

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

INTERNATIONAL BUSINESS MACHINES CORPORATION

Petitioner

v.

SECURITY FIRST INNOVATIONS, LLC

Patent Owner

Case IPR2025-01202

U.S. Patent No. 9,135,456

Claims 1-30

DECLARATION OF EREZ ZADOK, PH.D.

TABLE OF CONTENTS

	<u>Page</u>
I. BACKGROUND AND QUALIFICATIONS	1
II. MATERIALS CONSIDERED	12
III. SUMMARY OF OPINIONS	15
IV. RELEVANT LAW	16
A. Claim Construction	16
B. Obviousness	17
C. Ordinary Skill in the Art	18
V. TECHNOLOGY OVERVIEW	19
A. General Computer Operations	19
B. Networking	24
1. Networked and Distributed Storage Systems	28
C. Data Security and Privacy	32
1. Hash Functions and Their Uses	39
2. Secret Splitting and Sharing	42
D. Recovering from Data Loss	44
E. All-Or-Nothing-Transform (“AONT”)	55
VI. OVERVIEW OF THE ’456 PATENT	60
A. Prosecution History	63
VII. LEVEL OF ORDINARY SKILL IN THE ART	64
VIII. CLAIM CONSTRUCTION	65

IX.	SPECIFIC GROUNDS FOR CHALLENGE	65
A.	Ground I: Torre and Desai Render Obvious Claims 1–3, 10–15, 22–27, And 30.	65
1.	Desai Color-Coding Scheme.....	66
2.	Overview of Torre (EX1003).....	66
a.	Overview of Torre Figure 21 Embodiment (“Torre System-1”)	76
b.	Overview of Torre Figure 22 Embodiment (“Torre System-2”)	81
3.	Overview of Desai (EX1004)	86
4.	Combining Torre With Desai (“Torre/Desai System”)	92
a.	Overview of Torre/Desai System-1	95
b.	Overview of Torre/Desai System-2.....	96
c.	Reasons That A POSITA Would Have Been Motivated To Combine Torre and Desai To Arrive At Torre/Desai System-1 and Torre/Desai System- 2	98
5.	Claim 1	106
a.	[1Pre] A method performed by a computer system for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the method comprising:.....	106
b.	[1A] reading the first data set from a memory that stores the first data set;	113
c.	[1B] processing the first data set to produce a second data set,	115

- d. [1B-1] the processing comprising ... encrypting the first data set using an encryption key to produce an encrypted data set, and.....118
 - e. [1B-2] the processing comprising ... logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant,120
 - f. [1B-3] wherein the second data set comprises the encrypted data set and the resultant, and wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant;.....121
 - g. [1C] producing redundancy information based on information in the second data set;124
 - i. Torre/Desai System-1124
 - ii. Torre/Desai System-2.....127
 - h. [1D] producing a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set; and.....129
 - i. Torre/Desai System-1129
 - ii. Torre/Desai System-2.....131
 - i. [1E] storing the plurality of n shares and the redundancy information, wherein at least some of the n shares are stored in separate storage locations and the first data set is recoverable using at least a threshold number, less than n, of the plurality of n shares134
 - i. Torre/Desai System-1134
 - ii. Torre/Desai System-2.....138
6. Claim 2: The method of claim 1, wherein each of the n shares further comprises at least some of the redundancy

	information and storing the plurality of n shares and the redundancy information comprises storing the redundancy information in the n shares.....	142
7.	Claim 3: The method of claim 2, wherein storing the redundancy information in the n shares comprises including in each of the n shares information redundant to information included in at least one other of the n shares.....	145
8.	Claim 10: The method of claim 1, wherein at least some of the separate storage locations are geographically separate from one another.....	148
9.	Claim 11: The method of claim 1, wherein logically combining comprises logically combining using an exclusive-OR operation.	150
10.	Claim 12: The method of claim 1, wherein the at least a portion of the encrypted data set comprises all of the encrypted data set.....	151
11.	Claim 13.....	152
	a. [13Pre] A computer system for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the system comprising one or more processors configured to:	153
12.	Claim 25.....	154
	a. [25Pre] A non-transitory computer-readable medium comprising instructions that, when executed by one or more processors, cause a computer system to carry out a method for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the method comprising:.....	155
13.	Claims 14, 15, 22-24, 26, 27, and 30	157

