).” Ex. PA-1, *Colgrove*, 2:57-58, 2:63-3:4. *Colgrove* discloses the file system may encrypt a portion of each inode “that allows access to data,” such as the addresses of the file’s blocks. Ex. PA-1, *Colgrove*, 5:46-59 (encrypting “the block maps in inodes”), 7:49-51; Ex. C, ¶ 83. These encrypted data block addresses are “data block numbers that are associated with a file” and “contained in an index node associated with said file.” Ex. PA-1, *Colgrove*, 2:47-50, 2:57-58, 2:63-3:4; Ex. C, ¶ 83; *see* Ex. A, 2:15-17 (“As known to those skilled in the art, an inode (index node) is a data structure that contains information about a file, directory, or other object in a file system.”).

Second, *Colgrove* discloses “applying a mapping function to data block numbers,” because it discloses a data protection mechanism that decrypts the encrypted portion of the metadata, such as the encrypted addresses of the file’s data blocks. Ex. PA-1, *Colgrove*, 3:32-36, 3:57-4:4, 5:56-59, 4:64-5:3, Fig. 9; Ex. C, ¶ 84. The ’686 patent describes a “mapping function” as including a basic multiplicative cipher such as $f(x)=5x$ or a similar function that “converts (maps)” the data block numbers to mapped data block numbers. Ex. A, 3:61-66, 5:29-35. *Colgrove*’s decryption, depicted at step 802 below, converts the encrypted addresses of the file’s data blocks to mapped data block numbers like the “mapping function” in the ’686 patent. Ex. PA-1, *Colgrove*, 5:56-59, 7:49-51; Ex. C, ¶ 84.

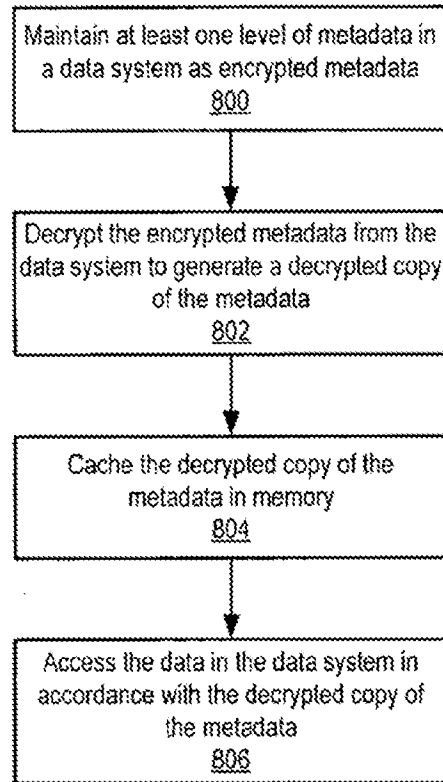


FIG. 9

Ex. PA-1, *Colgrove*, FIG. 9.

Colgrove discloses that any encryption algorithm may be used and “selected to match the security and performance requirements of the file system 310.” Ex. PA-1, *Colgrove*, 7:51-56.

This disclosure teaches or at least suggests using a symmetric encryption algorithm, which was a well-known example of a mapping function. Ex. C, ¶ 85; *see* Ex. PA-5, *Carter*, [0005]-[0008] (disclosing several well-known symmetric encryption methods, which “utilize a mathematical algorithm to transform the legible data (plaintext) into an encrypted form (ciphertext), that can not be comprehended without the knowledge and use of a key to decrypt the encrypted form”). For example, *Lai* discloses using the well-known “Exclusive Or” operation using the same key for encryption and decryption. Ex. PA-3, *Lai*, [0033]-[0034], [0044]-[0045]. An Exclusive Or

operation compares the bits of two binary numbers and, for each bit, returns a “0” if the bits are the same and a “1” if the bits differ. Ex. C, ¶ 85. To demonstrate, if an address is 1 and the symmetric key is 5, then the encryption using an Exclusive Or (XOR) operation would be processed as follows:

$$\begin{array}{r} 001 \text{ (plaintext address)} \\ \text{XOR } 101 \text{ (key)} \\ \hline 100 \text{ (encrypted address)} \end{array}$$

Ex. C, ¶ 85. Applying the same key to the encrypted address using the same Exclusive Or operation then returns the plaintext address:

$$\begin{array}{r} 100 \text{ (encrypted address)} \\ \text{XOR } 101 \text{ (key)} \\ \hline 001 \text{ (plaintext address)} \end{array}$$

Ex. C, ¶ 85. Other more sophisticated encryption algorithms, like DES and AES, use an Exclusive Or operation in combination with additional operations (e.g., translation and permutation). Ex. C, ¶ 85.

- c. **1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Colgrove discloses or suggests 1[b]. Ex. C, ¶¶ 86-87.

Colgrove discloses that, after the encrypted portions of the metadata are decrypted (“after applying the mapping function”), the data system generates a copy of the decrypted metadata and caches the copy in memory. Ex. PA-1, *Colgrove*, 11:51-55, Fig. 9; Ex. C, ¶ 87. When the encrypted portions of the metadata are the addresses of the file’s data blocks, the decrypted portions of the metadata are “mapped data block numbers” that “are addresses of data of the file in a storage device.” Ex. PA-1, *Colgrove*, 2:57-3:4, 5:46-59; Ex. C, ¶ 87. *Colgrove* discloses that “data system software may then access the data in the data system in accordance with the decrypted copy of the metadata cached in memory.” Ex. PA-1, *Colgrove*, 6:40-56, 11:55-60,

FIG. 9. Storing decrypted addresses of the file's blocks in memory and/or using decrypted addresses to access data in the data systems discloses "obtaining mapped data block numbers after applying the mapping function." Ex. C, ¶ 87.

3. Claim 5: "The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file."

Colgrove discloses or suggests claim 5. Ex. C, ¶¶ 88-90.

As discussed for 1[a], *Colgrove* discloses applying a mapping function on data block numbers in an inode by decrypting "the addresses of the file's blocks on disk." Ex. PA-1, *Colgrove*, 2:57-3:4, 4:64-5:3, 5:46-59, 11:51-54; Ex. C, ¶ 89; *supra* Section VII.A.2.b (1[a]). *Colgrove* discloses that any encryption algorithm "may be used to encrypt the metadata," such as well-known symmetric encryption algorithms that apply an equation based on the same key to both encrypt and decrypt data. Ex. PA-1, *Colgrove*, 7:51-56; *see* Ex. PA-5, *Carter*, [0005]-[0008] (disclosing several well-known symmetric encryption algorithms that use equations); Ex. PA-3, *Lai*, [0033]-[0034], [0044]-[0045] (disclosing an encryption/decryption equation using an "Exclusive Or" operation); Ex. C, ¶ 89. Accordingly, to decrypt the file's data block addresses and "generate a decrypted copy of the metadata," the data protection mechanism applies the decryption equation to each encrypted data block number of the file to obtain the mapped data block numbers. Ex. PA-1, *Colgrove*, 11:51-54; Ex. C, ¶ 89.

To the extent it is argued or determined that *Colgrove* does not explicitly disclose that the mapping function is an equation applied to each data block number, it would have been obvious to a POSITA. Ex. C, ¶ 90. *Colgrove* discloses that "[a]ny of various types of encryption algorithms may be used" and that "a particular encryption algorithm may be selected to match the security and performance requirements of the file system 310." Ex. PA-1, *Colgrove*, 7:51-56. Before the priority date, it was well-known that "encryption algorithms" employ an equation to

encrypt/decrypt data. A POSITA would have been motivated to choose one of many well-known symmetric encryption/decryption algorithms to encrypt and decrypt each data block number in the inode. Ex. C, ¶ 90. For example, symmetric encryption algorithms, like an Exclusive Or (XOR), the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES), were well known before the '686 patent and likewise applied an equation to each portion of the data. Ex. C, ¶ 90; Ex. PA-3, *Lai*, [0033]-[0034], [0044]-[0045]; Ex. PA-5, *Carter*, [0005]-[0008]. A POSITA would have been motivated to use one of these well-known encryption/decryption algorithms employing equations because doing so would have combined well-known prior art elements according to known methods to yield predictable results. Ex. C, ¶ 90. These were standardized algorithms and would have been routine to implement with simple software functions, giving a POSITA a reasonable expectation of success. Ex. C, ¶ 90.

4. Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”

Colgrove discloses or suggests claim 6. Ex. C, ¶¶ 91-92.

As discussed for 1[a], *Colgrove* discloses applying a mapping function on data block numbers in an inode by decrypting “the addresses of the file’s blocks on disk.” Ex. PA-1, *Colgrove*, 2:57-3:4, 4:64-5:3, 5:46-59, 11:51-54; Ex. C, ¶ 92; *supra* Section VII.A.2.b (1[a]). *Colgrove* teaches “selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function” because it discloses that “a particular encryption algorithm may be selected to match the security and performance requirements of the file system 310.” Ex. PA-1, *Colgrove*, 7:51-56. A POSITA would have understood *Colgrove* as teaching that selecting an encryption algorithm with a higher complexity level may both increase security and decrease performance (“efficiency”), and that the complexity level should be selected according to the system requirements. Ex. C, ¶ 92.

5. Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”

The '686 patent discloses an embodiment where “the data blocks of the file 205 are not mapped” and “the locations of the addresses in the storage device 125 of the data of the file 205 are identified by the data block numbers of the inode 210.” Ex. A, 3:13-49. To the extent this embodiment is within the scope of claim 7, *Colgrove* discloses or suggests claim 7. Ex. C, ¶¶ 93-95.

Colgrove discloses that each inode for a file may include “the addresses of the file’s blocks on disk.” Ex. PA-1, *Colgrove*, 2:57-58, 2:63-3:4. These addresses may be in the form of “device:block number(s).” Ex. PA-1, *Colgrove*, 2:47-50; Ex. C, ¶ 94.

Colgrove discloses that “only a portion of the data 214 in a data system may need to be protected,” so the data protection mechanism may “encrypt only a portion of the metadata that references the portion of the data 214 in the data system that needs to be protected.” Ex. PA-1, *Colgrove*, 3:43-48, 6:61-7:6. For example, the data protection may only encrypt addresses of a file’s data blocks for inodes corresponding to files needing protection and leave unencrypted the addresses in inodes for files not needing protection. Ex. PA-1, *Colgrove*, 2:57-3:4, 5:56-59, 6:61-7:6. For the unencrypted inodes, the data block numbers are not encrypted and, thus, are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers. Ex. C, ¶ 95.

6. Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”

Colgrove discloses or suggests claim 8. Ex. C, ¶¶ 96-97.

Colgrove discloses that “the metadata may be necessary to locate, access, reconstruct and/or understand the data itself,” so “it may be difficult or impossible to reconstruct the data

into a meaningful form without having access to the metadata.” Ex. PA-1, *Colgrove*, 5:7-12.

Colgrove discloses that accessing data includes, e.g., copying the data. Ex. PA-1, *Colgrove*, 6:48-60. By encrypting “a portion of metadata that allows access to data,” such as the addresses of the file’s data blocks, *Colgrove*’s mapping function prevents copying of data of the file in the disk (“storage device”) because unauthorized users cannot decrypt the addresses without the key and, therefore, cannot use the inode to determine where on the disk the file’s data is stored, copy the data, and reconstruct the file. Ex. PA-1, *Colgrove*, 5:46-59; Ex. C, ¶ 97. This solves the known problem in unencrypted file systems where an unauthorized user that gains access to the metadata directory “could access the i-nodes to recover the data associated with the file.” Ex. PA-5, *Carter*, [0218]; Ex. C, ¶ 97.

7. Independent Claim 9

a. 9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”

If the preamble is limiting, *Colgrove* discloses or suggests “[a]n apparatus for providing data security for use with a file system.” Ex. C, ¶¶ 98-99.

Colgrove discloses or suggests “providing data security for use with a file system” for the reasons discussed above for 1[pre]. *Supra* Section VII.A.2.a (1[pre]). *Colgrove* discloses “[a]n apparatus for providing” this data security because it discloses file system software encompasses “any associated hardware needed to store and/or execute” the software and includes “file system functionality 302 for performing the operations of the file system 310 and a data protection mechanism 306 that implements data protection for the data 314 in the file system” via encryption/decryption of metadata. Ex. PA-1, *Colgrove*, 1:51-62, 7:11-46; Ex. C, ¶ 99. The memory storing the file system software and the processor for executing the software teach an apparatus for providing this data security. Ex. C, ¶ 99.

- b. 9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”**

Colgrove discloses or suggests “apply[ing] a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed for 1[a]. *Supra* Section VII.A.2.b (1[a]).

Colgrove discloses or suggests “a block distribution engine configured to apply” the mapping function because it discloses file system software, including a data protection mechanism that decrypts the encrypted data block numbers of a file. Ex. PA-1, *Colgrove*, 4:64-5:3, 6:40-51; Ex. C, ¶ 100; *supra* Section VII.A.2.b (1[a]).

- c. 9[b]: “obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Colgrove discloses or suggests “obtain[ing] mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed for 1[b]. *Supra* Section VII.A.2.c (1[b]). *Colgrove* discloses or suggests “a block distribution engine configured to” obtain the mapped data block numbers because it discloses file system software, including a data protection mechanism that obtains mapped data block numbers after decrypting the encrypted data block numbers (“applying the mapping function”). Ex. PA-1, *Colgrove*, 4:64-5:3, 6:40-51; Ex. C, ¶ 101; *supra* Section VII.A.2.c (1[b]).

A POSITA would have understood that this functionality performed by *Colgrove*’s data protection mechanism is akin to that performed by the block distribution engine in the ’686 patent, which “appl[ies] the mapping function” to the data block numbers and “return[s] the

corresponding mapped block numbers . . . to the disk I/O subsystem 135.” Ex. A, 4:49-57; Ex. C, ¶ 102.

8. **Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”**

Colgrove discloses or suggests the additional limitations of claim 13 for the reasons discussed for Claim 5. Ex. C, ¶ 103.

9. **Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”**

Colgrove discloses or suggests the additional limitations of claim 14 for the reasons discussed for Claim 6. Ex. C, ¶ 104.

10. **Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”**

Colgrove discloses or suggests the additional limitations of claim 15 for the reasons discussed for Claim 7. Ex. C, ¶ 105.

11. **Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”**

Colgrove discloses or suggests the additional limitations of claim 16 for the reasons discussed for Claim 8. Ex. C, ¶ 106.

12. **Independent Claim 17**

- a. **17[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”**

If the preamble is limiting, *Colgrove* discloses or suggests “[a]n apparatus for providing data security for use with a file system” for the reasons discussed for 9[pre]. Ex. C, ¶ 107.

- b. **17[a]: “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”**

Colgrove discloses or suggests “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed for 9[a]. Ex. C, ¶ 108. As discussed in Section IV.E.1, a block distribution engine, which is recited in 9[a], is structure for the means-plus-function term in 17[a]. *Supra* Section IV.E.1, VII.A.7.b (9[a]); Ex. C, ¶ 108.

- c. **17[b]: “means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Colgrove discloses or suggests “means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed for 9[b]. Ex. C, ¶ 109. As discussed in Section IV.E.2, a block distribution engine, which is recited in 9[a] and performs the function in 9[b], is structure for the means-plus-function term recited in 17[b]. *Supra* Sections IV.E.2, VII.A.7.c (9[b]); Ex. C, ¶ 109.

13. Independent Claim 18

- a. **18[a]: “Article of manufacture, comprising: a machine-readable medium having stored thereon instructions to:”**

Colgrove discloses or suggests an “[a]rticle of manufacture, comprising: a machine-readable medium having stored thereon instructions.” Ex. C, ¶¶ 110-111.

Colgrove discloses “hardware needed to store and/or execute” file system software including “file system functionality 302 for performing the operations of the file system 310 and a data protection mechanism 306 that implements data protection for the data 314 in the file

system” via encryption/decryption of metadata. Ex. PA-1, *Colgrove*, 1:51-62, 7:11-46; Ex. C, ¶ 111. The memory (“Article of manufacture, comprising: a machine-readable medium”) having stored instructions for performing the functions of 18[a-b], *infra*, is depicted in the “exemplary generic data system” below. Ex. PA-1, *Colgrove*, 1:29-46 (storage device “may be any type of computer-accessible medium”), 6:5-7, 12:37-44, 14:18-28, FIG. 3B; Ex. C, ¶ 111.

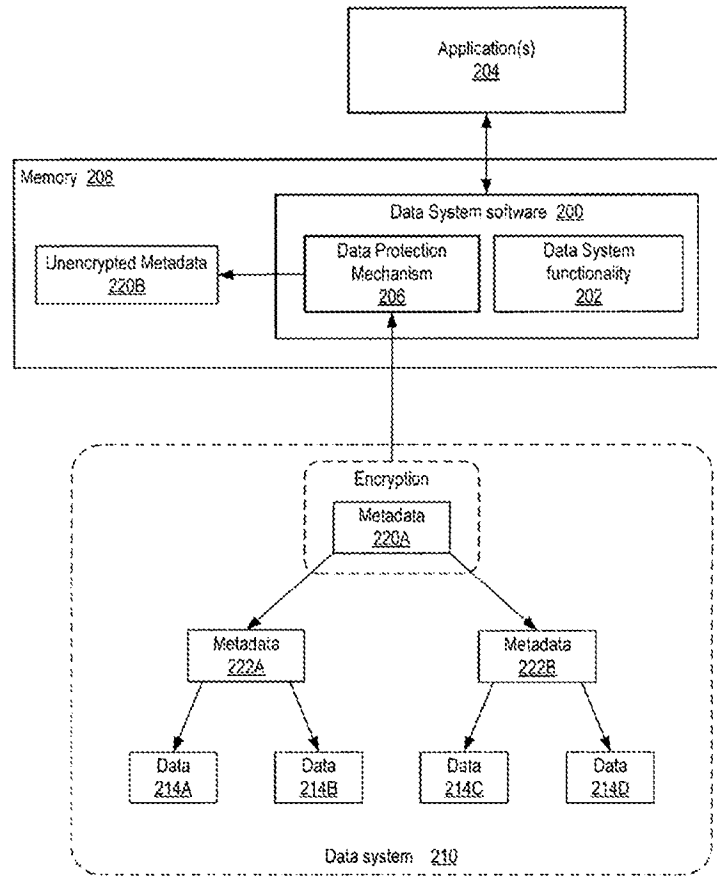


FIG. 3B

Ex. PA-1, *Colgrove*, FIG. 3B.

- b. 18[a]: “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”**

Colgrove discloses or suggests “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed for 1[a]. Ex. C, ¶ 112.

- c. 18[b]: “obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Colgrove discloses or suggests “obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed for 1[b]. Ex. C, ¶ 113.

B. SNQ 3: *Colgrove* in view of *Lai* Renders Obvious Claims 1-3, 5-11, and 13-16

1. Overview of *Lai*

Lai relates to a “method for protecting digital information” using a data protection system that maps default addresses into re-mapped addresses within a storage device’s protected zone. Ex. PA-3, *Lai*, Abstract, [0002], [0031]-[0046]. To write data within the protected zone, the data protection system encrypts the data to be written and re-maps the default address sequence for the data, as “designed by the base computer system,” to a re-mapped address sequence. Ex. PA-3, *Lai*, [0031]-[0037]. The data protection system then stores the data as ciphertext into the re-mapped address sequence. Ex. PA-3, *Lai*, [0038]. Similarly, to read data within the protected zone, the data protection system re-maps the default address sequence for the data to the corresponding re-mapped address sequence, reads the ciphertext data from the storage device using the re-mapped address sequence, and decrypts the ciphertext. Ex. PA-3, *Lai*, [0039]-[0046].

Lai teaches storing “a protected zone address re-mapping table 65” that contains the mapping between the default address sequence 70 and the protected zone re-mapped address sequence 75 based on a re-mapping rule 60. Ex. PA-3, *Lai*, [0006], [0010]-[0011], [0017]-[0018], [0024], [0037], [0041], claim 3, Figs. 1-5 & 7. This table 65 “can be used for look-up to convert the protected zone default address sequence 70 to the protected zone re-mapped address sequence 75.” *Id.*, [0017]. Figures 4 and 5, below, depict sample re-mapping tables based on two re-mapping rules. Ex. PA-3, *Lai*, [0010]-[0011].

default address	0	1	2	3	4	5	6	7	...	996	997	998	999	1000	↖ 70
re-mapped address	1000	999	998	997	996	995	994	993	...	4	3	2	1	0	↖ 75

FIG. 4

address conversion key	a				b				K				9				(unused)	(unused)	↖ 70/75
ASCII code	97				98				75				77				128	128	
default address	0	1	...	96	97	...	145	146	...	230	231	...	277	278	...	...	...	499	↖ 70
re-mapped address	96	95	...	0	145	...	97	220	...	146	277	...	221	405	...	...	...	496	↖ 75

FIG. 5

Ex. PA-3, *Lai*, Figs. 4-5.

Accordingly, when obtaining the re-mapped address sequence from the default address sequence during the read or write operations, the system may use the address re-mapping rule 60, the protected zone address re-mapping table 65, or a mixture of the two. *Id.*, [0031]-[0046], claim 4. Figure 7, below, depicts using the re-mapping table 65 to convert the default address sequence 80 [1, 2, 4, 6, 7, 996] to the re-mapped address sequence 85 [999, 998, 996, 994, 993, 4]. Ex. PA-3, *Lai*, [0013], [0036]-[0037], [0041]-[0042].