B.	Ground II: Torre, Desai, and Watanabe Render Obvious Claims 1-3, 10-12, 25-27, And 30.	157
1.	Torre/Desai/Watanabe Color-Coding Scheme	157
2.	Overview of Watanabe (EX1005)	158
3.	Combining Torre/Desai With Watanabe ("Torre/Desai/Watanabe System")	159
a.	Reasons That A POSITA Would Have Been Motivated To Combine Torre/Desai With Watanabe To Arrive At The Torre/Desai/Watanabe System	161
4.	Claim 1	164
a.	[1A] reading the first data set from a memory that stores the first data set;	164
5.	Claim 25	166
6.	Claims 2, 3, 10–12, 26, 27, and 30	166
C.	Ground III: Ground I Or Ground II In Combination With Orsini Renders Obvious Claims 4-9, 16-21, and 28-29.....	167
1.	Torre/Desai(/Watanabe)/Orsini Color-Coding Scheme	167
2.	Overview of Orsini (EX1006)	167
3.	Combining Grounds I or II With Orsini	169
a.	Combining Torre/Desai(/Watanabe) System With Orsini's Cryptosplit Process ("Torre/Desai (/Watanabe)/Orsini System-1")	170
i.	Reasons Why A POSITA Would Have Combined Torre, Desai, Orsini (and, optionally, Watanabe) As Provided In Torre/Desai (/Watanabe)/Orsini System-1	172

b.	Combining Torre/Desai(/Watanabe) System With Orsini's Data Shuffling Process ("Torre/Desai (/Watanabe)/Orsini System-2")	176
i.	Reasons that a POSITA Would Have Combined Torre, Desai, Orsini (and, optionally, Watanabe) As Provided In Torre/Desai (/Watanabe)/Orsini System-2	178
4.	Claim 4: The method of claim 1, wherein producing the plurality of n shares comprises causing the n shares to comprise a substantially random distribution of units of data of the second data set.	181
5.	Claim 5: The method of claim 4, wherein causing the n shares to comprise a substantially random distribution comprises using a key.	181
6.	Claim 6: The method of claim 5, wherein substantially randomly distributing units of data comprises using a key.	181
7.	Claim 7: The method of claim 1, wherein producing the plurality of n shares comprises substantially randomly distributing units of data of the second data set into the plurality of n shares.	181
8.	Claim 8: The method of claim 1, wherein producing the plurality of n shares comprises distributing units of data into the plurality of n shares and substantially randomly shuffling the units of data in the shares.	185
9.	Claim 9: The method of claim 8, wherein substantially randomly shuffling the units of data comprises using a key.	187
10.	Claims 16-21 and 28-29	188
X.	CONCLUSION	189
XI.	AVAILABILITY FOR CROSS-EXAMINATION	189

XII. RIGHT TO SUPPLEMENT	189
XIII. JURAT	189

LISTING OF CLAIMS

Claim 1

[1Pre] A method performed by a computer system for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the method comprising:

[1A] reading the first data set from a memory that stores the first data set,;

[1B] processing the first data set to produce a second data set, the processing comprising:

[1B-1] encrypting the first data set using an encryption key to produce an encrypted data set, and

[1B-2] logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant,

[1B-3] wherein the second data set comprises the encrypted data set and the resultant, and wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant;

[1C] producing redundancy information based on information in the second data set;

[1D] producing a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set; and;

[1E] storing the plurality of n shares and the redundancy information, wherein at least some of the n shares are stored in separate storage locations and the first data set is recoverable using at least a threshold number, less than n , of the plurality of n shares

Claim 2

[2] The method of claim 1, wherein each of the n shares further comprises at least some of the redundancy information and storing the plurality of n shares and the redundancy information comprises storing the redundancy information in the n shares.

Claim 3

[3] The method of claim 2, wherein storing the redundancy information in the n shares comprises including in each of the n shares information redundant to information included in at least one other of the n shares.

Claim 4

[4] The method of claim 1, wherein producing the plurality of n shares comprises causing the n shares to comprise a substantially random distribution of units of data of the second data set.