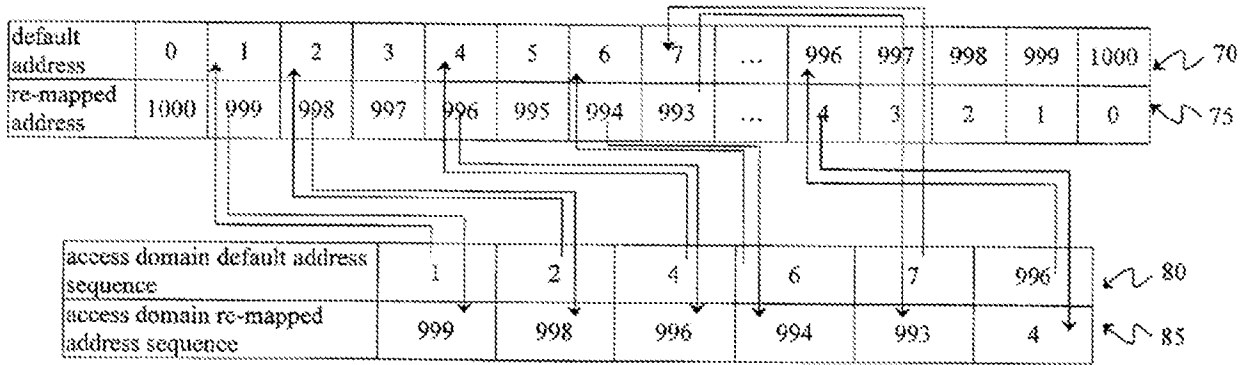


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

Lai is analogous art at least because it is in the same field of endeavor as the '686 patent—protecting digital information in storage devices. Ex. PA-3, *Lai*, Title, Abstract, [0002], [0005]-[0006], [0047]; Ex. C, ¶ 68. *Lai* is also analogous art because it is reasonably pertinent to the particular problem the patent at issue is concerned with because it deals with technologies relating to converting data location information into mapped data location information. Compare Ex. PA-3, *Lai*, Title, Abstract, [0002], [0005]-[0006], Figs. 1-5, with Ex. A, Abstract, 1:53-61, 5:30-52, Claims 1, 9, 17, 18; Ex. C, ¶ 68.

2. Reasons to Combine *Colgrove* and *Lai*

To the extent Patent Owner contends *Colgrove* lacks a mapping function that uses a stored table containing the mapping between the encrypted addresses in the inode and the addresses where the file's data is stored, it would have been obvious to incorporate that functionality based on the teachings of *Lai*. Ex. C, ¶ 115. *Colgrove* and *Lai* are each analogous art to the '686 patent for the reasons discussed in Sections VII.A.1 and VII.B.1.

Colgrove discloses its data system may have multiple layers of metadata, where higher layers “may be necessary to locate, access, reconstruct, and/or understand the one or more layers of metadata below the higher layer.” Ex. PA-1, *Colgrove*, 5:7-25. *Colgrove* discloses that the

data protection mechanism of the file system software “implements data protection for the data 314 in the file system via encryption of metadata on one or more levels.” Ex. PA-1, *Colgrove*, 6:17-22. *Colgrove* discloses its data system software “decrypt[s] the encrypted metadata from the data system to generate a decrypted copy of the metadata” and “cache[s] the decrypted copy of the metadata in memory.” Ex. PA-1, *Colgrove*, 11:51-54. “[T]he data system software may then access the data in the data system in accordance with the decrypted copy of the metadata cached in memory.” Ex. PA-1, *Colgrove*, 11:55-58. It would have been obvious to a POSITA in view of *Lai* to store a mapping table as another layer of metadata below the inode and to apply a mapping function to the encrypted addresses in the inode to identify the corresponding unencrypted addresses in the table to implement *Colgrove*’s data protection techniques. Ex. C, ¶ 116.

Lai discloses a “data protection system” similar to *Colgrove*’s, where the system applies a mapping function to a sequence of addresses to obtain a sequence of mapped addresses that identifies where in the storage area data is stored. Ex. PA-3, *Lai*, [0002], [0005]-[0006], [0031]-[0046]; Ex. C, ¶ 117. *Lai* discloses storing a “protected zone address re-mapping table 65,” which converts a default address sequence into a mapped address sequence. Ex. PA-3, *Lai*, [0024]. This table 65 “can be used for look-up to convert the protected zone default address sequence 70 to the protected zone re-mapped address sequence 75.” *Id.*, [0017]. Accordingly, when obtaining the re-mapped address sequence from the default address sequence during the read operation, the system may use the address re-mapping rule 60, the protected zone address re-mapping table 65, or a mixture of the two. *Id.*, [0039]-[0046], claim 4; Ex. C, ¶ 117. Figure 7, below, depicts using the re-mapping table 65 to convert the default address sequence 80 [1, 2, 4, 6, 7, 996] to the re-mapped address sequence 85 [999, 998, 996, 994, 993, 4]. Ex. PA-3, *Lai*, [0013], [0036]-[0037], [0041]-[0042].

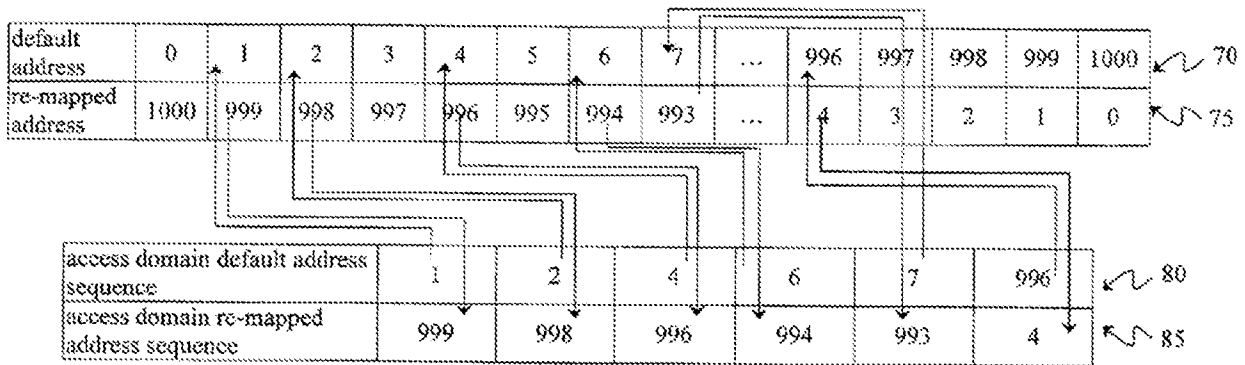


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

A POSITA would have been motivated to combine *Colgrove* and *Lai* to store a table containing the mapping between the encrypted addresses of the file (data block numbers) and the corresponding unencrypted addresses (mapped data block numbers). Ex. C, ¶ 118. For example, *Colgrove* discloses that file system metadata may include both inodes and “mapping information in the form of indirect blocks.” Ex. PA-1, *Colgrove*, 2:42-47. The inode for a corresponding file may include “addresses of the file’s blocks on disk” as well as “pointers to indirect blocks that reference the file blocks.” Ex. PA-1, *Colgrove*, 2:57-3:4. Because the size of the inode was limited, the pointers to indirect blocks provided a metadata hierarchy that allowed much larger file sizes, where the indirect pointers within the inode would lead to another data structure containing the indirect blocks of the file, as depicted in the diagram below. Ex. C, ¶ 118. For larger files, the vast majority of the addresses for the file’s data blocks were found within these separate data structures. Ex. C, ¶ 118; Ex. PA-6, *Mauro*, 584-585 (disclosing the inode’s direct blocks allow 12 addresses, the first level of indirection allows 2048 addresses, and the second level of indirection allows 4,194,304 addresses).

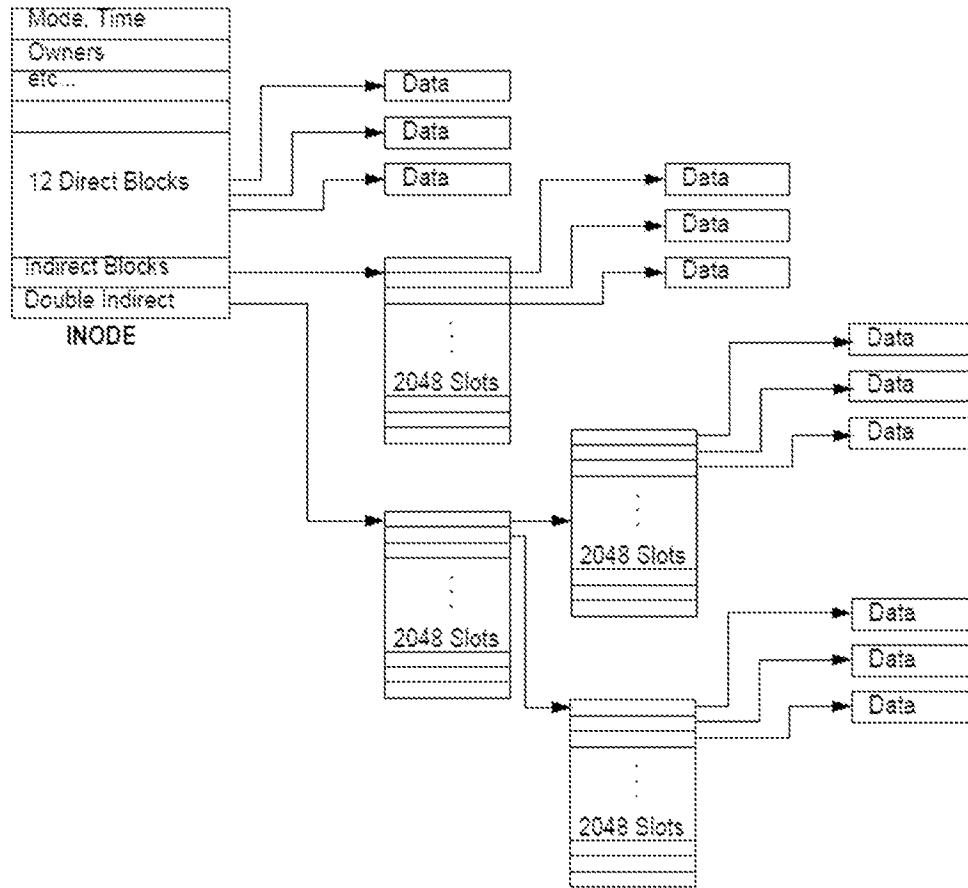


Figure 14.5 The UFS inode Format

Ex. PA-6, *Mauro*, 584.

A POSITA would have recognized that decrypting each of the addresses in both the inode and the indirect data structure(s) each time a file is accessed may impact performance of the file system. Ex. C, ¶ 119; *see* Ex. PA-1, *Colgrove*, 5:34-38 (“There may be a cost in performance when encrypting or decrypting a data structure.”). To limit the number of decryption operations needed to access a file’s addresses on disk, a POSITA would have seen the benefits of storing a table, like in *Lai*, that contains the mapping between the encrypted addresses of the file and the corresponding unencrypted addresses. Ex. PA-3, *Lai*, claim 4 (disclosing that performing an “address conversion” using a table “is simpler than” using a mapping rule); Ex. C, ¶ 119. Just as *Lai*’s data protection system uses a table to convert a default address sequence to a mapped

address sequence when reading or writing data, a POSITA would have recognized that implementing this table would have allowed *Colgrove*'s file system software to apply a mapping function to the inode's encrypted addresses and identify the corresponding unencrypted addresses by locating the encrypted address in the table and returning the corresponding unencrypted address. Ex. C, ¶ 119. This would have increased performance of the system because the data protection mechanism of the file system would only need to encrypt the file's addresses once and then store the mapping between the encrypted addresses contained in the inode and the corresponding unencrypted addresses on disk in a table for later access. Ex. C, ¶ 119.

Indeed, *Colgrove* suggests such a configuration because it discloses encrypting "a metadata data structure that does not change very frequently and/or is not accessed very frequently to minimize the cost of encryption and decryption" and encrypting "only a portion of the metadata data structure," such as a portion "that allows access to data." Ex. PA-1, *Colgrove*, 5:34-59; Ex. C, ¶ 120. A POSITA would have recognized that the address to the table would not frequently change and that the file system software would maintain protection over the data if it encrypted the address of the table containing the mapping, similar to *Colgrove*'s alternative embodiments where higher level metadata data structure, like the inode list, is encrypted. Ex. C, ¶ 120; Ex. PA-1, *Colgrove*, 7:47-49. Doing so would have beneficially allowed the file system software to perform only one decryption operation of the table address each time the file is read. Ex. C, ¶ 120.

A POSITA would have reasonably expected success in implementing a mapping table with the encrypted addresses of the file and the corresponding unencrypted addresses. Ex. C, ¶ 121. Both *Colgrove* and *Lai* relate to data security and teach applying a mapping function to

obtain the actual addresses where the desired data is stored. Ex. C, ¶ 121. *Lai* discloses that using a mapping table to convert one type of address to another was well known. Ex. PA-3, *Lai*, [0036]-[0037], [0041]-[0042], FIG. 7; Ex. C, ¶ 121. A POSITA would have recognized that implementing the mapping table and performing the corresponding mapping function would require only conventional software functions where the encrypted addresses of the requested file blocks are used to identify the corresponding unencrypted addresses in the table. Ex. C, ¶ 121. Combining *Colgrove* and *Lai* in this way would have involved the use of a well-known technique to improve *Colgrove*'s file system.

3. Independent Claim 1

- a. **1[pre]: “A method of providing data security for use with a file system, the method comprising:”**

Colgrove discloses or suggests “[a] method of providing data security for use with a file system” for the reasons discussed in Section VII.A.2.a. Ex. C, ¶ 122.

- b. **1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”**

Colgrove discloses or at least suggests “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed in Section VII.A.2.b. Ex. C, ¶ 123.

Alternatively, *Colgrove* in view of *Lai* renders obvious this limitation. Ex. C, ¶¶ 123-124.

As discussed in Section VII.A.2.b, *Colgrove* discloses encrypted addresses (data block numbers) that are associated with a file and contained in an inode associated with the file. Ex. C, ¶ 124. For the reasons discussed above in Section VII.B.2, it would have been obvious to combine *Colgrove* and *Lai* to store a table containing the mapping between the encrypted addresses contained in an inode (the data block numbers) and the corresponding addresses on

disk (the mapped data block numbers). Ex. C, ¶ 124; *supra* Section VII.B.2. In this configuration, upon receiving a read request for the file, it would have been obvious for the reasons discussed in Section VII.B.2 to apply a mapping function to the encrypted addresses contained in the inode to obtain the corresponding unencrypted addresses in the table. Ex. C, ¶ 124.

c. 1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”

Colgrove discloses or at least suggests “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed in Section VII.A.2.c. Ex. C, ¶ 125. Alternatively, *Colgrove* in view of *Lai* renders obvious this limitation. Ex. C, ¶ 125-126.

As discussed in Section VII.B.3.b, it would have been obvious to apply a mapping function to the encrypted addresses contained in the inode to obtain the corresponding unencrypted addresses in the table. Ex. C, ¶ 126. The unencrypted addresses obtained after applying the mapping functions are mapped data block numbers, which are addresses of data of the file in a storage device. Ex. C, ¶ 126; *supra* Sections VII.A.2.c, VII.B.2.

4. Claim 2: “The method of claim 1, further comprising: storing a mapping of the data block numbers and the mapped data block numbers.”

Colgrove discloses or at least suggests the method of claim 1 for the reasons discussed in Section VII.A.2 (SNQs 1-2, claim 1). Alternatively, *Colgrove* in view of *Lai* renders obvious claim 1 for the reasons discussed in Section VII.B.3. *Colgrove* in view of *Lai* also renders obvious “storing a mapping of the data block numbers and the mapped data block numbers.” For the reasons discussed above in Section VII.B.2, it would have been obvious to combine *Colgrove* and *Lai* to store a table containing the mapping between the encrypted addresses contained in an

inode (the data block numbers) and the corresponding addresses on disk (the mapped data block numbers). Ex. C, ¶ 127; *supra* Section VII.B.2.

5. Claim 3

a. 3[pre]: “The method of claim 1, wherein a read operation for the file comprises:”

Colgrove discloses or renders obvious the method of claim 1 for the reasons discussed in Section VII.A.2 (SNQs 1-2, claim 1). Alternatively, *Colgrove* in view of *Lai* renders obvious claim 1 for the reasons discussed in Section VII.B.3. *Colgrove* also discloses or suggests, or in view of *Lai* renders obvious, “a read operation for the file.” Ex. C, ¶¶ 128-132.

Colgrove discloses that an application may access data in the data system via data system software (e.g., file system software). Ex. PA-1, *Colgrove*, 1:67-2:4, 6:5-23, 7:7-10, 7:45-46; Ex. C, ¶ 129. *Colgrove* discloses that, once the encrypted metadata is decrypted and cached in memory, “the data system software may then access the data in the data system in accordance with the decrypted copy of the metadata.” Ex. PA-1, *Colgrove*, 11:51-58. *Colgrove* also discloses that the data system functionality of the data system software may access the data with the decrypted metadata to perform various operations (e.g., “copying”). Ex. PA-1, *Colgrove*, 6:40-56. A POSITA would have understood that accessing the stored data corresponding to a file’s inode teaches a read operation for the file. Ex. C, ¶ 129.

Additionally, a read operation for the file would have been obvious in view of *Lai*. Ex. C, ¶ 130. *Lai* discloses “reading data from the storage cells corresponding to re-mapped addresses.” Ex. PA-3, *Lai*, [0002], [0006], [0022].

A POSITA would have found it obvious to combine *Colgrove* and *Lai* to perform a read operation for a file. Ex. C, ¶ 131. A POSITA would have looked to *Lai*, for example, for implementation details on how to carry out a read operation for a file using *Colgrove*’s data

system functionality. Ex. C, ¶ 131. Moreover, a POSITA would have understood that file systems, as disclosed in *Colgrove* and *Lai*, routinely perform read operations for files and would have needed this functionality to perform the basic functions of the file system. Ex. C, ¶ 131.

A POSITA would have had a reasonable expectation of success in implementing a read operation for a file in *Colgrove*'s file system because doing so was one of the basic, well-known functions performed on a file system. Ex. C, ¶ 132. Because *Colgrove* and *Lai* both disclose file systems that implement data security by storing data in mapped addresses based on a mapping function, a POSITA would have seen *Lai*'s read operation steps as being compatible with *Colgrove* and would have expected success in implementing a read operation for *Colgrove*'s files using similar steps outlined in *Lai*. Ex. PA-1, *Colgrove*, 2:31-3:4, 7:11-8:2; Ex. PA-3, *Lai*, [0039]-[0046]; Ex. C, ¶ 132.

b. 3[a]: “obtaining the data block numbers of the file;

Colgrove in view of *Lai* renders obvious “obtaining the data block numbers of the file.” Ex. C, ¶¶ 133-138.

Colgrove discloses multiple levels of metadata in a file system, where a first level of metadata includes “a table or index to elements in the second level of metadata.” Ex. PA-1, *Colgrove*, 1:62-67, 7:28-31; Ex. C, ¶ 134. For example, *Colgrove* discloses that an inode list 320 is a first level of metadata that “include[s] information for locating Inodes 322A and 322B on a second level of the metadata.” Ex. PA-1, *Colgrove*, 3:56-60, 7:28-40. *Colgrove* discloses storing in the storage device(s) of the file system inodes where “the addresses of the file’s blocks on disk” are encrypted. Ex. PA-1, *Colgrove*, 2:57-3:4, 5:46-61; Ex. C, ¶ 134; *supra* Section VII.A.2.b. These encrypted addresses are the data block numbers of the file. Ex. C, ¶ 134; *supra* Section VII.A.2.b (SNQs 1-2, 1[a]).

Lai discloses a “procedure for reading data” that includes the address conversion module 25 obtaining the default address sequence 80 corresponding to re-mapped address sequence 85. Ex. PA-3, *Lai*, [0006], [0022], [0039]-[0041]; Ex. C, ¶ 135.

A POSITA would have found it obvious to combine *Colgrove* and *Lai* to obtain the encrypted addresses contained in *Colgrove*’s inodes during a read operation for a file for the reasons discussed in 3[pre]. Ex. C, ¶ 136; *supra* Section VII.B.5.a (3[pre]). For the embodiment where *Colgrove*’s inode list 320 is encrypted, *Colgrove* discloses the encrypted inode list 320 may be decrypted “at startup of the file system 310” and “an unencrypted copy” is cached in memory. Ex. PA-1, *Colgrove*, 7:47-60. But in *Colgrove*’s alternative embodiment where the addresses in the inode are encrypted, a POSITA would have been motivated to decrypt the encrypted addresses only upon receiving a read request corresponding to the file, as doing so would have improved performance of the file system by limiting the number of decryption operations. Ex. PA-1, *Colgrove*, 5:34-38 (disclosing “a cost in performance when encrypting or decrypting a data structure”). Ex. C, ¶ 136. A POSITA would have recognized that decrypting all addresses contained in all inodes at startup would have been an unnecessary cost in performance and would have seen decrypting the encrypted portions of the inode only upon receiving a request related to that file as the more efficient approach. Ex. C, ¶ 136.