Claim 5

[5] The method of claim 4, wherein causing the n shares to comprise a substantially random distribution comprises using a key.

Claim 6

[6] The method of claim 5, wherein substantially randomly distributing units of data comprises using a key.

Claim 7

[7] The method of claim 1, wherein producing the plurality of n shares comprises substantially randomly distributing units of data of the second data set into the plurality of n shares:

Claim 8

[8] The method of claim 1, wherein producing the plurality of n shares comprises distributing units of data into the plurality of n shares and substantially randomly shuffling the units of data in the shares.

Claim 9

[9] The method of claim 8, wherein substantially randomly shuffling the units of data comprises using a key.

Claim 10

[10] The method of claim 1, wherein at least some of the separate storage locations are geographically separate from one another:

Claim 11

[11] The method of claim 1, wherein logically combining comprises logically combining using an exclusive-OR operation.

Claim 12

[12] The method of claim 1, wherein the at least a portion of the encrypted data set comprises all of the encrypted data set.

Claim 13

[13Pre] A computer system for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the system comprising:

one or more processors configured to:

[13A] produce a second data set from the first data set by:

[13A-1] encrypting the first data set using an encryption key to produce an encrypted data set, and

[13A-2] logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant,

[13A-3] wherein the second data set comprises the encrypted data set and the resultant, and wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant;

[13B] produce redundancy information based on information in the second data set;

[13C] produce a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set; and

[13D] store the plurality of n shares and the redundancy information in a plurality of storage locations, wherein at least some of the n shares are stored in separate storage locations and the first data set is recoverable using at least a threshold number, less than n , of the plurality of n shares.

Claim 14

[14] The system of claim 13, wherein the one or more processors is configured to store the plurality of n shares and the redundancy information by storing the redundancy information in the n shares.

Claim 15

[15] The system of claim 14, wherein the one or more processors is configured to store the redundancy information in the n shares by including in each of the n shares information redundant to information included in at least one other of the n shares.

Claim 16

[16] The system of claim 13, wherein the one or more processors is configured to produce the plurality of n shares by causing the n shares to comprise a substantially random distribution of units of data of the second data set.

Claim 17

[17] The system of claim 16, wherein the one or more processors is configured to cause the n shares to comprise a substantially random distribution by using a key.

Claim 18

[18] The system of claim 13, wherein the one or more processors is configured to produce the plurality of n shares by substantially randomly distributing units of data of the second data set into the plurality of n shares.

Claim 19

[19] The system of claim 18, wherein the one or more processors is configured to substantially randomly distribute units of data by using a key.

Claim 20

[20] The system of claim 13, wherein the one or more processors is configured to produce the plurality of n shares by distributing units of data into the plurality of n shares and substantially randomly shuffling the units of data in the shares.

Claim 21

[21] The system of claim 20, wherein the one or more processors is configured to substantially randomly shuffle the units of data by using a key.

Claim 22

[22] The system of claim 13, wherein at least some of the separate storage locations are geographically separate from one another.

Claim 23

[23] The system of claim 13, wherein the one or more processors is configured to logically combine by logically combining using an exclusive-OR operation.

Claim 24

[24] The system of claim 13, wherein the at least a portion of the encrypted data set comprises all of the encrypted data set.

Claim 25

[25] A non-transitory computer-readable medium comprising instructions that, when executed by one or more processors, cause a computer system to carry out a method for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the method comprising:

[25A] reading the first data set from a memory that stores the first data set,

[25B] producing a second data set from the first data set, said producing comprising:

[25B-1] encrypting the first data set using an encryption key to produce an encrypted data set, and

[25B-2] logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant,

[25B-3] wherein the second data set comprises the encrypted data set and the resultant, and wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant;

[25C] producing redundancy information based on information in the second data set;

[25D] producing a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set; and

[25E] storing the plurality of n shares and the redundancy information, wherein at least some of the n shares are stored in separate storage locations and the first data set is recoverable using at least a threshold number, less than n , of the plurality of n shares.

Claim 26

[26] The non-transitory computer-readable medium of claim 25, wherein storing the plurality of n shares and the redundancy information comprises storing the redundancy information in the n shares.