A POSITA would have looked to *Lai* for implementation details on how to obtain the encrypted addresses in *Colgrove*’s inode during a read operation, which correspond to the addresses in *Lai*’s default address sequence that are used during *Lai*’s read operation. Ex. C, ¶ 137. Both *Colgrove* and *Lai* disclose data protection systems that requires converting metadata (e.g., addresses) from one form to another to obtain the addresses where the desired data is stored. Ex. PA-1, *Colgrove*, 7:28-8:2, 11:35-60; Ex. PA-3, *Lai*, [0039]-[0046]. Implementing

Lai's technique for obtaining default addresses in *Colgrove* to obtain the encrypted addresses in *Colgrove*'s inode would have been a simple combination of prior art elements according to known methods to predictably yield a file system that uses inodes to access corresponding stored data, as *Colgrove* teaches. Ex. C, ¶ 137. In the combination, implementing *Lai*'s technique would result in the file system obtaining the encrypted addresses contained in the inode for the requested file. Ex. C, ¶ 137.

A POSITA would have reasonably expected success for the reasons discussed in 3[pre]. Ex. C, ¶ 138; *supra* Section VII.B.5.a (3[pre]). Moreover, a POSITA would have seen the timing for decrypting the addresses as a design choice and would have reasonably expected success in decrypting the addresses in response to a read request, rather than at startup. Ex. C, ¶ 138.

c. 3[b]: “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”

Colgrove discloses or at least suggests “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed in 1[a-b] for SNQs 1-2. Ex. C, ¶ 139; *supra* Sections VII.A.2.b-VII.A.2.c.

d. 3[c]: “reading the data in the mapped data block numbers in the storage device.”

Colgrove discloses or at least suggests, or in view of *Lai* renders obvious, “reading the data in the mapped data block numbers in the storage device.” Ex. C, ¶¶ 140-143.

Colgrove discloses that the data in the file system “may represent files or portions of files” stored in blocks that “are accessible as files via the metadata for the file system 310.” Ex. PA-1, *Colgrove*, 7:31-34. *Colgrove* discloses that each inode in the metadata includes “information for locating various elements of data 314 in the file system 310.” Ex. PA-1, *Colgrove*, 7:38-40. Once the metadata is decrypted, the file system functionality may access the

data in the file system “in accordance with the decrypted copy of the metadata cached in memory” to perform various operations, including “copying.” Ex. PA-1, *Colgrove*, 1:67-2:4, 6:40-56, 7:40-46, 11:51-58. A POSITA would have understood these disclosures to teach reading the data in the mapped data block numbers in the storage device. Ex. C, ¶ 141.

Additionally, *Lai* discloses “reading data from the [storage device’s] storage cells corresponding to re-mapped addresses,” where the re-mapped addresses in the combination correspond to mapped data block numbers for the reasons discussed in 1[b]. Ex. PA-3, *Lai*, [0006], [0022], [0039]-[0042]; Ex. C, ¶ 142; *supra* Section VII.B.3.c.

Colgrove in view of *Lai* renders obvious reading the data in the mapped data block numbers in the storage device for the reasons discussed in 3[pre]-3[a]. Ex. C, ¶ 143; *supra* Sections VII.B.5.a-VII.B.5.b. In the combination, because reading the data from the mapped data block numbers follows the decryption of the encrypted addresses in the inode, it uses the mapped data block numbers and, thus, teaches reading the data in the mapped block numbers in the storage device. Ex. C, ¶ 143.

6. Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”

Colgrove discloses or suggests the additional limitations of claim 5 for the reasons discussed in connection with claim 5 in SNQs 1-2. Ex. C, ¶ 144; *supra* Section VII.A.3.

7. Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”

Colgrove discloses or suggests the additional limitations of claim 6 for the reasons discussed in connection with claim 6 in SNQs 1-2. Ex. C, ¶ 145; *supra* Section VII.A.4.

8. **Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”**

Colgrove discloses or suggests the additional limitations of claim 7 for the reasons discussed in connection with claim 7 in SNQs 1-2. Ex. C, ¶ 146; *supra* Section VII.A.5.

9. **Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”**

Colgrove discloses or suggests the additional limitations of claim 8 for the reasons discussed in connection with claim 8 in SNQs 1-2. Ex. C, ¶ 147; *supra* Section VII.A.6.

10. **Independent Claim 9**

- a. **9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”**

Colgrove discloses or suggests “[a]n apparatus for providing data security for use with a file system” for the reasons discussed in Section VII.A.7.a. Ex. C, ¶ 148.

- b. **9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”**

Colgrove discloses or at least suggests “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed in Section VII.A.7.b. Ex. C, ¶ 149. Alternatively, *Colgrove* in view of *Lai* renders obvious this limitation. Ex. C, ¶¶ 149-151.

For the reasons discussed in VII.B.3.b, *Colgrove* in view of *Lai* renders obvious applying a mapping function to the encrypted addresses contained in the inode (data block numbers that are associated with a file and contained in an inode associated with the file) to obtain the corresponding unencrypted addresses in the table. Ex. C, ¶ 150.

It would have been obvious to have *Colgrove*'s data protection mechanism perform this function. Ex. C, ¶ 151. *Colgrove* discloses that the data protection mechanism "implements data protection for the data 314 in the file system via encryption of metadata on one or more levels." Ex. PA-1, *Colgrove*, 7:40-45. Because the data protection mechanism is designed to help maintain the protection of the metadata from unauthorized access, a POSITA would have been motivated to have the data protection mechanism apply the mapping function to obtain the mapped data block numbers, as doing so would have helped maintain the table containing the mappings from unauthorized access. Ex. C, ¶ 151. At a minimum, using the data protection mechanism for this function would have been obvious to try because *Colgrove* discloses that the file system software includes a finite number of components, just the file system functionality and data protection mechanism. Ex. PA-1, *Colgrove*, 7:40-45, FIG. 4; Ex. C, ¶ 151. Using either of these components would have been obvious to try and a routine way to organize the functionality of the file system software. Ex. C, ¶ 151. And because modular organization of software functionality within different engines, like the data protection mechanism, is a best practice of developing software, a POSITA would have reasonably expected success. Ex. C, ¶ 151.

a. 9[b]: "obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device."

Colgrove discloses or at least suggests a block distribution engine to "obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device," as discussed above in Section VII.A.7.c. Ex. C, ¶ 152. Alternatively, *Colgrove* in view of *Lai* renders obvious this limitation. Ex. C, ¶¶ 152-153.

As discussed in Section VII.B.10.b, it would have been obvious for *Colgrove*'s data protection mechanism (block distribution engine) to apply a mapping function to the encrypted

addresses contained in the inode to obtain the corresponding unencrypted addresses in the table.

Ex. C, ¶ 153. The unencrypted addresses obtained after applying the mapping functions are mapped data block numbers, which are addresses of data of the file in a storage device. Ex. C, ¶ 153; *supra* Sections VII.A.2.c, VII.B.2.

11. Claim 10: “The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.”

Colgrove in view of *Lai* renders obvious “stor[ing] a mapping of the data block numbers and the mapped data block numbers” for the reasons discussed for claim 2. Ex. C, ¶ 154. For the reasons discussed in Section VII.B.2, it would have been obvious to have *Colgrove*’s data protection mechanism (block distribution engine) store the table (a mapping). Ex. C, ¶ 154. For example, the data protection mechanism encrypts “metadata on one or more levels.” Ex. PA-1, *Colgrove*, 7:40-45. Because the data protection mechanism is what would encrypt the data block numbers to create the table, a POSITA would have found it obvious to likewise use the data protection mechanism to store the table after creating it. Ex. C, ¶ 154.

12. Claim 11

a. 11[pre]: “The apparatus of claim 9, wherein for a read operation for the file,

Colgrove discloses or at least suggests the apparatus of claim 9 for the reasons discussed in Section VII.A.7 (SNQs 1-2, claim 9). Alternatively, *Colgrove* and *Lai* render obvious claim 9 as discussed above in Section VII.B.10. *Colgrove* discloses or suggests or, in view of *Lai*, renders obvious the additional limitations of 11[pre] for the reasons discussed for 3[pre]. Ex. C, ¶ 155.

b. 11[a]: “the block distribution engine obtains the data block numbers of the file from a file system, and”

Colgrove in view of *Lai* renders obvious “obtain[ing] the data block numbers of the file from a file system” for the reasons discussed for 3[a]. *Supra* Section VII.B.5.b; Ex. C, ¶ 156.

It would have been obvious to have *Colgrove*’s data protection mechanism perform this function. Ex. C, ¶ 157. *Colgrove* discloses that the data protection mechanism is what decrypts the metadata when necessary. Ex. PA-1, *Colgrove*, 4:64-5:3, 7:40-45. Because the data protection mechanism must decrypt the encrypted addresses in the inode to access the corresponding data, a POSITA would have been motivated to have the data protection mechanism obtain the encrypted addresses (the data block numbers) of the file from the file system. Ex. C, ¶ 157. Doing so would have been obvious because the data protection mechanism must obtain the encrypted data before decrypting it. Ex. C, ¶ 157. And given that organizing software functionality within different engines, like the data protection mechanism, is a routine aspect of developing software, a POSITA would have reasonably expected success. Ex. C, ¶ 157.

c. 11[b]: “obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”

Colgrove discloses or at least suggests “obtain[ing] the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed in 1[a-b] for SNQs 1-2. Ex. C, ¶ 158; *supra* Sections VII.A.2.b-VII.A.2.c. *Colgrove* discloses the data protection mechanism performing this function because it discloses that the data protection mechanism decrypts the encrypted portions of the metadata when necessary. Ex. PA-1, *Colgrove*, 4:64-5:3; Ex. C, ¶ 158.

d. 11[c]: “wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.”

Colgrove discloses or at least suggests, or in view of *Lai* renders obvious, “read[ing] the data in the mapped data block numbers in the storage device” for the reasons discussed for 3[c]. Ex. C, ¶ 159.

It would have been obvious to have a block device driver perform this function. Ex. C, ¶ 160. For example, *Colgrove* discloses that the storage device(s) containing the stored data corresponding to the mapped data block numbers “may include physical and/or logical devices” and “may be any type of computer-accessible medium capable of storing data 114,” including “magnetic or optical media.” Ex. PA-1, *Colgrove*, 1:26-46. These types of storage commonly included block device drivers to translate a data block number of the files to the physical locations on the disk. Ex. C, ¶ 160; *see* Ex. PA-6, *Mauro*, 25 (“Traditional UNIX implements kernel-resident device drivers to interface with hardware devices.”), 592-593 (disclosing that, after the file system “find[s] the location of the underlying physical blocks for the file being read,” it “calls the device driver’s strategy routine for the device containing the file system to initiate the read operation”). A POSITA would have been motivated to include a block device driver to perform this function because inodes typically included only the logical block addresses, requiring another component to then determine where on the disk that logical block address was stored. Ex. C, ¶ 160. This component typically was the block device driver. Ex. PA-6, *Mauro*, 25. Indeed, such a driver was a well-known, “conventional UNIX component[.]” Ex. PA-5, *Carter*, [0111].

13. Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”

Colgrove discloses or suggests the additional limitations of claim 13 for the reasons discussed for claim 13 in SNQs 1-2. Ex. C, ¶ 161; *supra* Section VII.A.8.

14. Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”

Colgrove discloses or suggests the additional limitations of claim 14 for the reasons discussed for claim 14 in SNQs 1-2. Ex. C, ¶ 162; *supra* Section VII.A.9.

15. Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”

Colgrove discloses or suggests the additional limitations of claim 15 for the reasons discussed for claim 15 in SNQs 1-2. Ex. C, ¶ 163; *supra* Section VII.A.10.

16. Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”

Colgrove discloses or suggests the additional limitations of claim 16 for the reasons discussed for claim 16 in SNQs 1-2. Ex. C, ¶ 164; *supra* Section VII.A.11.

C. SNQ 4: *Colgrove* in view of *Merrell* Renders Obvious Claims 1-3, 5-11, and 13-16

1. Overview of *Merrell*

Merrell “relates to the management of access to shared files managed by file systems.” Ex. PA-2, *Merrell*, 1:7-10. *Merrell* discloses a storage area network (“SAN”) connecting “mass storage systems in a communication network,” a file server managing “access to storage according to a file system architecture,” and an intermediate device processing read/write commands from clients and the file server. *Id.*, 1:22-27, 2:31-60. *Merrell* discloses that processing these commands in the intermediate device improves data security because they lack “the physical addressing parameters” of the file system’s files and, therefore, cannot bypass the file system’s “logical file management processes.” *Id.*, 2:60-64.

Figure 1, below, illustrates clients 10-12 connected to a SAN including file system server 13, intermediate device 15, and storage resources 20-22. *Id.*, 4:37-50. The file system resides in the file server or the intermediate device. *Id.*, 3:1-4.

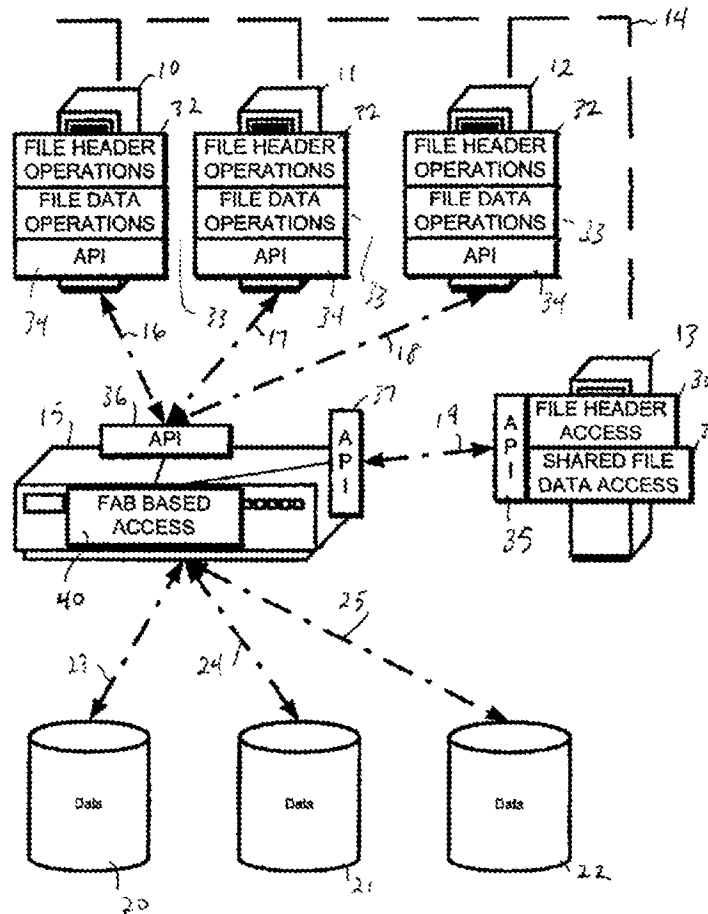


FIG. 1

Id., FIG. 1.

The clients and file server produce read and write commands that “express file access parameters using the logical addressing of the file system, such as using file access block based addressing in which an INODE defines the extent of a file.” *Id.*, 6:51-61. The intermediate device matches “the file access block based file system parameters” to “physical parameters of the storage area network.” *Id.*, 6:51-61, 7:2-8, FIG. 3. To perform this matching, the intermediate device uses “a BMAP function and tables” to find the physical address in storage corresponding

“to the logical address within a given file.” *Id.*, 7:15-18. The intermediate device obtains the inode for the requested file and “the logical offset and transfer length needed to satisfy the read or write process,” “comp[le]tes the logical to physical address transformation using this information,” and uses the resulting physical address to satisfy the read/write command. *Id.*, 7:15-29. This allows a client to make a logical request (e.g., “read file ‘a,’ offset 10 for 100 bytes”) “directly to the SAN fabric.” *Id.*, 4:6-16.

Merrell is analogous art because it is in the same field of endeavor as the ’686 patent—protecting digital information in storage devices. *Compare id.*, 1:59-2:12, 7:30-44, with Ex. A, Title, Abstract, Claims 1, 9, 17, 18; Ex. C, ¶ 64. *Merrell* is also analogous art because it is reasonably pertinent to a problem discussed in the ’686 patent because it deals with preventing users from using physical storage addresses to bypass a file system. *Compare* Ex. PA-2, *Merrell*, 1:59-2:12, 7:30-44, with Ex. A, 1:25-32, 4:34-38; Ex. C, ¶ 64.

2. Reasons to Combine *Colgrove* and *Merrell*

To the extent Patent Owner contends *Colgrove* lacks a mapping function that uses a stored table containing the mapping between the encrypted addresses in the inode and the addresses where the file’s data is stored, it would have been obvious to incorporate that functionality based on the teachings of *Merrell*. Ex. C, ¶ 166. *Colgrove* and *Merrell* are each analogous art to the ’686 patent for the reasons discussed in Sections VII.A.1 and VII.C.1.

Colgrove discloses its data system may have multiple layers of metadata, where higher layers “may be necessary to locate, access, reconstruct, and/or understand the one or more layers of metadata below the higher layer.” Ex. PA-1, *Colgrove*, 5:7-25. *Colgrove* discloses that the data protection mechanism of the file system software “implements data protection for the data 314 in the file system via encryption of metadata on one or more levels.” Ex. PA-1, *Colgrove*, 6:17-22. *Colgrove* discloses its data system software “decrypt[s] the encrypted metadata from the

data system to generate a decrypted copy of the metadata” and “cache[s] the decrypted copy of the metadata in memory.” Ex. PA-1, *Colgrove*, 11:51-54. “[T]he data system software may then access the data in the data system in accordance with the decrypted copy of the metadata cached in memory.” Ex. PA-1, *Colgrove*, 11:55-58. It would have been obvious to a POSITA in view of *Merrell* to store a mapping table as another layer of metadata below the inode and to apply a mapping function to the encrypted addresses in the inode to identify the corresponding unencrypted addresses in the table to implement *Colgrove*’s data protection techniques. Ex. C, ¶ 167.

Merrell discloses a “file system architecture” similar to *Colgrove*’s where inodes are used to identify the physical blocks of a corresponding file. Ex. PA-2, *Merrell*, 2:31-67, 7:15-29; Ex. C, ¶ 168. *Merrell* discloses that, when a client or file server produces a read or write command, the command “express[es] file access parameters using the logical addressing of the file system, such as using file access block based addressing in which an INODE defines the extent of a file.” Ex. PA-2, *Merrell*, 6:51-61. An intermediate device or switch then matches “the logical parameters of the file system to physical parameters of the storage area network” using the INODE data structure, tables, and a BMAP function, and issues a command using the resulting physical address to satisfy the read or write request. Ex. PA-2, *Merrell*, 5:49-62, 7:5-8, 7:15-29.

A POSITA would have been motivated to combine *Colgrove* and *Merrell* to store a table containing the mapping between the encrypted addresses of the file (data block numbers) and the corresponding unencrypted addresses (mapped data block numbers). Ex. C, ¶ 169. For example, *Colgrove* discloses that file system metadata may include both inodes and “mapping information in the form of indirect blocks.” Ex. PA-1, *Colgrove*, 2:42-47. The inode for a corresponding file may include “addresses of the file’s blocks on disk” as well as “pointers to indirect blocks that

reference the file blocks.” Ex. PA-1, *Colgrove*, 2:57-3:4. Because the size of the inode was limited, the pointers to indirect blocks provided a metadata hierarchy that allowed much larger file sizes, where the indirect pointers within the inode would lead to another data structure containing the indirect blocks of the file, as depicted in the diagram below. Ex. C, ¶ 169. For larger files, the vast majority of the addresses for the file’s data blocks were found within these separate data structures. Ex. C, ¶ 169; Ex. PA-6, *Mauro*, 584-585 (disclosing the inode’s direct blocks allow 12 addresses, the first level of indirection allows 2048 addresses, and the second level of indirection allows 4,194,304 addresses).

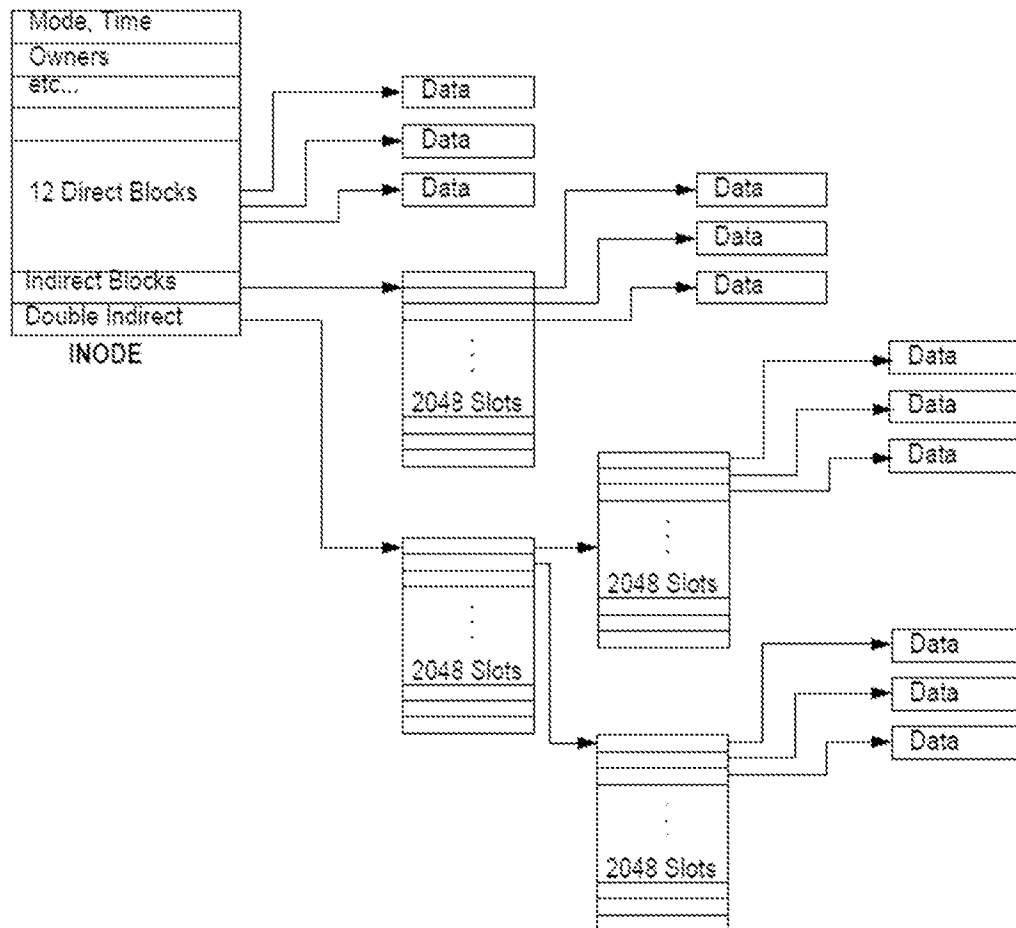


Figure 14.5 The UFS inode Format

Ex. PA-6, *Mauro*, 584.

A POSITA would have recognized that decrypting each of the addresses in both the inode and the indirect data structure(s) each time a file is accessed may impact performance of the file system. Ex. C, ¶ 170; *see* Ex. PA-1, *Colgrove*, 5:34-38 (“There may be a cost in performance when encrypting or decrypting a data structure.”). To limit the number of decryption operations needed to access a file’s addresses on disk, a POSITA would have seen the benefits of storing a table, like in *Merrell*, that contains the mapping between the encrypted addresses of the file and the corresponding unencrypted addresses. Ex. PA-3, *Lai*, claim 4 (disclosing that performing an “address conversion” using a table “is simpler than” using a mapping rule); Ex. C, ¶ 170. Just as *Merrell*’s BMAP mapping function uses tables to transform logical addresses to physical addresses based on a read or write command, a POSITA would have recognized that implementing this table would have allowed *Colgrove*’s file system software to apply a mapping function to the inode’s encrypted addresses and identify the corresponding unencrypted addresses by locating the encrypted address in the table and returning the corresponding unencrypted address. Ex. C, ¶ 170. This would have increased performance of the system because the data protection mechanism of the file system would only need to encrypt the file’s addresses once and then store the mapping between the encrypted addresses contained in the inode and the corresponding unencrypted addresses on disk in a table for later access. Ex. C, ¶ 170.

Indeed, *Colgrove* suggests such a configuration because it discloses encrypting “a metadata data structure that does not change very frequently and/or is not accessed very frequently to minimize the cost of encryption and decryption” and encrypting “only a portion of the metadata data structure,” such as a portion “that allows access to data.” Ex. PA-1, *Colgrove*, 5:34-59. A POSITA would have recognized that the address to the table would not frequently

change and that the file system software would maintain protection over the data if it encrypted the address of the table containing the mapping, similar to *Colgrove*'s alternative embodiments where higher level metadata data structure, like the inode list, is encrypted. Ex. C, ¶ 171; Ex. PA-1, *Colgrove*, 7:47-49. Doing so would have beneficially allowed the file system software to perform only one decryption operation of the table address each time the file is read. Ex. C, ¶ 171.

A POSITA would have reasonably expected success in implementing a mapping table with the encrypted addresses of the file and the corresponding unencrypted addresses. Both *Colgrove* and *Merrell* use inodes to identify the locations of stored files in a file system and relate to data security. Ex. C, ¶ 172. *Merrell* discloses that using a table and a mapping function to transform one type of address to another was well known. Ex. PA-2, *Merrell*, 7:5-8, 7:15-27; Ex. C, ¶ 172. A POSITA would have recognized that implementing the table and performing the corresponding mapping function would require only conventional software functions where the encrypted addresses of the requested file blocks are used to identify the corresponding unencrypted addresses in the table. Ex. C, ¶ 172. Combining *Colgrove* and *Merrell* in this way would have involved the use of a well-known technique to improve *Colgrove*'s file system.

3. Independent Claim 1

a. 1[pre]: “A method of providing data security for use with a file system, the method comprising:”

Colgrove discloses or suggests “[a] method of providing data security for use with a file system” for the reasons discussed in Section VII.A.2.a. Ex. C, ¶ 173.

b. 1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”

Colgrove discloses or at least suggests “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed in Section VII.A.2.b. Ex. C, ¶ 174.

Alternatively, *Colgrove* in view of *Merrell* renders obvious this limitation. Ex. C, ¶¶ 174-175

As discussed in Section VII.A.2.b, *Colgrove* discloses encrypted addresses (data block numbers) that are associated with a file and contained in an inode associated with the file. Ex. C, ¶ 175. For the reasons discussed above in Section VII.C.2, it would have been obvious to combine *Colgrove* and *Merrell* to store a table containing the mapping between the encrypted addresses contained in an inode (the data block numbers) and the corresponding addresses on disk (the mapped data block numbers). Ex. C, ¶ 175; *supra* Section VII.C.2. In this configuration, upon receiving a read request for the file, it would have been obvious for the reasons discussed in Section VII.C.2 to apply a mapping function to the encrypted addresses contained in the inode to obtain the corresponding unencrypted addresses in the table. Ex. C, ¶ 175.

c. 1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”

Colgrove discloses or at least suggests “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed in Section VII.A.2.c. Ex. C, ¶ 176.

Alternatively, *Colgrove* in view of *Merrell* renders obvious this limitation. Ex. C, ¶¶ 176-177.

As discussed in Section VII.C.3.b, it would have been obvious to apply a mapping function to the encrypted addresses contained in the inode to obtain the corresponding unencrypted

addresses in the table. Ex. C, ¶ 177. The unencrypted addresses obtained after applying the mapping functions are mapped data block numbers, which are addresses of data of the file in a storage device. Ex. C, ¶ 177; *supra* Sections VII.A.2.c, VII.C.2.

4. Claim 2: “The method of claim 1, further comprising: storing a mapping of the data block numbers and the mapped data block numbers.”

Colgrove discloses or at least suggests the method of claim 1 for the reasons discussed in Section VII.A.2 (SNQs 1-2, claim 1). Alternatively, *Colgrove* in view of *Merrell* renders obvious claim 1 for the reasons discussed in Section VII.C.3. *Colgrove* in view of *Merrell* also renders obvious “storing a mapping of the data block numbers and the mapped data block numbers.” For the reasons discussed above in Section VII.C.2, it would have been obvious to combine *Colgrove* and *Merrell* to store a table containing the mapping between the encrypted addresses contained in an inode (the data block numbers) and the corresponding addresses on disk (the mapped data block numbers). Ex. C, ¶ 178; *supra* Section VII.C.2.

5. Claim 3

a. 3[pre]: “The method of claim 1, wherein a read operation for the file comprises:”

Colgrove discloses or renders obvious the method of claim 1 for the reasons discussed in Sections VII.A.2 (SNQs 1-2, claim 1). Alternatively, *Colgrove* in view of *Merrell* renders obvious claim 1 for the reasons discussed in Section VII.C.3. *Colgrove* also discloses or suggests, or in view of *Merrell* renders obvious, “a read operation for the file.” Ex. C, ¶¶ 179-183.

Colgrove discloses that an application may access data in the data system via data system software (e.g., file system software). Ex. PA-1, *Colgrove*, 1:67-2:4, 6:5-23, 7:7-10, 7:45-46. *Colgrove* discloses that, once the encrypted metadata is decrypted and cached in memory, “the

data system software may then access the data in the data system in accordance with the decrypted copy of the metadata.” Ex. PA-1, *Colgrove*, 11:51-58. *Colgrove* also discloses that the data system functionality of the data system software may access the data with the decrypted metadata to perform various operations (e.g., “copying”). Ex. PA-1, *Colgrove*, 6:40-56. A POSITA would have understood that accessing the stored data corresponding to a file’s inode teaches a read operation for the file. Ex. C, ¶ 180.

Additionally, a read operation for the file would have been obvious in view of *Merrell*. Ex. C, ¶ 181. *Merrell* discloses a read command, including a “block read” operation, that “is produced according to a file system architecture in a client or in the file system server” and sent to an intermediate device. Ex. PA-2, *Merrell*, 3:55-61, 6:51-64. The read command may, for example, request “read file ‘a,’ offset 10 for 100 bytes.” Ex. PA-2, *Merrell*, 4:10-13. *Merrell* then discloses “resources to translate the logical request into the proper physical address of the data on the disk” and issuing a command to read the stored data corresponding to the file at the physical address. Ex. PA-2, *Merrell*, 4:14-16, 7:15-29, 7:37-44; Ex. C, ¶ 181. Figure 3, below, illustrates “the messaging process by which clients access storage of a file system.” Ex. PA-2, *Merrell*, 4:30-32.

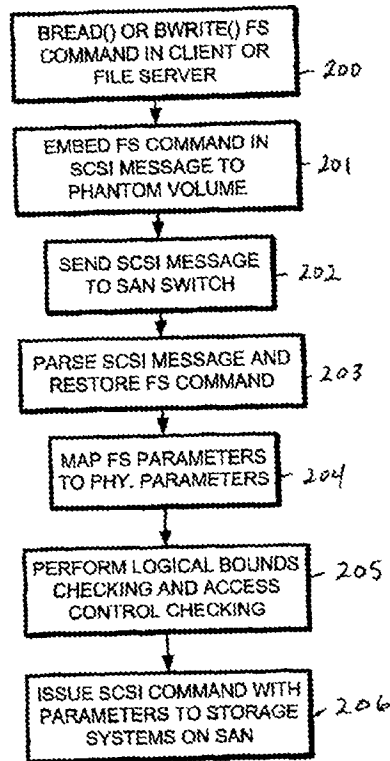


FIG. 3

Ex. PA-2, *Merrell*, FIG. 3.

A POSITA would have found it obvious to combine *Colgrove* and *Merrell* to perform a read operation for a file. Ex. C, ¶ 182. A POSITA would have looked to *Merrell*, for example, for implementation details on how to carry out a read operation for a file using *Colgrove*'s data system functionality. Ex. C, ¶ 182. Moreover, a POSITA would have understood that file systems, as disclosed in *Colgrove* and *Merrell*, routinely perform read operations for files and would have needed this functionality to perform the basic functions of the file system. Ex. C, ¶ 182.

A POSITA would have had a reasonable expectation of success in implementing a read operation for a file in *Colgrove*'s file system because doing so was one of the basic, well-known

functions performed on a file system. Ex. C, ¶ 183. Because *Colgrove* and *Merrell* both disclose file systems that organize the locations of files using inodes, a POSITA would have seen *Merrell*'s read operation steps for a file as being compatible with *Colgrove* and would have expected success in implementing a read operation using similar steps outlined in *Merrell*. Ex. PA-1, *Colgrove*, 2:31-3:4, 7:11-8:2; Ex. PA-2, *Merrell*, 6:51-7:29; Ex. C, ¶ 183.

b. 3[a]: “obtaining the data block numbers of the file;

Colgrove in view of *Merrell* renders obvious “obtaining the data block numbers of the file.” Ex. C, ¶¶ 184-189.

Colgrove discloses multiple levels of metadata in a file system, where a first level of metadata includes “a table or index to elements in the second level of metadata.” Ex. PA-1, *Colgrove*, 1:62-67, 7:28-31; Ex. C, ¶ 185. For example, *Colgrove* discloses that an inode list 320 is a first level of metadata that “include[s] information for locating Inodes 322A and 322B on a second level of the metadata.” Ex. PA-1, *Colgrove*, 3:56-60, 7:28-40. *Colgrove* discloses storing in the storage device(s) of the file system inodes where “the addresses of the file’s blocks on disk” are encrypted. Ex. PA-1, *Colgrove*, 2:57-3:4, 5:46-61; Ex. C, ¶ 185; *supra* Section VII.A.2.b. These encrypted addresses are the data block numbers of the file. Ex. C, ¶ 185; *supra* Section VII.A.2.b (SNQs 1-2, 1[a]).

Merrell discloses obtaining addresses of a file contained in an inode during a read operation for the file. Ex. C, ¶ 186. For example, when a client or file server produces a read command, that command “express[es] file access parameters using the logical addressing of the file system, such as using file access block based addressing in which an INODE defines the extent of a file.” Ex. PA-2, *Merrell*, 3:55-58, 6:51-61. The read command specifies the file using, for example, “an INODE identifier.” Ex. PA-2, *Merrell*, 7:15-20. “The INODE identifier provides an address that maps into the file system metadata,” within which “[e]ach file is

described by an INODE data structure.” Ex. PA-2, *Merrell*, 7:22-24. *Merrell*’s intermediate device invokes “a BMAP command” to obtain the INODE corresponding to the identifier, “reads the INODE data into local memory,” and uses the addresses contained in the INODE to perform “the logical to physical address transformation.” Ex. PA-2, *Merrell*, 7:15-27. Accordingly, *Merrell* discloses obtaining addresses contained in an inode during a read operation for the file, for example, when it reads the inode data into memory. Ex. C, ¶ 186.

A POSITA would have found it obvious to combine *Colgrove* and *Merrell* to obtain the encrypted addresses contained in *Colgrove*’s inodes during a read operation for a file for the reasons discussed in 3[pre]. Ex. C, ¶ 187; *supra* Section VII.C.5.a (3[pre]). For the embodiment where *Colgrove*’s inode list 320 is encrypted, *Colgrove* discloses the encrypted inode list 320 may be decrypted “at startup of the file system 310” and “an unencrypted copy” is cached in memory. Ex. PA-1, *Colgrove*, 7:47-60. But in *Colgrove*’s alternative embodiment where the addresses in the inode are encrypted, a POSITA would have been motivated to decrypt the encrypted addresses only upon receiving a read request corresponding to the file, as doing so would have improved performance of the file system by limiting the number of decryption operations. Ex. PA-1, *Colgrove*, 5:34-38 (disclosing “a cost in performance when encrypting or decrypting a data structure”). Ex. C, ¶ 187. A POSITA would have recognized that decrypting all addresses contained in all inodes at startup would have been an unnecessary cost in performance and would have seen decrypting the encrypted portions of the inode only upon receiving a request related to that file as the more efficient approach. Ex. C, ¶ 187.

A POSITA would have looked to *Merrell* for implementation details on how to obtain the data contained in an inode during a read operation. Ex. C, ¶ 188. Both *Colgrove* and *Merrell* disclose file systems with multiple levels of metadata and use inodes to access files stored on the

file system. Ex. PA-1, *Colgrove*, 7:28-8:2, 11:35-60; Ex. PA-2, *Merrell*, 15-29. Implementing *Merrell*'s technique for obtaining the addresses contained in an inode in *Colgrove* would have been a simple combination of prior art elements according to known methods to predictably yield a file system that uses inodes to access corresponding stored data, as *Colgrove* teaches. Ex. C, ¶ 188. In the combination, implementing *Merrell*'s technique would result in the file system obtaining the encrypted addresses contained in the inode for the requested file. Ex. C, ¶ 188.

A POSITA would have reasonably expected success for the reasons discussed in 3[pre]. Ex. C, ¶ 189; *supra* Section VII.C.5.a (3[pre]). Moreover, a POSITA would have seen the timing for decrypting the addresses as a design choice and would have reasonably expected success in decrypting the addresses in response to a read request, rather than at startup. Ex. C, ¶ 189.

c. 3[b]: “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”

Colgrove discloses or at least suggests “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed in 1[a-b] for SNQs 1-2. Ex. C, ¶ 190; *supra* Sections VII.A.2.b-VII.A.2.c.

d. 3[c]: “reading the data in the mapped data block numbers in the storage device.”

Colgrove discloses or at least suggests, or in view of *Merrell* renders obvious, “reading the data in the mapped data block numbers in the storage device.” Ex. C, ¶ 191.

Colgrove discloses that the data in the file system “may represent files or portions of files” stored in blocks that “are accessible as files via the metadata for the file system 310.” Ex. PA-1, *Colgrove*, 7:31-34. *Colgrove* discloses that each inode in the metadata includes “information for locating various elements of data 314 in the file system 310.” Ex. PA-1, *Colgrove*, 7:38-40. Once the metadata is decrypted, the file system functionality may access the

data in the file system “in accordance with the decrypted copy of the metadata cached in memory” to perform various operations, including “copying.” Ex. PA-1, *Colgrove*, 1:67-2:4, 6:40-56, 7:40-46, 11:51-58. A POSITA would have understood these disclosures to teach reading the data in the mapped data block numbers in the storage device. Ex. C, ¶ 192.

Additionally, *Merrell* discloses that, after the intermediate device obtains the physical addresses of the file for the read command, it issues “[a] standard SCSI command” using the obtained physical addresses “on the storage area network to satisfy the file system request.” Ex. PA-2, *Merrell*, 6:51-59, 7:15-29. A POSITA would have understood that issuing an SCSI command using the data block numbers in the storage device to satisfy the file system read request discloses reading the data in the data block numbers in the storage device. Ex. C, ¶ 193.

Colgrove in view of *Merrell* renders obvious reading the data in the mapped data block numbers in the storage device for the reasons discussed in 3[pre]-3[a]. Ex. C, ¶ 194; *supra* Sections VII.C.5.a-VII.C.5.b. In the combination, because the SCSI command follows the decryption of the encrypted addresses in the inode, it uses the mapped data block numbers and, thus, teaches reading the data in the mapped block numbers in the storage device. Ex. C, ¶ 194.

6. Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”

Colgrove discloses or suggests the additional limitations of claim 5 for the reasons discussed in connection with claim 5 in SNQs 1-2. Ex. C, ¶ 195; *supra* Section VII.A.3.

7. Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”

Colgrove discloses or suggests the additional limitations of claim 6 for the reasons discussed in connection with claim 6 in SNQs 1-2. Ex. C, ¶ 196; *supra* Section VII.A.4.

8. **Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”**

Colgrove discloses or suggests the additional limitations of claim 7 for the reasons discussed in connection with claim 7 in SNQs 1-2. Ex. C, ¶ 197; *supra* Section VII.A.5.

9. **Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”**

Colgrove discloses or suggests the additional limitations of claim 8 for the reasons discussed in connection with claim 8 in SNQs 1-2. Ex. C, ¶ 198; *supra* Section VII.A.6.

10. **Independent Claim 9**

- a. **9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”**

Colgrove discloses or suggests “[a]n apparatus for providing data security for use with a file system” for the reasons discussed in Section VII.A.7.a. Ex. C, ¶ 199.

- b. **9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”**

Colgrove discloses or at least suggests “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed in Section VII.A.7.b. Ex. C, ¶ 200. Alternatively, *Colgrove* in view of *Merrell* renders obvious this limitation. Ex. C, ¶¶ 200-202.

For the reasons discussed in VII.C.3.b, *Colgrove* in view of *Merrell* renders obvious applying a mapping function to the encrypted addresses contained in the inode (data block numbers that are associated with a file and contained in an inode associated with the file) to obtain the corresponding unencrypted addresses in the table. Ex. C, ¶ 201.

It would have been obvious to have *Colgrove*'s data protection mechanism perform this function. Ex. C, ¶ 202. *Colgrove* discloses that the data protection mechanism "implements data protection for the data 314 in the file system via encryption of metadata on one or more levels." Ex. PA-1, *Colgrove*, 7:40-45. Because the data protection mechanism is designed to help maintain the protection of the metadata from unauthorized access, a POSITA would have been motivated to have the data protection mechanism apply the mapping function to obtain the mapped data block numbers, as doing so would have helped maintain the table containing the mappings from unauthorized access. Ex. C, ¶ 202. At a minimum, using the data protection mechanism for this function would have been obvious to try because *Colgrove* discloses that the file system software includes a finite number of components, just the file system functionality and data protection mechanism. Ex. PA-1, *Colgrove*, 7:40-45, FIG. 4. Using either of these components would have been obvious to try and a routine way to organize the functionality of the file system software. Ex. C, ¶ 202. And given that organizing software functionality within different engines, like the data protection mechanism, is a routine aspect of developing software, a POSITA would have reasonably expected success. Ex. C, ¶ 202.

c. 9[b]: "obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device."

Colgrove discloses or at least suggests a block distribution engine to "obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device" as discussed above in Section VII.A.7.c. Ex. C, ¶ 203. Alternatively, *Colgrove* in view of *Merrell* renders obvious this limitation. Ex. C, ¶¶ 203-204.

As discussed in Section VII.C.10.b, it would have been obvious for *Colgrove*'s data protection mechanism (block distribution engine) to apply a mapping function to the encrypted

addresses contained in the inode to obtain the corresponding unencrypted addresses in the table.

Ex. C, ¶ 204. The unencrypted addresses obtained after applying the mapping functions are mapped data block numbers, which are addresses of data of the file in a storage device. Ex. C, ¶ 204; *supra* Sections VII.A.2.c, VII.C.2.