Claim 27

[27] The non-transitory computer-readable medium of claim 26, wherein storing the redundancy information in the n shares comprises including in each of the n shares information redundant to information included in at least one other of the n shares.

Claim 28

[28] The non-transitory computer-readable medium of claim 25, wherein producing the plurality of n shares comprises causing the n shares to comprise a substantially random distribution of units of data of the second data set.

Claim 29

[29] The non-transitory computer-readable medium of claim 28, wherein causing the n shares to comprise a substantially random distribution comprises using a key.

Claim 30

[30] The non-transitory computer-readable medium of claim 25, wherein the at least a portion of the encrypted data set comprises all of the encrypted data set.

I, Erez Zadok, declare as follows:

1. My name is Erez Zadok.
2. I have been retained as an expert witness in the field of storage, file systems, and data security protocols on behalf of Petitioner International Business Machines Corporation (“IBM”) to provide expert opinions on the validity of U.S. Patent No. 9,135,456 (the “’456 Patent”).

I. Background and Qualifications

3. My *curriculum vitae*, which includes a more detailed summary of my background, experience and publications, is attached as EX1028.

4. I am a Professor in the Computer Science Department at Stony Brook University (part of the State University of New York (“SUNY”) system). I direct the File-systems and Storage Lab (FSL) at Stony Brook’s Computer Science Department. My research interests include file systems and storage systems, operating systems, transactional systems including database technologies, information technology and system administration, security/privacy and information assurance, networking, energy efficiency, performance and benchmarking, virtualization, cloud systems, compilers, applied machine learning, and software engineering.

5. I studied at a professional high school in Israel, focusing on electrical engineering (“EE”), and graduated in 1982. I spent one more year at the high

school's college division, receiving a special Certified Technician's degree in EE. I then went on to serve in the Israeli Defense Forces for three years (1983–1986). I received my Bachelor of Science degree in computer science ("CS") in 1991, my Master's degree in CS in 1994, and my PhD in CS in 2001—all from Columbia University in New York.

6. When I began my undergraduate studies at Columbia University, I also started working as a student assistant in the various campus-wide computer labs, eventually becoming an assistant to the head labs manager, who was managing all public computer labs on campus. During that time, I also became more involved with research within the CS Department at Columbia University, conducting research on operating systems, file and storage systems, distributed and networked systems, security, and other topics. I also assisted the CS department's computer administrators in managing the department's computers, which included storage, IT, networking, and cyber-security related duties.

7. In 1991, I joined Columbia University's CS department as a full-time systems administrator, studying towards my MS degree part-time. My MS thesis topic is related to file system reliability, fault tolerance, replication, and failover in mobile networked storage systems using file virtualization. My main duties as a systems administrator involved installing, configuring, and managing many networked servers, proxies, and desktops running several operating systems, as well

as network devices setup; this included many software and hardware upgrades, device upgrades, and BIOS firmware/chipset updates/upgrades. My duties also included ensuring reliable, secure, authenticated access to networked systems/storage and licensed software, as well as software updates, security and bug fixes. Examples of servers and their protocols included email (SMTP), file transfer (FTP), domain names (DNS), network file systems (NFS), network news systems (NNTP), and Web (HTTP).

8. In 1994, I left my systems administrator position to pursue my doctoral studies at Columbia University. My PhD thesis topic was on versatile file system development using stackable (virtualized) file systems, with examples in the fields of security and encryption, efficiency, reliability, and failover. I continued to work part-time as a systems administrator at the CS department, and eventually I was asked to serve as manager to the entire information technology (“IT”) staff. From 1991 to 2001, I was also a member of the faculty-level Facilities Committee that oversaw all IT operations at the CS department.

9. As part of my PhD studies at Columbia, I collaborated on projects to develop advanced AI-like techniques to detect previously unknown viruses (a.k.a. “zero-day malware”), using data mining and rule-based detection. This work led to several highly cited papers (over 1,600 citations for one of the papers alone) and two patents. I also became a Teaching Assistant (“TA”) for a first-ever Computer

Security course given at Columbia University's CS department with Dr. Matt Blaze as instructor.