11. Claim 10: “The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.”

Colgrove in view of *Merrell* renders obvious “stor[ing] a mapping of the data block numbers and the mapped data block numbers” for the reasons discussed for claim 2. Ex. C, ¶ 205. For the reasons discussed in Section VII.C.2, it would have been obvious to have *Colgrove*’s data protection mechanism (block distribution engine) store the table (a mapping). Ex. C, ¶ 206. For example, the data protection mechanism encrypts “metadata on one or more levels.” Ex. PA-1, *Colgrove*, 7:40-45. Because the data protection mechanism is what would encrypt the data block numbers to create the table, a POSITA would have found it obvious to likewise use the data protection mechanism to store the table after creating it. Ex. C, ¶ 206.

12. Claim 11

a. 11[pre]: “The apparatus of claim 9, wherein for a read operation for the file,

Colgrove discloses or at least suggests the apparatus of claim 9 for the reasons discussed in Section VII.A.7 (SNQs 1-2, claim 9). Alternatively, *Colgrove* and *Merrell* render obvious claim 9 as discussed above in Section VII.C.10. *Colgrove* discloses or suggests or, in view of *Merrell*, renders obvious the additional limitations of 11[pre] for the reasons discussed for 3[pre]. Ex. C, ¶ 207.

b. 11[a]: “the block distribution engine obtains the data block numbers of the file from a file system, and”

Colgrove in view of *Merrell* renders obvious “obtain[ing] the data block numbers of the file from a file system” for the reasons discussed for 3[a]. *Supra* Section VII.C.5.b; Ex. C, ¶ 208.

It would have been obvious to have *Colgrove*’s data protection mechanism perform this function. Ex. C, ¶ 209. *Colgrove* discloses that the data protection mechanism is what decrypts the metadata when necessary. Ex. PA-1, *Colgrove*, 4:64-5:3, 7:40-45. Because the data protection mechanism must decrypt the encrypted addresses in the inode to access the corresponding data, a POSITA would have been motivated to have the data protection mechanism obtain the encrypted addresses (the data block numbers) of the file from the file system. Ex. C, ¶ 209. Doing so would have been obvious because the data protection mechanism must obtain the encrypted data before decrypting it. Ex. C, ¶ 209. And given that organizing software functionality within different engines, like the data protection mechanism, is a routine aspect of developing software, a POSITA would have reasonably expected success. Ex. C, ¶ 209.

c. 11[b]: “obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”

Colgrove discloses or at least suggests “obtain[ing] the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed in 1[a-b] for SNQs 1-2. Ex. C, ¶ 210; *supra* Sections VII.A.2.b-VII.A.2.c. *Colgrove* discloses the data protection mechanism performing this function because it discloses that the data protection mechanism decrypts the encrypted portions of the metadata when necessary. Ex. PA-1, *Colgrove*, 4:64-5:3; Ex. C, ¶ 210.

d. 11[c]: “wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.”

Colgrove discloses or at least suggests, or in view of *Merrell* renders obvious, “read[ing] the data in the mapped data block numbers in the storage device” for the reasons discussed for 3[c]. Ex. C, ¶ 211.

It would have been obvious to have a block device driver perform this function. Ex. C, ¶ 212. For example, *Colgrove* discloses that the storage device(s) containing the stored data corresponding to the mapped data block numbers “may include physical and/or logical devices” and “may be any type of computer-accessible medium capable of storing data 114,” including “magnetic or optical media.” Ex. PA-1, *Colgrove*, 1:26-46. These types of storage commonly included block device drivers to translate a data block number of the files to the physical locations on the disk. Ex. C, ¶ 212; *see* Ex. PA-6, *Mauro*, 25 (“Traditional UNIX implements kernel-resident device drivers to interface with hardware devices.”), 592-593 (disclosing that, after the file system “find[s] the location of the underlying physical blocks for the file being read,” it “calls the device driver’s strategy routine for the device containing the file system to initiate the read operation”). A POSITA would have been motivated to include a block device driver to perform this function because inodes typically included only the logical block addresses, requiring another component to then determine where on the physical disk the logical block address was stored. Ex. C, ¶ 212. This component typically was the block device driver. Ex. PA-6, *Mauro*, 25. Indeed, such a driver was a well-known, “conventional UNIX component[.]” Ex. PA-5, *Carter*, [0111]; Ex. C, ¶ 212.

13. Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”

Colgrove discloses or suggests the additional limitations of claim 13 for the reasons discussed for claim 13 in SNQs 1-2. Ex. C, ¶ 213; *supra* Section VII.A.8.

14. Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”

Colgrove discloses or suggests the additional limitations of claim 14 for the reasons discussed for claim 14 in SNQs 1-2. Ex. C, ¶ 213; *supra* Section VII.A.9.

15. Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”

Colgrove discloses or suggests the additional limitations of claim 15 for the reasons discussed for claim 15 in SNQs 1-2. Ex. C, ¶ 213; *supra* Section VII.A.10.

16. Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”

Colgrove discloses or suggests the additional limitations of claim 16 for the reasons discussed for claim 16 in SNQs 1-2. Ex. C, ¶ 213; *supra* Section VII.A.11.

D. SNQ 5: *Lai* in View of *Card* Renders Obvious Claims 1-3, 5-11, and 13-18

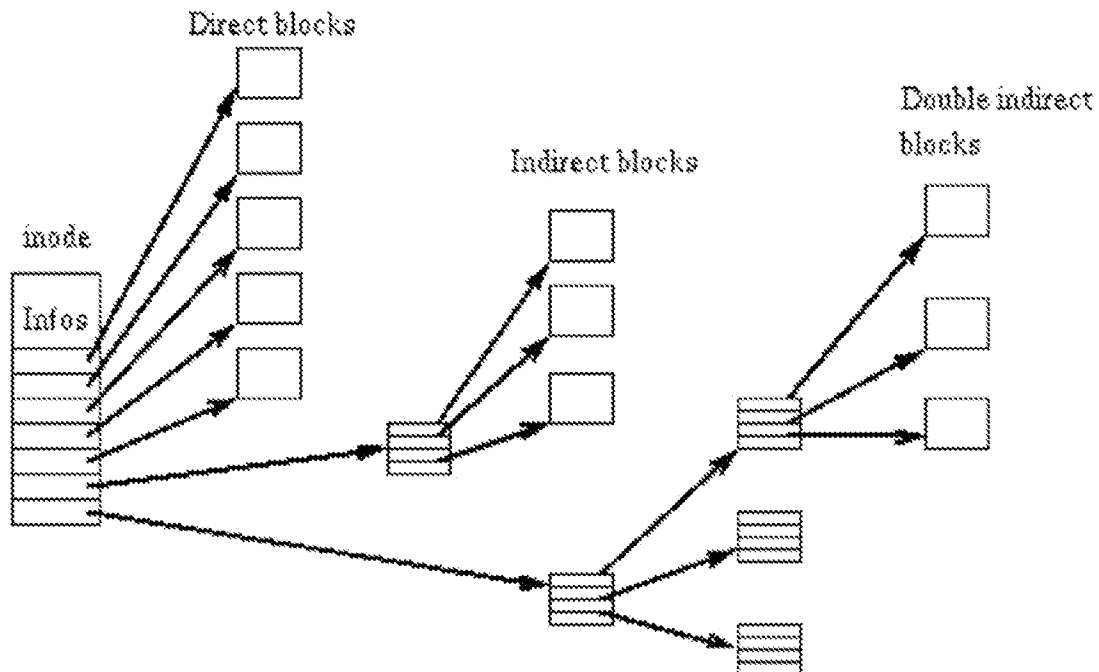
1. Overview of *Card*

Card is prior art to the '686 patent because it was published and publicly accessible to persons interested and ordinarily skilled in the art before July 31, 2007, via MIT's website and on linuxdoc.org. Ex. D. The fact that persons interested and ordinarily skilled in the art could find *Card* through these websites is further supported by the fact that it was, in fact, accessed by such persons because it was submitted to the USPTO during prosecution of two applications involving similar technology before July 31, 2007. Ex. E, 97-98 (identifying *Card* on July 7, 2000), 377-390 (*Card*); Ex. F, 357-367 (*Card*), 412-413 (identifying *Card* on May 3, 2007). The file histories of these applications show that persons interested and ordinarily skilled in the art accessed *Card* through the MIT website and submitted copies to the PTO. *Card* was further publicly accessible upon issuance of U.S. Patent No. 6,377,958 on April 23, 2002, which

identifies *Card* as a publication on the face of the patent and would have allowed persons interested and ordinarily skilled in the art to access *Card* through the patent's file history.

Ex. E, 377-390.

Card is entitled "Design and Implementation of the Second Extended Filesystem" and relates to the history and implementation of the efs2 filesystem that was widely used with the Linux operating system. *See generally* Ex. PA-4, *Card*; Ex. C, ¶ 69. *Card* summarizes the basic concepts of Linux filesystems, including their inodes. Ex. PA-4, *Card*, 2-3. Each file in the filesystem is represented by an inode containing information about the file, including "[t]he addresses of data blocks allocated to a file." Ex. PA-4, *Card*, 2. "When a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table[,] and reads or writes the physical block." Ex. PA-4, *Card*, 2. The structure of an inode is depicted below.



Ex. PA-4, *Card*, 2.

Card teaches additional implementation details, including allocating disk blocks to files for storage on disk; deallocating blocks to free them; and creating links between a file's filename and inode number, and thus the corresponding data block numbers. Ex. PA-4, *Card*, 2-5; Ex. C, ¶ 70.

Card is analogous art at least because it is in the same field of endeavor as the '686 patent—protecting digital information in storage devices. Ex. PA-4, *Card*, Title, 4 (“prevent[ing] malicious people from gaining access” to a deleted file's content); Ex. C, ¶ 71. *Card* is also analogous art because it is reasonably pertinent to the particular problem of the '686 patent because the '686 patent deals with retrieving “data blocks of a file by accessing” its inode, and *Card* discloses inodes and how they are used to retrieve a file's data blocks. Compare Ex. PA-4, *Card*, 2-3, with Ex. A, 1:29-32, 2:15-61; Ex. C, ¶ 71.

2. Reasons to Combine *Lai* and *Card*

A POSITA would have found it obvious to combine *Lai* and *Card* to protect data within a file system by applying *Lai*'s re-mapping rule to addresses associated with a file and contained in *Card*'s inodes. Ex. C, ¶ 218. *Lai* and *Card* are each analogous art to the '686 patent for the reasons discussed in Sections VII.B.1 and VII.D.1.

Lai discloses “[a] data protection system” that “protect[s] data stored on storage devices or media by changing the mapping between the physical position and the operating system acknowledged position of storage cells.” Ex. PA-3, *Lai*, Abstract. After applying a mapping function (re-mapping rule) to change the “sequence of addresses originally designated by the base computer system while accessing data within the access domain,” the data is written to and read from “re-mapped addresses instead of system-designated addresses.” Ex. PA-3, *Lai*, [0006], [0031]-[0046]; Ex. C, ¶ 219. This protects the data (e.g., “documents, technical data, [and] confidential data”) because “computers without the data protection system . . . cannot read the

correct data out of the protected zone of the storage devices or media.” Ex. PA-3, *Lai*, [0004]-[0005].

While *Lai* discloses that the default address sequence for data to be protected is “designated by the base computer system” and acknowledged by the operating system, *Lai* does not explicitly describe how the base computer/operating system organizes the default address sequences for files stored on the associated storage devices or media. Ex. PA-3, *Lai*, Abstract, [0006], [0036], [0040]. A POSITA would have looked to *Card* for implementation details of a filesystem. Ex. C, ¶ 220.

Card teaches how file systems organize and manage the storage of files on disk. Ex. PA-4, *Card*, 2-5; Ex. C, ¶ 221. *Card* discloses that “[e]very Linux filesystem” represents files using inodes, each of which contains “addresses of data blocks allocated to a [particular] file.” Ex. PA-4, *Card*, 2. When the system receives a read/write request on a file, it converts the request to a block number, “uses this number as an index in the block addresses table” of the inode, and “reads or writes the physical block.” Ex. PA-4, *Card*, 2. *Card* also discloses a directory that “contain[s] a list of entries,” each of which “contains an inode number and a file name.” Ex. PA-4, *Card*, 2-3.

inode number	entry length	name length	filename
--------------	--------------	-------------	----------

Ex. PA-4, *Card*, 5. “[T]he inode number points to the inode” that in turn contains the data block numbers. Ex. PA-4, *Card*, 3; Ex. C, ¶ 221. “When a process uses a pathname, the kernel code search[es] in the directories to find the corresponding inode number. After the name has been converted to an inode number, the inode is loaded into memory and is used by subsequent requests.” Ex. PA-4, *Card*, 3. *Card* discloses that, “[w]hen writing data to a file, Ext2fs preallocates up to 8 adjacent blocks when allocating a new block,” that “the kernel code always

tries to allocate data blocks for a file in the same group as its inode,” and that allocation routines “allow user mode programs to allocate and free blocks and inodes.” Ex. PA-4, *Card*, 5.

A POSITA would have been motivated to look to and incorporate these common file-system concepts from *Card* into the teachings of *Lai*. Ex. C, ¶ 222. *Lai* provides data-protection techniques applicable to the storage of digital information generally, and *Card* teaches details about how well-known file systems organize metadata and allocate storage to files in a file system. Ex. C, ¶ 222. Combining *Lai* and *Card* would have been a combination of familiar elements according to known methods to predictably yield a filesystem with a typical metadata hierarchy used to efficiently read and write files and improved protection for some or all of the files in the filesystem. Ex. C, ¶ 222.

A POSITA would have seen the teachings of *Lai* and *Card* as complementary. Ex. C, ¶ 223. *Card* discloses concepts used in every Linux file system. Ex. PA-4, *Card*, 2. A POSITA would have been motivated to use *Card*’s filesystem and corresponding metadata hierarchy because it would have allowed ease of implementation within then-existing operation systems. Ex. C, ¶ 223. But these concepts had a known potential security vulnerability where, “[o]nce an unauthorized user gained access to the directory, he could access the i-nodes to recover the data associated with the file.” Ex. PA-5, *Carter*, [0218]. This is because the inodes contained “the locations of the data blocks that contain the data associated with the file.” Ex. PA-5, *Carter*, [0218]. A POSITA would have seen *Lai*’s data protection mechanism as solving this known potential security vulnerability because, when combined, *Lai*’s address conversion module 25 would convert the addresses associated with a file and contained in *Card*’s inode to a re-mapped address sequence where the data is physically stored. Ex. C, ¶ 223. This would have maintained *Lai*’s objective of preventing unauthorized access to the data because computers that accessed

the inode and lacked a data protection mechanism with the correct re-mapping mechanism still could not access the data associated with the file. Ex. PA-3, *Lai*, [0005].

A POSITA would have reasonably expected success in incorporating *Card*'s filesystem, including inodes that each contained addresses associated with a file, to organize the default address sequences of files designated by *Lai*'s base computer because they involved implementation details of well-known filesystems, which were in widespread use before the earliest claimed priority date of the '686 patent. Ex. C, ¶ 224. As such, a POSITA would have expected combining *Lai* and *Card* to require only conventional techniques compatible with every Linux file system and ubiquitous at the time. Ex. C, ¶ 224. A POSITA would have seen a file system that used inodes to organize the addresses of files on a computer, as taught by *Card*, as a well-known and predictable way for the operating system to read and write data on storage devices, as taught by *Lai*. Ex. C, ¶ 224.

3. Independent Claim 1:

a. 1[pre]: "A method of providing data security for use with a file system, the method comprising:"

If the preamble is limiting, *Lai* in view of *Card* renders obvious "[a] method of providing data security for use with a file system." Ex. C, ¶¶ 225-227.

Lai discloses the "protection of data on storage devices" and that such data is accessed by an "operating system." Ex. PA-3, *Lai*, Title, Abstract, [0002], [0005]-[0006], [0047]. *Card* discloses a file system that enables an operating system to store and retrieve files and their constituent data on a storage device through the use of metadata (e.g., inodes) corresponding to the files on the file system. Ex. PA-4, *Card*, 2, 4; Ex. C, ¶ 226.

For the reasons discussed in Section VII.D.2, it would have been obvious to combine the data-security techniques of *Lai* with *Card*'s file system to provide additional security when

storing sensitive files, as taught by *Lai*. Ex. PA-3, *Lai*, [0004]-[0006]; Ex. PA-4, *Card*, 2-3; Ex. C, ¶ 227. A POSITA would have understood that operating systems, as in *Lai*, commonly used file systems like *Card*'s to store and manage files on a storage device. Ex. PA-3, *Lai*, Abstract; Ex. PA-4, *Card*, 2-3 (describing concepts implemented in “[e]very Linux filesystem”); Ex. C, ¶ 227.

b. 1[a]: “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”

Lai in view of *Card* renders obvious “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file.” Ex. C, ¶¶ 228-232.

Card discloses “an index node associated with said file” because it discloses that “[e]ach file is represented by a structure, called an inode” (i.e., “index node”). Ex. PA-4, *Card*, 2; Ex. C, ¶ 229. Each inode (i.e., “index node”) stores the “addresses of data blocks allocated to” the corresponding file. Ex. PA-4, *Card*, 2.

Lai discloses “applying a mapping function to data block numbers” because it discloses an access domain address conversion module 25 using “re-mapping table 65” or “address conversion rule 60 to convert the access domain default address sequence 80 . . . designated by the base computer system to the access domain re-mapped address sequence 85” Ex. PA-3, *Lai*, [0031], [0036]-[0037], [0039]-[0041], Figs. 6-7. The address conversion rule 60, which is used to create re-mapping table 65 depicted below, is “a function of the range of the protected zone address” and may be, for example $f(x)=1000-x$, where the protected zone includes addresses ranging from 0-1000, and x is the address to be mapped. Ex. PA-3, *Lai*, [0024]-[0027], FIGs. 1-4; Ex. C, ¶ 230.

default address	0	1	2	3	4	5	6	7	...	996	997	998	999	1000	↖ 70
re-mapped address	1000	999	998	997	996	995	994	993	...	4	3	2	1	0	↖ 75

FIG. 4

Ex. PA-3, *Lai*, FIG. 4.

Applying the mapping function to access domain default address sequence [1, 2, 4, 6, 7, 996] results in re-mapped address sequence [999, 998, 996, 994, 993, 4], as shown below. Ex. PA-3, *Lai*, [0037], [0041]; Ex. C, ¶ 231.

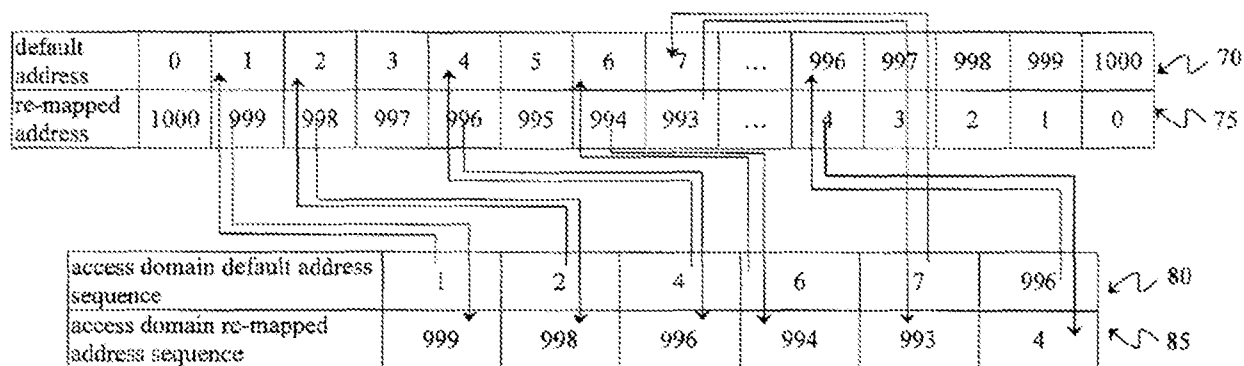


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

For the reasons discussed in Section VII.D.2, it would have been obvious to combine *Lai* and *Card* such that the addresses to which *Lai*'s mapping function is applied are addresses associated with a file and contained in *Card*'s inodes. Ex. C, ¶ 232. In this combination, *Card*'s disclosure of "addresses of data blocks allocated to a file" that are contained in each inode are "data block numbers" that "are associated with a file." Ex. PA-4, *Card*, 2; Ex. C, ¶ 232.

Applying *Lai*'s mapping function to these data block numbers creates re-mapped addresses of data blocks corresponding to where the file's data is stored within the storage device. Ex. PA-3, *Lai*, [0036]-[0038], [0040]-[0042].

- c. **1[b]: “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Lai in view of *Card* renders obvious “obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.” Ex. C, ¶¶ 233-235.

Lai discloses “obtaining mapped data block numbers after applying the mapping function” because it discloses access domain address conversion module 25 that obtains a re-mapped address sequence (“mapped data block numbers”) after applying the mapping function (e.g., address conversion rule 60 or re-mapping table 65) to the default address sequence. Ex. PA-3, *Lai*, [0036]-[0037], [0040]-[0041]; Ex. C, ¶ 234. This conversion is depicted in Figure 7, below. Ex. PA-3, *Lai*, [0013], [0037]. The re-mapped address sequence is a sequence of addresses where the data (e.g., “documents, technical data, [or] confidential data”) is stored in the storage device. Ex. PA-3, *Lai*, [0004], [0038], [0042]; Ex. C, ¶ 234.

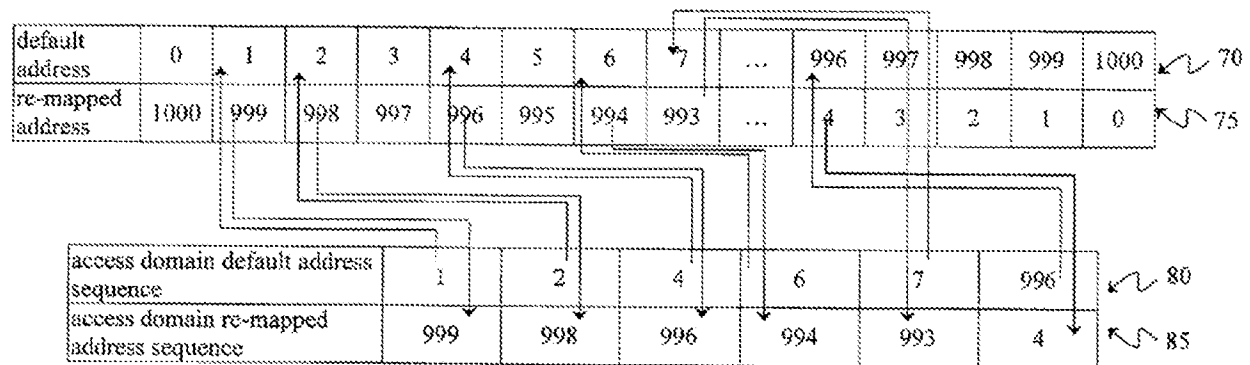


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

As discussed in Section VII.D.3.b (SNQ 5, 1[a]), it would have been obvious to combine *Lai* and *Card* to apply *Lai*’s mapping function to the addresses of data blocks allocated to a file in *Card*’s inodes. Ex. C, ¶ 235. In this combination, after applying *Lai*’s mapping function, *Lai*’s

access domain address conversion module 25 obtains re-mapped addresses of data blocks that are the “addresses of data of the file in a storage device.” Ex. PA-3, *Lai*, [0036]-[0038], [0040]-[0042] (data “is stored to the storage cells corresponding to the access domain re-mapped address sequence 85”); Ex. C, ¶ 235.

4. Claim 2: “The method of claim 1, further comprising: storing a mapping of the data block numbers and the mapped data block numbers.”

Lai in view of *Card* renders obvious the additional limitations of claim 2. Ex. C, ¶¶ 236-239.

For example, *Lai* teaches “a protected zone address re-mapping table 65” is created using address re-mapping rule 60 and is “used for look-up to convert” a default address sequence 70 to the protected zone re-mapped address sequence 75. Ex. PA-3, *Lai*, [0006], [0010]-[0011], [0017]-[0018], [0024], [0037], [0041], claim 3, Figs. 1-5 & 7; Ex. C, ¶ 237. *Lai* explains that “FIG. 4 is a protected zone address re-mapping table setup using a sample address re-mapping rule.” Ex. PA-3, *Lai*, [0010].

default address	0	1	2	3	4	5	6	7	...	996	997	998	999	1000	↙ 70
re-mapped address	1000	999	998	997	996	995	994	993	...	4	3	2	1	0	↘ 75

FIG. 4

Id., [0010], Fig. 4.

Lai’s address re-mapping table 65 is stored because *Lai* states that it is “used for look-up.” *Id.*, [0006]; Ex. C, ¶ 238. For example, each time data is read or written, *Lai*’s address conversion module 25 may “use the protected zone address re-mapping table 65” to convert the default address sequence to a re-mapped address sequence, meaning that it is stored for use each time an address conversion is needed. Ex. PA-3, *Lai*, [0036]-[0037], [0040]-[0041].

As discussed in Section VII.D.3.b-VII.D.3.c (SNQ 5, 1[a]), it would have been obvious to combine *Lai* and *Card* to apply *Lai*'s mapping function to data block numbers and obtain mapped data block numbers. Ex. C, ¶ 239. In the combination, address re-mapping table 65 is a mapping of the data block numbers and the mapped data block numbers. Ex. C, ¶ 239. As depicted in Figure 7, below, *Lai* uses re-mapping table 65 to convert those data block numbers [1, 2, 4, 6, 7, 996] to the mapped data block numbers [999, 998, 996, 994, 993, 4]. Ex. PA-3, *Lai*, [0013], [0036]-[0037], [0041]-[0042].

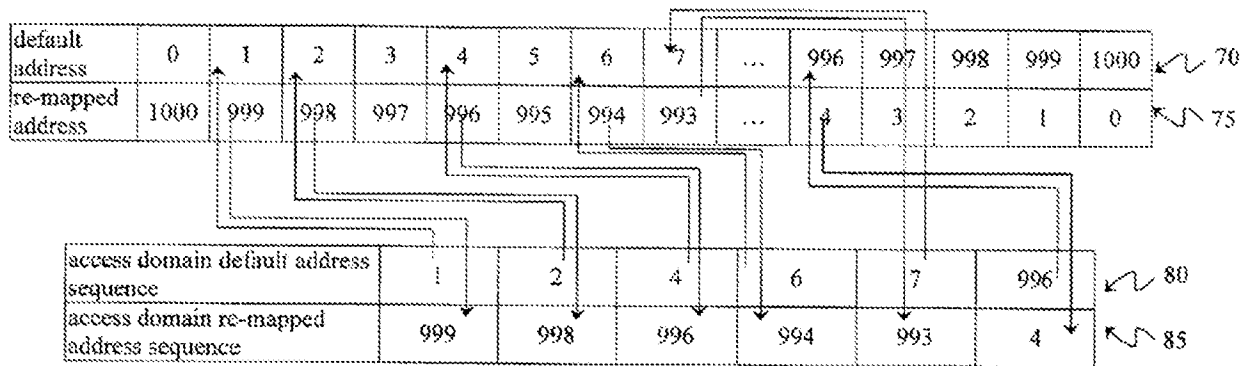


FIG. 7

Ex. PA-3, *Lai*, FIG. 7.

5. Claim 3

- a. 3[pre]: “The method of claim 1, wherein a read operation for the file comprises:”

Lai in view of *Card* discloses “wherein a read operation for the file comprises. Ex. C, ¶¶ 240-241.

Lai discloses “reading data from the storage cells corresponding to re-mapped addresses.” Ex. PA-3, *Lai*, [0002], [0006], [0022]. As discussed for claim 1, in the *Lai-Card* combination, the data corresponds to a file in the file system. *Supra* Section VII.D.3; Ex. C, ¶ 241.

b. 3[a]: “obtaining the data block numbers of the file;

Lai in view of *Card* discloses “obtaining the data block numbers of the file.” Ex. C, ¶¶ 242-243.

Lai discloses a “procedure for reading data” that includes the address conversion module 25 obtaining the default address sequence 80 corresponding to re-mapped address sequence 85. Ex. PA-3, *Lai*, [0006], [0022], [0039]-[0041]. As discussed for 1[a], in the *Lai-Card* combination, the default address sequence 80 corresponds to the data block numbers of the file. *Supra* Section VII.A.2.b; Ex. C, ¶ 243. Address conversion module 25 would obtain the data block numbers of the file from the filesystem to do the conversion, for example, according to *Card*’s procedure for loading the inode into memory. Ex. PA-4, *Card*, 2-3; Ex. C, ¶ 243; *supra* Section VII.D.2.

c. 3[b]: “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”

Lai in view of *Card* renders obvious “obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed in 1[a-b] for SNQ 5. Ex. C, ¶ 244; *supra* Sections VII.D.3.b-VII.D.3.c.

d. 3[c]: “reading the data in the mapped data block numbers in the storage device.”

Lai in view of *Card* discloses or at least suggests “reading the data in the mapped data block numbers in the storage device.” Ex. C, ¶¶ 245-246.

Lai discloses “reading data from the [storage device’s] storage cells corresponding to re-mapped addresses,” where the re-mapped addresses are mapped data block numbers for the reasons discussed in 1[b] for SNQ 5. Ex. PA-3, *Lai*, [0006], [0022], [0039]-[0042]; Ex. C, ¶ 246; *supra* Section VII.D.3.c.

6. Claim 5: “The method of claim 1, wherein the mapping function is an equation that is applied to each data block number of the file.”

Lai in view of *Card* renders obvious “wherein the mapping function is an equation that is applied to each data block number of the file.” Ex. C, ¶¶ 247-248.

For the reasons discussed for 1[a], *Lai* in view of *Card* renders obvious applying a mapping function to each data block number of the file because it discloses applying “address conversion rule 60” to convert the sequence of default addresses (each data block number of the file) to a re-mapped address sequence. Ex. PA-3, *Lai*, Title, Abstract, [0002], [0005]-[0011], [0016]-[0030], [0036]-[0037], [0039]-[0040], Figs. 1-5. *Lai* teaches that the re-mapping rule is an equation and provides two examples:

$$f(x)=1000-x$$

(Corresponding to Figure 4)

$$f(x) = \begin{array}{ll} 96 - x & \text{if } 0 \leq x < 97 \\ 145 - x + 97 & \text{if } 97 \leq x < 146 \\ 220 - x + 146 & \text{if } 146 \leq x < 221 \\ 277 - x + 221 & \text{if } 221 \leq x < 278 \\ 405 - x + 278 & \text{if } 278 \leq x < 406 \\ 499 - x + 406 & \text{if } 406 \leq x < 500 \end{array}$$

(Corresponding to Figure 5)

Ex. PA-3, *Lai*, [0024]-[0030], Figs. 4-5; Ex. C, ¶ 248.

7. Claim 6: “The method of claim 1, further comprising: selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.”

Lai in view of *Card* renders obvious “selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.” Ex. C, ¶¶ 249-252.

Lai teaches selecting a complexity level of the mapping function because it provides two exemplary functions, the results of which are illustrated in Figures 4 and 5, and discloses “said address re-mapping rule defining a one-to-one and onto function with said address conversion key, whose domain and range are the protected zone default address sequence[and the d]efined function may be a polynomial function, a triangle function, a dynamic function, a logarithm function, an exponential function, . . . etc.” Ex. PA-3, *Lai*, [0010]-[0011], [0025]-[0029], cl. 2-5, Figs. 4-5; Ex. C, ¶ 250.

default address	0	1	2	3	4	5	6	7	...	996	997	998	999	1000
re-mapped address	1000	999	998	997	996	995	994	993	...	4	3	2	1	0

FIG. 4

address conversion key	a				b				K				9				(unused)	(unused)
ASCII code	97				49				75				57				128	128
default address	0	1	...	96	97	...	145	146	...	230	231	...	277	278	...	...	499	
re-mapped address	96	95	...	0	145	...	97	230	...	146	277	...	221	405	...	...	496	

FIG. 5

Ex. PA-3, *Lai*, [0010]-[0011], Figs. 4-5.

Lai teaches that the selection of the re-mapping function sets an “efficiency of the mapping function” by disclosing that “using a mixture of said address re-mapping rule and said protected zone address re-mapping table” simplifies the calculation compared to using the re-mapping rule only and requires less memory space compared to using the protected zone re-mapping table only. Ex. PA-3, *Lai*, cl. 4; Ex. C, ¶ 251.

Lai teaches that selection of the re-mapping function sets “a security level . . . of the mapping function” by disclosing selecting a re-mapping rule that is a function of an “address conversion key” (higher security) or a function of the range of the protected zone (lower security). Ex. PA-3, *Lai*, [0024]-[0030]; Ex. C, ¶ 252. When the mapping rule is a function of a

key, “the address conversion can only be accomplished using said address re-mapping table” and, thus, is higher security. Ex. PA-3, *Lai*, cl. 4; Ex. C, ¶¶ 252.

8. Claim 7: “The method of claim 1, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”

Lai in view of *Card* renders obvious “wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.” Ex. C, ¶¶ 253-254.

As discussed for 1[a], the claimed “data block numbers” are *Card*’s “addresses of data blocks allocated to a file,” to which *Lai*’s re-mapping function is applied. Section VII.D.3.b; Ex. C, ¶ 254. *Lai*’s re-mapping function converts a default address sequence to a re-mapped address sequence within a protected zone. Ex. PA-3, *Lai*, [0024]-[0030], [0036]-[0037], [0040]-[0041]. If the file takes up the entire protected zone, then the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers because *Lai*’s re-mapping function converts an address within the range of the protected zone (e.g., a number between 0 and 1000) to another number within the same range where the data is stored. *Id.*; Ex. C, ¶ 254.

9. Claim 8: “The method of claim 1, wherein the mapping function prevents copying of data of the file in the storage device.”

Lai discloses or at least suggests “wherein the mapping function prevents copying of data of the file in the storage device.” Ex. C, ¶¶ 255-256.

Lai’s re-mapping rule “prevents copying of data of the file in the storage device” because *Lai* discloses that “computers without the data protection system and those computers with the data protection system but different re-mapping mechanism cannot read the correct data out of

the protected zone of the storage devices or media.” Ex. PA-3, *Lai*, Abstract, [0005]; Ex. C, ¶ 256.

10. Independent Claim 9

a. 9[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”

If the preamble is limiting, *Lai* in view of *Card* renders obvious “[a]n apparatus of providing data security for use with a file system.” Ex. C, ¶¶ 257-258.

Lai in view of *Card* renders obvious “providing data security for use with a file system” for the reasons discussed above for 1[pre]. *Supra* Section VII.D.3.a (1[pre]). *Lai* discloses an apparatus for providing this data security because it discloses, “in FIG. 1, the hardware system 10 of this configuration comprises a computer 11 providing . . . an access domain address conversion module 25 . . .” Ex. PA-3, *Lai*, [0015]; Ex. C, ¶ 258. The computer 11 (“[a]n apparatus”) comprising address re-mapping rule 60 and protected zone re-mapping table 65 teaches an apparatus for providing this data security as depicted below. Ex. C, ¶ 258.

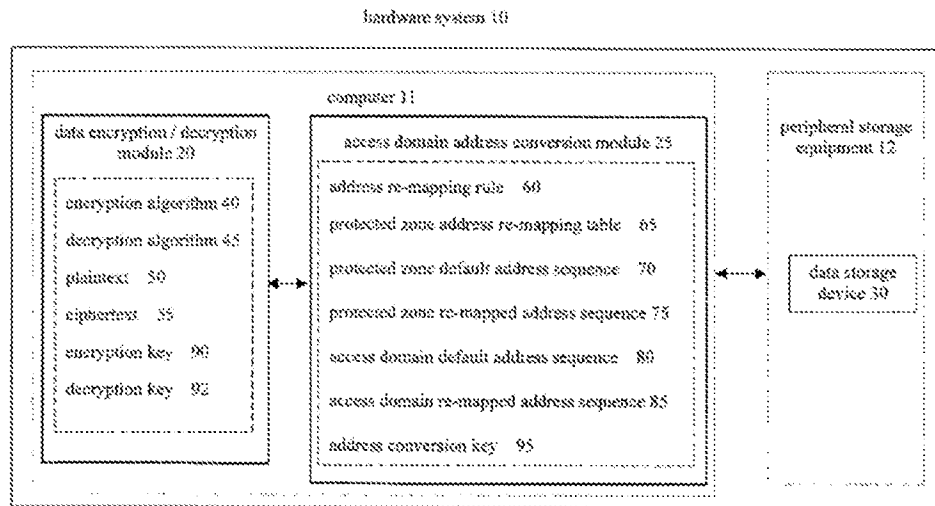


FIG. 1

Ex. PA-3, *Lai*, FIG. 1.

- b. 9[a]: “a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and”**

Lai in view of *Card* renders obvious “apply[ing] a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed for 1[a]. *Supra* Section VII.D.3.b (1[a]). *Lai* discloses or at least suggests a block distribution engine configured to perform this because it discloses an address conversion module 25 of a computer 11 that applies the mapping function. Ex. PA-3, *Lai*, Abstract, [0002], [0005]-[0011], [0016]-[0030], [0036]-[0037], [0040]-[0041], FIGs. 1-5 & 7; Ex. PA-4, *Card*, 2; Ex. C, ¶ 259; *supra* Section VII.D.3.b (1[a]).

- c. 9[b]: “obtain mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Lai in view of *Card* renders obvious “obtain[ing] mapped data block numbers after applying the mapping function; wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed for 1[b]. *Supra* Section VII.D.3.c (1[b]). *Lai* discloses or at least suggests a block distribution engine configured to perform this function because it discloses an address conversion module 25 of computer 11 that applies the mapping function to obtain re-mapped addresses (mapped data block numbers). Ex. PA-3, *Lai*, Abstract, [0002], [0005]-[0011], [0016]-[0030], [0036]-[0037], [0040]-[0041], FIGs. 1-5 & 7, cl. 1; Ex. PA-4, *Card*, 2; Ex. C, ¶ 260; *supra* Section VII.D.3.b (1[a]).

- 11. Claim 10: “The apparatus of claim 9, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.”**

Lai in view of *Card* renders obvious the additional limitations of claim 10 for the reasons discussed for claim 2. Ex. C, ¶¶ 261-262.

Lai discloses that address conversion module 25 (“block distribution engine”) is configured to store the mapping because it depicts the protected zone address re-mapping table 65 within the address conversion module 25, below, and discloses that the address conversion module 25 uses the table to obtain the re-mapped address sequence. Ex. PA-3, *Lai*, FIGs. 1-3, [0036]-[0037], [0040]-[0041]; Ex. C, ¶ 262.

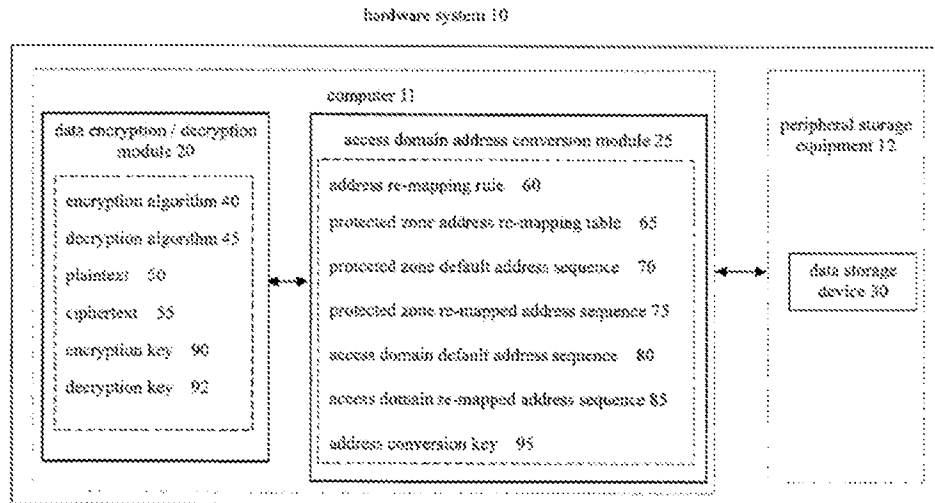


FIG. 1

Ex. PA-3, *Lai*, FIG. 1.

12. Claim 11

- a. **11[pre]: “The apparatus of claim 9, wherein for a read operation for the file,**

Lai in view of *Card* renders obvious “wherein for a read operation for the file” for the reasons discussed for 3[pre]. Ex. C, ¶ 263.

- b. **11[a]: “the block distribution engine obtains the data block numbers of the file from a file system, and”**

Lai in view of *Card* renders obvious “obtain[ing] the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed for 3[a]. Ex. C, ¶ 264; *supra* Section VII.D.5.b. As discussed for 3[a], *Lai*’s address conversion module 25 (“block distribution engine”) obtains the data block numbers of the file

from the filesystem to do the conversion, for example, according to *Card*'s procedure for loading the inode into memory. Ex. PA-3, *Lai*, [0039]-[0041]; Ex. PA-4, *Card*, 2-3; Ex. C, ¶ 264; *supra* Section VII.D.2.

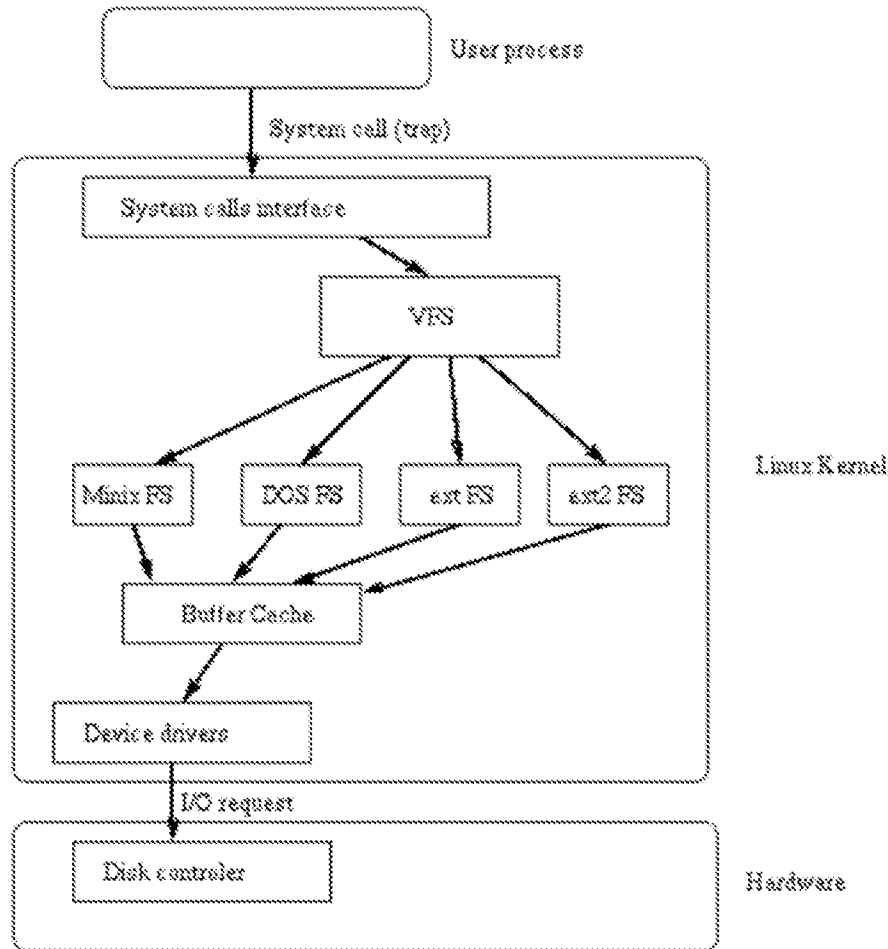
c. 11[b]: “obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and”

Lai in view of *Card* renders obvious “obtain[ing] the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers” for the reasons discussed in 3[b]. Ex. C, ¶ 265; *supra* Section VII.D.5.c. *Lai*'s address conversion module 25 (“block distribution engine”) performs this function because *Lai* discloses that address conversion module 25 applies the mapping function to convert the default address sequence 80 (“data block numbers”) to the re-mapped address sequence 85 (“mapped data block numbers”). Ex. PA-3, *Lai*, [0040]-[0041]; Ex. C, ¶ 265.

d. 11[c]: “wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.”