10. From 1990 to 1998, I consulted for SOS Corporation and HydraWEB Technologies as a systems administrator and programmer, managing data storage use and backup/restore duties, databases, web servers, as well as information assurance and cyber-security (*e.g.*, malware protection, software licensing). From 1994 to 2000, I led projects at HydraWEB Technologies, and then became the Director of Software Development—overseeing the development of several products and appliances such as stateful firewalls and HTTP load-balancers, utilizing network-virtualization and high-availability techniques. From 2009 to 2019, I have consulted for Packet General Networks, a startup specializing in secure, virtualized, network storage and applications' data security in the cloud.

11. In 2001, I joined the faculty of Stony Brook University, a position I have held since that time. In 2002, I joined the Operations Committee, which oversees the IT operations of the CS department at Stony Brook University. From 2006 to 2010, I was the Director of IT Operations of the CS department. My day-to-day duties included setting policies regarding computing, hiring and training new staff, assisting any staff with topics of my specialty, defining requirements for new software/hardware, and purchasing. From 2010 to 2015, I had served as the Co-Chair to the Operations Committee. From 2016 to 2019, I oversaw the IT Operations

as the Chair of the Operations Committee. A significant component of these duties included defining and helping implement policies for data management, to ensure the security of users and their data, and data reliability and availability, while minimizing the inconvenience and performance impact to users. I personally helped setup and maintain an initial virtual-host infrastructure in the department. Since late 2019, I've been a member of the department's Executive Committee that also oversees all IT operations.

12. In 2017, I became the department's Graduate Academic Adviser, advising all Master students (over 400 annually on average) and many other graduate students on an assortment of academic matters. In 2024, I took over as the department's Graduate Program Director, overseeing the entire graduate CS program (700-800 students annually on average).

13. Since 2001, I have personally configured and managed my own research lab's network. This includes setting up and configuring multiple storage systems (*e.g.*, NFS, CIFS/SMB, NAS), virtual and physical environments, applications such as database (*e.g.*, MySQL, PostgreSQL), Web servers (*e.g.*, Apache), and mail servers; user access control (*e.g.*, NIS, LDAP), backups and restores, snapshot policies, and more. I have personally installed, configured, changed, replaced parts, and upgraded components in numerous devices including mobile devices, laptops, desktops, and servers, both physical and virtual.

14. Since 1995, I have taught courses on operating systems, storage and file systems, advanced systems programming in Unix/C, systems administration, data structures, data/software security, and more. My courses often use storage, file systems, distributed systems, and system/network security as key teaching principles and practical examples for assignments and projects. I have taught these concepts and techniques to my students, both to my direct advisees as well as in my courses. For example, in my graduate Operating Systems course, I often cover Linux's kernel mechanisms to protect users, applications, and data files, virtual file systems, as well as distributed storage systems (*e.g.*, NFS). And in the System Administration undergraduate course, I covered many topics such as networking, storage, backups, and configuring complex applications such as mail, web, and database servers.

15. My research often investigates computer systems from many angles: security, efficiency, energy use, scalability, reliability, portability, survivability, usability, ease-of-use, versatility, flexibility, and more. My research gives special attention to balancing five often-conflicting aspects of computer systems: performance, reliability, energy use, security, and ease-of-use.

16. Since joining Stony Brook University in 2001, my group in the File-systems and Storage Lab ("FSL") has developed many file systems and operating system extensions; examples include a highly-secure cryptographic file system, a portable copy-on-write ("COW") versioning file system, a tracing file system useful

to detect intrusions, a replaying file system useful for forensics, a snapshotting and sandboxing file system, a namespace unification file system (that uses stackable, virtualized, file-based COW), an anti-virus file system, an integrity-checking file system, a load balancing and replication/mirroring file system, network file system extensions for security and performance, distributed secure cloud-based storage systems, transactional key-value stores and file systems, OS-level embedded databases, a compiler to convert user-level C code to in-kernel efficient yet safe code, GCC plugins, stackable file system templates, and a Web-based backup system. Many of these projects used one form of virtualization or another (storage, network, host, etc.). I continue to maintain and release newer versions of some of these file systems and software.