Lai in view of *Card* renders obvious “read[ing] the data in the mapped data block numbers in the storage device” for the reasons discussed for 3[c]. *Supra* Section VII.D.5.d; Ex. C, ¶¶ 266-268.

Lai in view of *Card* renders obvious “a block device driver” configured to perform this function. Ex. C, ¶ 267. For example, *Card* discloses a block device driver that reads data in the storage device because it discloses that the filesystem uses a “buffer cache” to request that a particular file be read, which passes the request to a device driver that reads the data by sending a corresponding “I/O request” to the disk controller of the storage device. Ex. PA-4, *Card*, 2-3. The block device driver is part of the “Device drivers” block in the figure below. Ex. C, ¶ 267.



Ex. PA-4, Card, 3.

Block device drivers were well known and commonly used to translate a data block number of the files to the physical locations on the disk. Ex. C, ¶ 268; see Ex. PA-6, *Mauro*, 25 (“Traditional UNIX implements kernel-resident device drivers to interface with hardware devices.”), 592-593 (disclosing that, after the file system “find[s] the location of the underlying physical blocks for the file being read,” it “calls the device driver’s strategy routine for the device containing the file system to initiate the read operation”). A POSITA would have been motivated to include a block device driver in *Lai*’s computer to perform this function because inodes typically included only the logical block addresses, requiring another component to then determine where on the disk that logical block address was stored. Ex. C, ¶ 268. This component

typically was the block device driver. Ex. PA-6, *Mauro*, 25. Indeed, such a driver was a well-known, “conventional UNIX component[.]” Ex. PA-5, *Carter*, [0111].

13. Claim 13: “The apparatus of claim 9, wherein the mapping function is an equation that is applied to each data block number of the file.”

Lai in view of *Card* renders obvious the additional limitations of claim 13 for the reasons discussed for Claim 5. *Supra* Section VII.D.6; Ex. C, ¶ 269.

14. Claim 14: “The apparatus of claim 9, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.”

Lai in view of *Card* renders obvious the additional limitations of claim 14 for the same reasons discussed in connection with Claim 6. Ex. C, ¶ 270.

15. Claim 15: “The apparatus of claim 9, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.”

Lai in view of *Card* renders obvious the additional limitations of Claim 15 for the same reasons discussed in connection with Claim 7. Ex. C, ¶ 271.

16. Claim 16: “The apparatus of claim 9, wherein the mapping function prevents copying of data of the file in the storage device.”

Lai discloses or at least suggests the additional limitations of claim 16 for the same reasons discussed in connection with Claim 8. Ex. C, ¶ 272.

17. Independent Claim 17:

a. 17[pre]: “An apparatus for providing data security for use with a file system, the apparatus comprising:”

Lai in view of *Card* renders obvious “[a]n apparatus for providing data security for use with a file system” for the reasons discussed for 9[pre]. Ex. C, ¶ 273.

- b. **17[a]: “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”**

Lai in view of *Card* renders obvious “means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed for 9[a]. Ex. C, ¶ 274. For example, as discussed in Section IV.E.1, a block distribution engine, which is recited in 9[a], is structure for the means-plus-function term recited in 17[a]. *Supra* Sections IV.E.1, VII.D.10.b (9[a]).

- c. **17[b]: “means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Lai in view of *Card* renders obvious “means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed for 9[b]. Ex. C, ¶ 275. For example, as discussed in Section IV.E.2, a block distribution engine, which is recited in 9[a] and performs the function in 9[b], is structure for the means-plus-function term recited in 17[b]. *Supra* Sections IV.E.2, VII.D.10.c (9[b]).

18. Independent Claim 18:

- a. **18[pre]: “Article of manufacture, comprising: a machine-readable medium having stored thereon instructions to:”**

If the preamble is limiting, *Lai* discloses or suggests an “[a]rticle of manufacture, comprising: a machine-readable medium having stored thereon instructions.” Ex. C, ¶¶ 276-277.

Lai discloses a hardware system 10 including a computer 11, highlighted below, having an access domain address conversion module 25. Ex. PA-3, *Lai*, [0015]. Computer 11 teaches an

article of manufacture comprising a machine-readable medium having stored instructions for performing the functions of 18[a] and 18[b], *infra*. Ex. PA-3, *Lai*, [0007], [0015], Figs. 1-3; Ex. C, ¶ 277. At a minimum, computer 11 containing two modules for carrying out *Lai*'s functions suggests a machine-readable medium having stored thereon instructions to perform the functions because modules encompass software, which is stored as instructions on a machine-readable medium. Ex. PA-3, *Lai*, FIG. 1; Ex. C, ¶ 277.

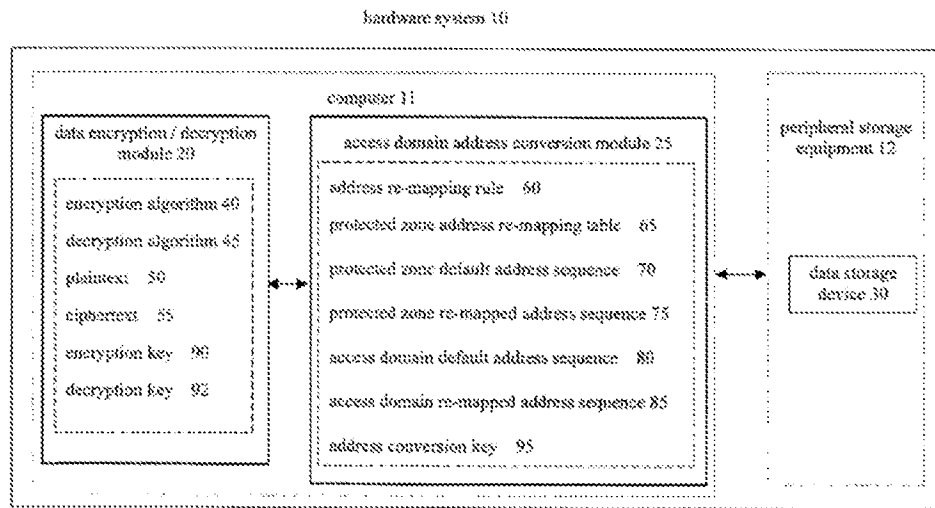


FIG. 1

Ex. PA-3, *Lai*, FIG. 1.

- b. **18[a]: “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and”**

Lai in view of *Card* renders obvious “apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file” for the reasons discussed for 1[a]. Ex. C, ¶ 278.

- c. **18[b]: “obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.”**

Lai in view of *Card* renders obvious “obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device” for the reasons discussed in connection with 1[b]. Ex. C, ¶ 279.

E. SNQ 6: *Lai* in view of *Card* and *Blumenau* Renders Obvious Claims 4 and 12

1. Overview of *Blumenau*

Blumenau relates to “[a] method and apparatus for manipulating data in a storage device that is coupled to a host computer.” PA-7, *Blumenau*, Abstract. *Blumenau* discloses a “typical computer system 100” that includes a host computer 110 and storage system 140. *Id.*, 1:12-19. *Blumenau* explains that “data stored in the storage device 140 is typically organized in units termed ‘blocks’, where each block includes a number of bytes of data (e.g., 512 bytes).” *Id.*, 1:31-33. Applications operating on the host computer 110 operate on files, one type of logical object, “that include a collection of one or more blocks of data that are logically related.” *Id.*, 1:34-37. But the applications do not see the physical locations of where the data is stored in the storage system. *Id.*, 2:56-60. Instead, “[t]he computer system 100 typically includes one or more mapping layers that map from the logical objects operated upon by the application programs to the particular physical locations in the storage system where the blocks that make up the logical object are stored.” *Id.*, 1:37-41.

Figure 2, below, depicts a computer system 100 having one or more mapping layers 220. *Id.*, 1:53-58, 2:8-17. “The mapping layer 220 can include a single layer of mapping, such as a file system 222 or a Logical Volume Manager (LVM) 224, or as shown in FIG. 2, can include multiple mapping layers 222 and 224.” *Id.*, 2:13-17. “The mapping layer maps each logical object specified

in application space 210 to one or more unique locations (e.g., blocks in physical space 230 where the data forming the logical object is stored.” *Id.* 2:10-13.

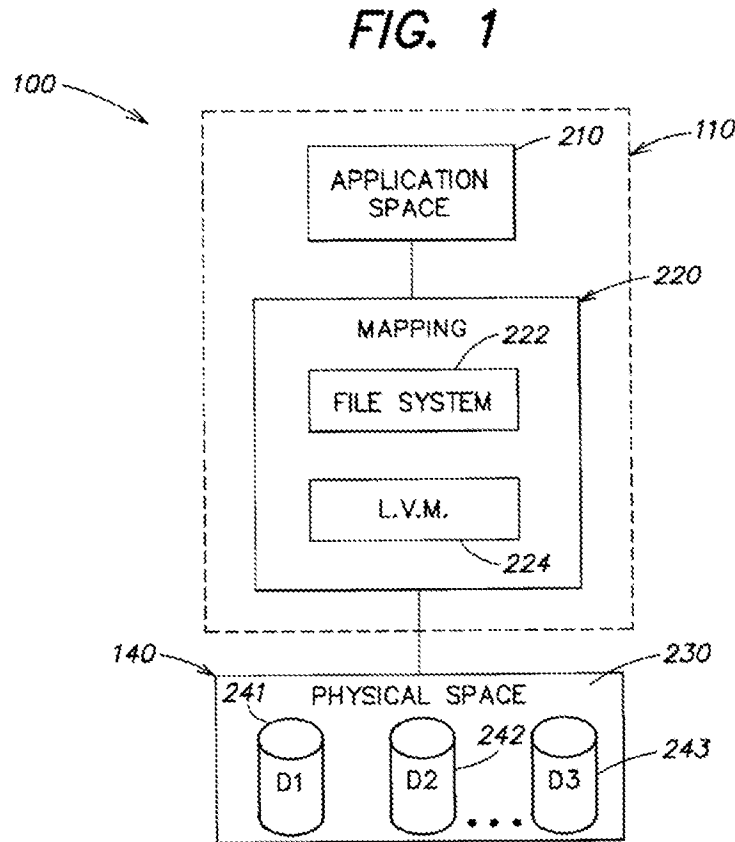


FIG. 2

Id., FIG. 2.

When an application program accesses a file, it identifies the file with “a logical object identifier, such as its file name.” *Id.*, 2:17-19. To write the file to the storage device 140, “the mapping layer defines the physical locations in the storage devices 241-243 that are written by the host computer 110 with the blocks of data that make up the [file].” *Id.*, 2:19-28. For example, if the mapping layer 220 includes a single mapping layer, “a word processing application issues a write command specifying the logical object identifier file 1,” the file system 222 “maps the logical object identifier file 1 to” a particular physical location in the storage devices (e.g., “disk 242,

blocks 0-3”), and the corresponding data of the file “is written to the appropriate location.” *Id.*, 2:34-56. If, instead, the mapping layer 220 includes multiple mapping layers, such as a file system 222 and an LVM 224, “the file system simply maps from the application space to what the file system perceives to be the physical space 230,” and the LVM converts from the file system’s perceived physical space into “the physical blocks that store” the file data. *Id.*, 2:61-3:11. In this configuration, when a write command specifies the logical object identifier file 1, the file system 222 maps this file to “a unique location where the file system 222 believes that data for the file 1 is stored.” *Id.*, 3:12-17. “[T]his unique location corresponds to a logical address, rather than an address in physical space 230.” *Id.*, 3:17-20. “The LVM 224, in turn, maps this logical address to physical blocks of data stored in physical space 230 on the disk drives 241-243 of storage device 140.” *Id.*, 3:20-22.

Blumenau discloses that “a typical file system manages large blocks of data including user data 310, metadata 320, and free space 330,” depicted in Figure 3A, below. *Id.*, 4:17-20. “[T]he user data 310 represents that area of memory where user data corresponding to files is stored, and the free space 330 represents blocks of user data that are currently unused.” *Id.*, 4:20-23. The block numbers depicted in user data 310 and free space 310 “are shown as a set of contiguous logical blocks of memory 0 to 10,000 that are accessible by the file system,” which “may map directly or indirectly to blocks in physical space 320, depending on the number of levels of mapping between the file system and physical space.” *Id.*, 4:23-29. In free space 330, unused blocks are circled. *Id.*, 4:34-36. “Typically a table indicating blocks of free space 330 is stored (or cached) in the memory 130 of the host computer 110 . . . so that blocks of free space 330 can be allocated quickly by the file system 222 operating on the host computer 110.” *Id.*, 4:36-40.

310 USER DATA	320 METADATA	330 FREE SPACE
0 1 2 3 4 5 6 7 8.....15 10,000		① 1 ② 3 ④ ⑤ 6 7 ⑧.....15 10,000

FIG. 3A

Id., FIG. 3A.

The metadata 320, depicted below in more detail, “is also typically stored (or cached) in memory 130 of the host computer.” *Id.*, 4:41-42. “The metadata 320 is used by the file system to keep track of the logical assignment of each block of user data 310.” *Id.*, 4:42-44. “Typically, there is a metadata entry for each logical object owned by the file system” and “includes a number of fields of information, such as the name of the file,” “the location of the logical object at the next lowest layer,” and “the level of protection assigned to the file.” *Id.*, 4:44-50. When there are multiple mapping layers, the location information in the file system metadata provides “an index into mapping information (metadata or an equivalent data structure) for the next lowest mapping layer,” and that process repeats “until reaching the lowest mapping layer.” *Id.*, 5:28-35, 14:59-61.

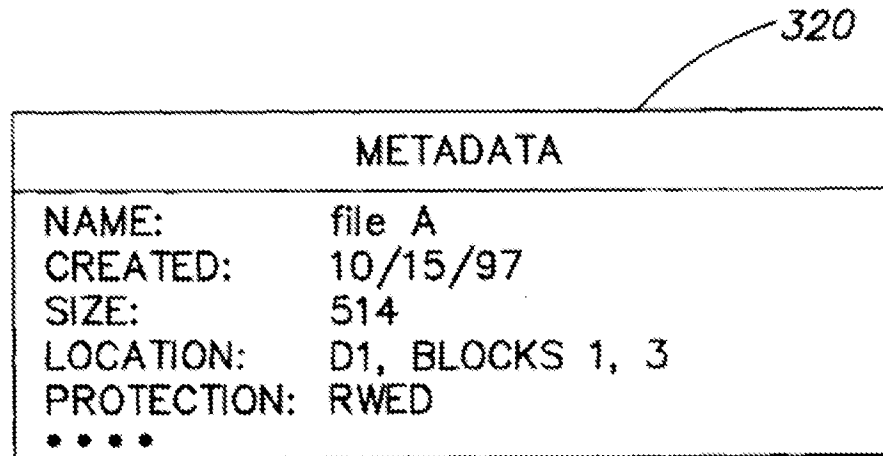


FIG. 3B

Id., FIG. 3B.

Blumenau discloses several examples of write operations for writing data to the disk. *E.g.*, *id.*, 4:59-6:24, 10:42-12:50. After the file system creates a metadata entry for the file to be written, the file system accesses the metadata entry of the file, determines that “the size and location fields are empty,” allocates “one or more blocks from free space 330 (e.g., logical block 0 or “the requisite number of blocks from the available blocks in free-space 330”) to store this data, “updates the metadata entry” for the file to reflect the size and location, “call[s] a mapping routine” that “returns the [file] in which is perceived by the host computer to be physical space,” and writes the file data to the allocated blocks. *Id.*, 5:23-6:24, 11:22-12:50.

Blumenau is analogous art because it is in the same field of endeavor as the '686 patent—protecting digital information in storage devices. *Compare id.*, 2:56-60, 2:61-3:11, with Ex. A, Title, Abstract, Claims 1, 9, 17, 18; Ex. C, ¶ 78. *Blumenau* is also analogous art because it is reasonably pertinent to a problem discussed in the '686 patent. For example, the '686 patent

discloses that, “on a UNIX file system, an individual can retrieve the data blocks of a file by accessing the index node (inode) of the file and completely bypass the application that is used for accessing the file.” Ex. A, 1:29-32. *Blumenau* is reasonably pertinent to this purported problem because it discloses that the metadata maintained for a file in the file system contains “the location of the logical object at the next lowest layer” and requires a “mapping routine” to iterate through the mapping layers of the host computer and obtain the mapping of the file “in what is perceived by the host computer to be physical space.” Ex. PA-7, *Blumenau*, 4:41-50, 12:1-17, 15:52-16:24; Ex. C, ¶ 78. Moreover, *Blumenau* discloses that the storage device may employ “an intelligent storage system” with “additional mapping,” such that the location perceived by the host computer to be physical space may not be “the location of the data in physical space.” Ex. PA-7, *Blumenau*, 17:4-27; Ex. C, ¶ 78. *Blumenau* discloses that existing applications at the time were “incapable of providing an overall view of the mapping of a logical object through any two or more layers of mapping in the mapping hierarchy.” Ex. PA-7, *Blumenau*, 21:16-22. Because the locations of a file contained in the file system’s metadata may not be the locations in physical space, *Blumenau* is reasonably pertinent to the purported problem of a file system’s inode allowing an individual to by the application and retrieve the data blocks of a file. Ex. C, ¶ 78.

2. Reasons to Combine *Lai*, *Card*, and *Blumenau*

For the reasons discussed above in Section VII.D.2, a POSITA would have found it obvious to combine *Lai* and *Card* to protect data within a file system by applying *Lai*’s re-mapping rule to addresses associated with a file and contained in *Card*’s inodes. Ex. C, ¶ 281. A POSITA would have found it obvious to further combine *Lai* and *Card* with *Blumenau* because *Blumenau* provides details on how to efficiently implement write operations for file data associated with a file system when the physical data blocks are not contiguous in the storage system and one or more layers of mapping are used. Ex. C, ¶ 281. *Lai*, *Card*, and *Blumenau* are

each analogous art to the '686 patent for the reasons discussed in Sections VII.B.1, VII.D.1, and VII.E.1.

Lai discloses a “procedure of storing data to the protected zone of the storage device or media” that includes applying the address conversion rule 60 (a mapping function) “to convert the access domain default address sequence 80 . . . designated by the base computer system to the access domain re-mapped address sequence 85” and then storing the data “to the storage device or media according to the access domain re-mapped address sequence 85.” Ex. PA-3, *Lai*, [0031], [0036]-[0038]. But *Lai* is silent as to how the base computer system allocates the default address sequence. Ex. C, ¶ 282. For example, *Lai* discloses how to convert the sequence of default addresses into a sequence of remapped addresses within the protected zone and discloses writing data into the remapped addresses, but *Lai* does not specify how the computer system allocates the initial sequence of default addresses for the data to be written Ex. PA-3, *Lai*, [0022], [0031]-[0038]; Ex. C, ¶ 282. Similarly, *Card* discloses that “addresses of data blocks allocated to a file are stored in its inode” but lacks implementation details on how the file system determines free blocks to allocate. Ex. PA-4, *Card*, 2; Ex. C, ¶ 282.

Blumenau discloses details regarding how a file system allocates data block numbers of free data blocks to a file during a write operation. Ex. C, ¶ 283. For example, *Blumenau* discloses a write operation in which data from file A is copied to file B. Ex. PA-7, *Blumenau*, 10:66-12:50. “If a metadata entry for file B does not exist, then the file system creates a metadata entry for file B in the appropriate mapping layer (e.g., file system 222 in FIG. 2).” *Id.*, 11:22-26. At this point, the metadata fields for “the size and location of file B will be empty.” *Id.*, 11:26-30. The write operation requests that the file system “allocate it a number of blocks for file B sufficient to store the amount of data of file A.” *Id.*, 11:50-55. In doing so, “the file system selects the requisite

number of blocks from the available blocks in free-space 330 (FIG. 3A),” reproduced below. *Id.*, 11:59-62. “Thereafter, the file system updates the metadata entry of file B” to include the data block numbers of file B and the file’s size. *Id.*, 11:62-67.

310 USER DATA	320 METADATA	330 FREE SPACE
0 1 2 3 4 5 6 7 8.....15 10,000		① 1 ② 3 ④ ⑤ 6 7 ⑧ 15 10,000

FIG. 3A

Id., FIG. 3A.

After updating the metadata entry with the block numbers for file B, the write operation “requests the mapping of the logical object file B by calling a mapping routine, which “returns the mapping of the logical object file B in what is perceived by the host computer to be the physical space.” *Id.*, 12:1-6. The write operation then “issues a command to the storage device instructing the storage device to do an internal copy of file A to file B,” which “identifies the location of each physical block of file A and its corresponding destination physical block in file B.” *Id.*, 12:11-17. The storage device then executes the command and writes the file data to the destination physical blocks for file B. *Id.*, 12:18-50; Ex. C, ¶ 284.

A POSITA would have been motivated to implement this write operation into the data protection system of *Lai* and *Card* because it provides implementation details for how to write

new data to files and because *Blumenau* discloses that its write operation improves performance by allowed the storage device to carry out write operations with “a single operation,” even when a file’s data blocks are stored in non-contiguous blocks. Ex. C, ¶ 285; Ex. PA-7, *Blumenau*, Abstract, 6:25-65, 9:62-10:8. In this combination, the metadata in *Blumenau*’s write operation corresponds to *Card*’s inode. Ex. C, ¶ 285. Both, for example, are types of metadata that contain information about a particular file, including the size and location of the file. Ex. C, ¶ 285; Ex. PA-4, *Card*, 2 (describing the inode as including “the description of the file,” including its “size” and “addresses of data blocks allocated to a file”); Ex. PA-7, *Blumenau*, 11:22-30, 11:63-67 (disclosing that “there is a metadata entry for each logical object owned by the file system,” which includes “the size of the file” and its location “at the next lowest layer”). Additionally, in this combination, the mapping routine in *Blumenau*’s write operation corresponds to *Lai*’s application of the re-mapping rule. Ex. C, ¶ 285. Both, for example, take one set of addresses as the input and output the addresses where the data is to be stored. Ex. PA-3, *Lai*, [0036]-[0038]; Ex. PA-7, *Blumenau*, 12:1-17.

Blumenau explains that storage devices in conventional computer systems were “not aware of the logical relationship amongst the blocks of data stored therein,” so the host computer, which had this knowledge, had to “be intimately involved in the execution of most operations with the storage device.” Ex. PA-7, *Blumenau*, 6:25-30. This meant that even a simple copy routine required multiple I/O operations from the host computer to the storage device. *Id.*, 6:30-34. “By providing the storage device with knowledge of the relationship between a logical object and the storage locations corresponding to that logical object, data corresponding to the logical object can be manipulated directly by the storage device” with “a single operation.” *Id.*, 6:47-59. Accordingly, a POSITA would have recognized benefits in implementing *Blumenau*’s

write operation as described above because doing so would have limited the number of I/O operations needed to write data to storage, thereby improving the efficiency of the system. Ex. C, ¶ 286.

Moreover, *Lai* and *Card* both already disclose similar write operations in which a sequence of addresses/data block numbers are allocated to a file. Ex. C, ¶ 287. In *Lai*, a “base computer system” designates a default address sequence, which is then converted to a re-mapped address sequence to obtain the addresses where the data is stored. Ex. PA-3, *Lai*, [0031], [0036]-[0038]. In *Card*, “the addresses of data blocks allocated to a file are stored in its inode,” and “[w]hen a user requests an I/O operation on the file, the kernel code converts the current offset to a block number, uses this number as an index in the block addresses table[,] and reads or writes the physical block.” Ex. PA-4, *Card*, 2. Because *Blumenau* simply provides the details of these write operations, combining *Lai*, *Card*, and *Blumenau* would have been a combination of prior art elements according to known methods to yield predictable results. Ex. C, ¶ 287.

A POSITA would have had a reasonable expectation of success in combining *Lai*, *Card*, and *Blumenau* to implement *Blumenau*’s write operation as described above. Ex. C, ¶ 288. *Blumenau* itself provides a reasonable expectation of success because it discloses that even file systems that do not support “pre-allocating a number of logical blocks to a logical object” “can be modified to provide this capability.” Ex. PA-7, *Blumenau*, 11:55-59. Accordingly, a POSITA would have expected combining *Lai*, *Card*, and *Blumenau* to require only conventional techniques compatible with the well-known file systems. Ex. C, ¶ 288. For example, *Blumenau* discloses an embodiment where the computer system implements a “UNIX operating system,” and *Card* discloses that its file system “is a Unix-like operating system.” Ex. PA-7, *Blumenau*, 15:4-22; Ex. PA-4, *Card*, 2. As such, a POSITA would have expected combining *Lai*, *Card*, and

Blumenau to require only conventional techniques compatible with every well-known file systems and ubiquitous at the time. Ex. C, ¶ 288. A POSITA would have seen a file system that used inodes to organize the addresses of files on a computer, as taught by *Card*, and a write operation that allocates data block numbers to a file, stores the allocated data block numbers in the inode, applying a mapping function to the data block numbers in the inode, and stores the file's data in the mapped data block numbers as a well-known and predictable way for the operating system to write data on storage devices, as taught by *Lai*, *Card*, and *Blumenau*. Ex. C, ¶ 288.

3. Claim 4:

a. 4[pre]: “The method of claim 1, wherein a write operation for the file comprises:”

Lai in view of *Card* or *Lai* in view of *Card* and *Blumenau* renders obvious “wherein a write operation for the file comprises.” Ex. C, ¶¶ 289-292.

Lai discloses writing data to “storage cells corresponding to re-mapped addresses.” Ex. PA-3, *Lai*, [0002], [0006], [0022]. As discussed for claim 1, in the *Lai-Card* combination, the data corresponds to a file in the file system. *Supra* Section VII.D.3; Ex. C, ¶ 290.

Additionally, *Blumenau* discloses “a write operation for the file” because it discloses writing data from a host computer to a storage device, for example, by “writing” data from one file to another file in the file system. Ex. PA-7, *Blumenau*, 10:36-12:50; Ex. C, ¶ 291.

For the reasons discussed in Section VII.E.2, it would have been obvious to combine *Lai*, *Card*, and *Blumenau* to implement *Blumenau*'s write operation for a file that uses *Card*'s inode as the metadata and *Lai*'s mapping function as the mapping routine. Ex. C, ¶ 292.

b. 4[a]: “allocating data block numbers of free data blocks in the storage device;”

Lai in view *Card* and *Blumenau* renders obvious “allocating data block numbers of free data blocks in the storage device.” Ex. C, ¶¶ 293-295.

Blumenau discloses “allocating data block numbers of free data blocks in the storage device” because it discloses that a file system “pre-allocat[es] blocks” to a particular file by “select[ing] the requisite number of blocks from the available blocks in free-space 330 (FIG. 3A).” Ex. PA-7, *Blumenau*, 11:59-62; Ex. C, ¶ 294. Free-space 330 “represents blocks of user data that are currently unused.” Ex. PA-7, *Blumenau*, 4:17-23. These free data blocks are in the storage device because *Blumenau* discloses that they are “logical blocks of memory” that “map directly or indirectly to blocks in physical space 230, depending on the number of levels of mapping between the file system and physical space.” Ex. PA-7, *Blumenau*, 4:23-29; Ex. C, ¶ 294. Physical space 230 refers to the storage device. Ex. PA-7, *Blumenau*, 1:63-65 (“The data forming those logical objects [(e.g., files)] is stored on one or more storage devices 241-243 that are included in the storage system 140 and define a physical space 230.”).

For the reasons discussed in Section VII.E.2, it would have been obvious to combine *Lai*, *Card*, and *Blumenau* to implement *Blumenau*’s write operation for a file, including its process for allocating data block numbers of free data blocks in the storage device. Ex. C, ¶ 295. Doing so would have been beneficial, for example, so that the file system allocates data block numbers for a new file without inadvertently overwriting data already stored in the storage device. Ex. C, ¶ 295.

c. 4[b]: “linking the data block numbers to a file name of the file;”

Lai in view *Card* and *Blumenau* renders obvious “linking the data block numbers to a file name of the file.” Ex. C, ¶¶ 296-298.

Blumenau discloses “linking the data block numbers to a file name of the file” because it discloses that, after the data block numbers are allocation, “the file system updates the metadata entry” of the file to include the allocated data block numbers (e.g., “at disk DI, blocks 0 and 2”). Ex. PA-7, *Blumenau*, 11:59-67; *see also id.*, 5:61-65; Ex. C, ¶ 297. This disclosure of updating the metadata entry discloses linking the data block numbers to a file name of the file because *Blumenau* discloses that the metadata entry for the file contains “certain fields of information” for the file, “such as its name.” Ex. PA-7, *Blumenau*, 11:26-30. By updating the metadata entry to include both the file name and the allocated data block numbers, *Blumenau* discloses “linking the data block numbers to a file name of the file.” Ex. C, ¶ 297.

For the reasons discussed in Section VII.E.2, it would have been obvious to combine *Lai*, *Card*, and *Blumenau* to implement *Blumenau*’s write operation for a file, including its process for linking the data block numbers to a file name of the file. Ex. C, ¶ 298. Doing so would have been beneficial, for example, to allow the file system to easily convert a write command from an application using the file name to an address in the inode. Ex. C, ¶ 298. While *Blumenau* discloses a single metadata entry containing both the file name and the allocated data block numbers, a POSITA would have recognized that implementing *Blumenau*’s linking using *Card*’s directories would involve linking the allocated data block numbers to a file name of the file through a hierarchical metadata tree. Ex. C, ¶ 298; Ex. PA-4, *Card*, 2-3. For example, *Card* discloses a directory with entries that each contain “an inode number and a file name.” Ex. PA-4, *Card*, 3. *Card*’s file system obtains an inode number from the file name and then loads the inode into memory for “subsequent requests.” *Id.* In this configuration, a POSITA would have understood that the file system links the allocated data block numbers to a file name of the file through the metadata hierarchy, rather than a single metadata entry. Ex. C, ¶ 298.

d. 4[c]: “applying the mapping function to the data block numbers in order to obtain the mapped data block numbers; and”

Lai in view of *Card* and *Blumenau* renders obvious “applying the mapping function to the data block numbers in order to obtain the mapped data block numbers.” Ex. C, ¶ 299. As discussed in SNQ 5 regarding 1[a], *Lai* in view of *Card* renders obvious “applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file. *Supra* Section VII.D.3.b; Ex. C, ¶ 299.

Blumenau discloses that, after the data block numbers are allocated and linked to the metadata entry of the file, the write operation “requests the mapping” of the file by “calling a mapping routine” that “returns the mapping” of the file “in what is perceived by the host computer to be physical space.” Ex. PA-7, *Blumenau*, 11:59-12:6. This mapping routine obtains mapped data block numbers because the write operation next “issues a command” that includes the physical data block numbers where the file data blocks are to be written. Ex. C, ¶ 300; Ex. PA-7, *Blumenau*, 12:1-17.

For the reasons discussed in Section VII.E.2, it would have been obvious to combine *Lai*, *Card*, and *Blumenau* to implement *Blumenau*’s write operation for a file such that the file system uses *Lai*’s mapping function as the mapping routine and applies this mapping function to the allocated data block numbers of the file to obtain the mapped data block numbers where the file will be stored within the storage device. Ex. C, ¶ 301. In this combination, *Lai*’s mapping function discussion in SNQ 5 regarding 1[a] is applied to the allocated data block numbers of the file contained in *Card*’s inode. Ex. C, ¶ 301. This mapping function obtains the mapped data block numbers for the reasons discussed in SNQ 5 regarding 1[b]. *Supra* Section VII.D.3.c; Ex. C, ¶ 301.

e. 4[d]: “writing the data of the file into the mapped data block numbers in the storage device.”

Lai in view of *Card* and *Blumenau* renders obvious “writing the data of the file into the mapped data block numbers in the storage device.” Ex. C, ¶¶ 302-305.

Lai discloses storing the data “to the storage device or media according to the access domain re-mapped address sequence.” Ex. PA-3, *Lai*, [0031], [0036]-[0038]; Ex. C, ¶ 303.

Blumenau discloses that, after obtaining “the mapping” of the file “in what is perceived by the host computer to be physical space,” the file system “issues a command to the storage device instructing the storage device” on the data to be written and the physical data block numbers on which to write the data. Ex. PA-7, *Blumenau*, 12:1-38; Ex. C, ¶ 304.

For the reasons discussed in Section VII.E.2, it would have been obvious to combine *Lai*, *Card*, and *Blumenau* to implement *Blumenau*’s write operation for a file such that the file system obtains the mapped data block numbers using *Lai*’s mapping function as the mapping routine and then writes the data of the file into the mapped data block numbers in the storage device by issuing a command to the storage device on where to write the data. Ex. PA-3, *Lai*, [0031], [0036]-[0038]; Ex. PA-7, *Blumenau*, 12:1-38; Ex. C, ¶ 305.

4. Claim 12

a. 12[pre]: “The apparatus of claim 9, wherein for a write operation for the file,”

Lai in view of *Card* or *Lai* in view of *Card* and *Blumenau* renders obvious “wherein for a write operation for the file” for the reasons discussed for 4[pre]. Ex. C, ¶ 306.

b. 12[a]: “the block distribution engine is configured to receive allocated data block numbers of free data blocks in the storage device from a file system,”

Lai in view of *Card* and *Blumenau* renders obvious “allocating data block numbers of free data blocks in the storage device” for the reasons discussed for 4[a]. Ex. C, ¶ 307; *supra* Section

VII.E.3.b. *Lai*'s address conversion module 25 ("block distribution engine") receives the allocated data block numbers from the file system to do the conversion, for example, according to *Card*'s procedure for loading the inode into memory. Ex. PA-3, *Lai*, [0036]-[0038]; Ex. PA-4, *Card*, 2-3; Ex. C, ¶ 307; *supra* Section VII.E.2.

c. 12[b]: "link the data block numbers to a file name of the file, and"

Lai in view of *Card* and *Blumenau* renders obvious "link[ing] the data block numbers to a file name of the file" for the reasons discussed in 4[b]. Ex. C, ¶ 308; *supra* Section VII.E.3.c. It would have been obvious to have *Lai*'s address conversion module 25 ("block distribution engine") perform this linking because doing so would have been a matter of routine software design. Ex. C, ¶ 308. While *Blumenau* discloses that "the file system updates the metadata entry" of the file, modifying this configuration to have *Lai*'s address conversion module 25 perform this linking would have been a simple substitution of one known element for another. Ex. C, ¶ 308. At a minimum, there are only a finite number of identified modules (e.g., file system or address conversion module), so modifying *Blumenau*'s teaching to have *Lai*'s address conversion module 25 perform the linking would have been obvious to try. Ex. C, ¶ 308; *see In re Kuhle*, 526 F.2d 553, 555 (CCPA 1975) (affirming obviousness because the placement of a feature provided "no novel or unexpected result" and would have been "an obvious matter of design choice within the skill of the art").

d. 12[c]: "apply the mapping function to the data block numbers in order to obtain the mapped data block numbers; and"

Lai in view of *Card* or *Lai* in view of *Card* and *Blumenau* renders obvious "apply[ing] the mapping function to the data block numbers in order to obtain the mapped data block numbers" for the reasons discussed in 4[c]. Ex. C, ¶ 309; *supra* Section VII.E.3.d. *Lai*'s address conversion module 25 ("block distribution engine") performs this function because *Lai* discloses that address

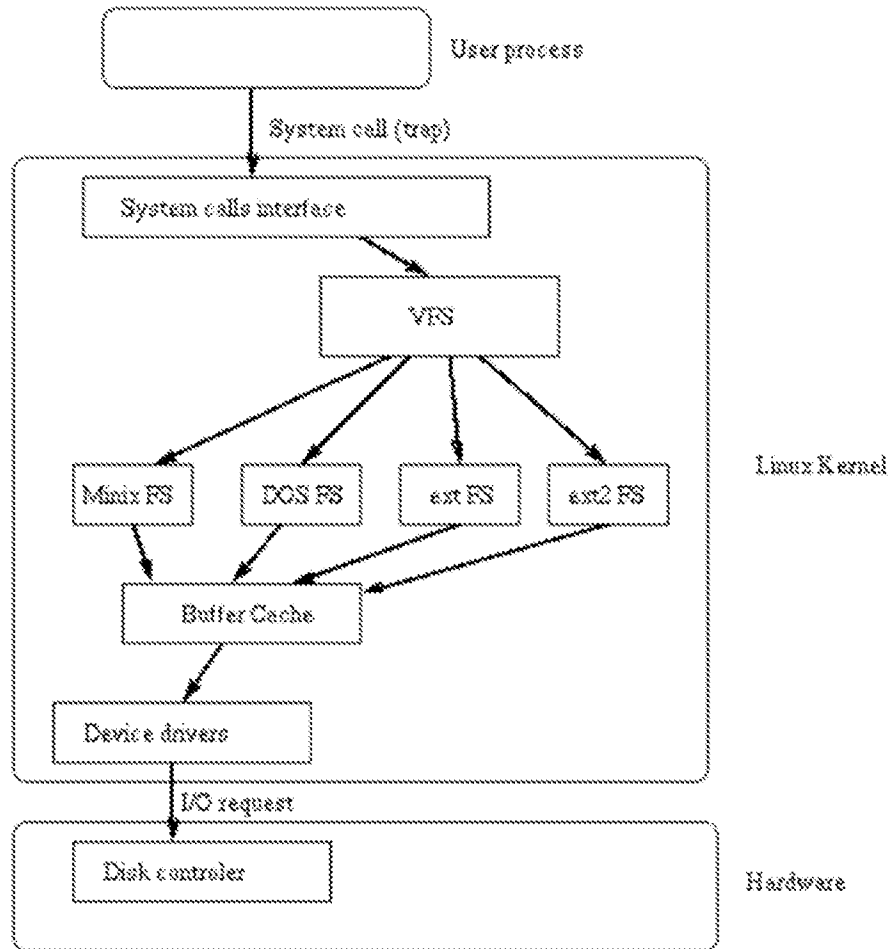
conversion module 25 applies the mapping function to convert the default address sequence 80 (“data block numbers”) to the re-mapped address sequence 85 (“mapped data block numbers”).

Ex. PA-3, *Lai*, [0040]-[0041]; Ex. C, ¶ 309.

- e. **12[d]: “wherein a block device driver is configured to write the data of the file into the mapped data block numbers in the storage device.”**

Lai in view of *Card* and *Blumenau* renders obvious “writing the data of the file into the mapped data block numbers in the storage device” for the reasons discussed for 4[d]. *Supra* Section VII.E.3.e; Ex. C, ¶ 310.

Lai in view of *Card* renders obvious “a block device driver” configured to perform this function. For example, *Card* discloses a block device driver that writes data in the storage device because it discloses that the filesystem uses a “buffer cache” to request that a particular file be written, which passes the request to a device driver that writes the data by sending a corresponding “I/O request” to the disk controller of the storage device. Ex. PA-4, *Card*, 2-3. The block device driver is part of the “Device drivers” block in the figure below. Ex. C, ¶ 311.



Ex. PA-4, Card, 3.

Block device drivers were well known and commonly used to translate a data block number of the files to the physical locations on the disk. Ex. C, ¶ 312; *see* Ex. PA-6, *Mauro*, 25 (“Traditional UNIX implements kernel-resident device drivers to interface with hardware devices.”), 592-593 (disclosing that, after the file system “find[s] the location of the underlying physical blocks for the file being read,” it “calls the device driver’s strategy routine for the device containing the file system to initiate the read operation”). A POSITA would have been motivated to include a block device driver in *Lai*’s computer to perform this function because inodes typically included only the logical block addresses, requiring another component to then determine where on the disk that logical block address is located. Ex. C, ¶ 312. This component

typically was the block device driver. Ex. PA-6, *Mauro*, 25. Indeed, such a driver was a well-known, “conventional UNIX component[.]” Ex. PA-5, *Carter*, [0111].

VIII. CONCLUSION

For the foregoing reasons, NetApp requests reexamination and cancellation of claims 1-18 of the '686 patent.

NetApp is willing to provide appropriate assistance to permit the Examiner to address and decide the issues presented by this Request. As the M.P.E.P. explains, the Examiner may, when appropriate, cut and paste claim charts or other material within the Request to incorporate them within the body of an Office Action. M.P.E.P. § 2262. NetApp is, thus, through the undersigned counsel, available to provide the Examiner with a digital copy of this Request, or any portion of it, in response to a request by email or phone. NetApp also understands that the Examiner may, in appropriate circumstances, set forth specific rejections in an Office Action and incorporate by reference NetApp's reasons for the proposed rejections, if the Examiner agrees with the proposed rejections and reasons supporting them.

If there is any additional fee due in connection with this filing that is not already submitted herewith, please charge the fee to Deposit Account No. 06-0916.

Respectfully submitted,
FINNEGAN, HENDERSON, FARABOW,
GARRETT & DUNNER, LLP

Dated: September 24, 2025

By: /Cory C. Bell/
Cory C. Bell
Reg. No. 75,096

Attachments: (1) Certificate of Service to Patent Owner
(2) Information Disclosure Statement Form PTO/SB/08
(3) Exhibits listed on page iii of this Request