17. I have published over 120 refereed publications (in ACM, IEEE, USENIX, and more). To date, my publications have been cited more than 10,000 times (as per Google Scholar as of June 13, 2025). My papers cover a wide range of related technologies such as file systems, storage systems, transactional systems, security, clouds and virtualization, performance benchmarking and optimization, energy efficiency, system administration, web systems, and more. I also published a book titled “Linux NFS and Automounter Administration” (Sybex, 2001), covering systems administration topics related to network storage and data security.

18. Some of my research has led to public software releases that have been used worldwide. I have publicly maintained the Amd Berkeley Automounter in a package called “am-utils” since 1992; this software helps administrators manage the multitude of file system mounts on dozens of different Unix systems, especially helping to automate access to multiple NFS/NAS storage volumes. Since 1997, I have maintained and released several stackable (virtualized) file system software projects for Linux, FreeBSD, and/or Sun Solaris, in a package called FiST. One of my stackable file system encryption projects, called Cryptfs, became the basis for IBM’s public release of eCryptfs, now part of Linux. Packet General Networks, for whom I have provided consulting services between 2009 and 2019, licensed another encryption file system called Ncryptfs. Another popular file system released in 2003, called Unionfs, offers virtual namespace unification, transparent shadow copying (a.k.a. copy-on-write or COW), file system snapshotting (*e.g.*, useful for forensics and disaster recovery), and the ability to save disk space by sharing a read-only copy of data across several computers, among other features.

19. My research and teaching make extensive use of data security features. For example, each time I taught the graduate operating system course, the first homework assignment includes the creation of a new system call that performs new or added functionality, often for encrypting a file or verifying its integrity; many of my other assignments cover topics of user/process access control, anti-virus filtering,

and more. Since 2001, over 1,000 graduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

20. Moreover, in an undergraduate course titled “Advanced Systems Programming in Unix/C,” I cover many topics of system security and vulnerabilities, such as the structure of UNIX processes, and memory segments such as the heap and stack. This course covers details of several hundred Linux system calls. Often, the first assignment for this course is to develop a tool to encrypt/decrypt files using advanced ciphers, use digital signatures to certify the cipher keys used, and reliably recover files in case of failures. Since 2001, several hundred undergraduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

21. In another undergraduate course, System Administration, I taught network configuration, security, and storage configuration and reliability. In a special topics course on Storage Systems, I covered many topics such as data deduplication, RAID, transactional storage, storage hardware including modern Flash based ones, virtual storage, backup/restore, snapshots and continuous data protection (“CDP”), NAS and SAN, and NFS.

22. Overall, in addition to the aforementioned experience, my technical experience relevant to the ’456 Patent at the time of the alleged invention included the following: I studied, researched, developed code, published, taught, and/or

released software for the following: encryption and integrity software using logical operations such as XOR; encryption file systems including my own (Cryptfs and a licensed NCryptfs); file system integrity using CRC codes and hashes such as MD4 and MD5; digital signatures; RAID software with parity codes such that data can be recovered from a subset of the data copies; distributed and networked storage and file systems (*e.g.*, NFS, SMB/CIFS); a mobile, networked file system that where clients can move across large geographic distances, then measure, locate, and [re]connect to the fastest responding file systems (this was part of my MS Thesis); and a file system that splits files across multiple locations.

23. My research has been supported by many federal and state grants as well as industry awards, including an NSF CAREER award, two IBM Faculty awards, two NetApp Faculty awards, a Western Digital award, a Facebook award, several Dell-EMC awards, and several equipment gifts. I received the 2008 SUNY Chancellor's Excellence in Teaching award, and the 2022 SUNY Chancellor's Award for Excellence in Scholarship and Creative Activities (both awards can be given only once in a lifetime). In 2021, I was named an ACM Distinguished Member for "Outstanding Scientific Contributions to Computing."

24. My service record to the community includes serving as the co-chair for the USENIX Annual Technical Conference in 2020 (ATC'20); serving as the co-chair for USENIX File and Storage Technologies (FAST'15) in 2015 and on the

FAST Conference Steering Committee from 2015 to 2023; serving on the ACM HotStorage Steering Committee since 2021; and serving as the co-chair in 2012 and on the Steering Committee of the ACM SYSTOR conference since 2012. I have served as an Associate Editor to the ACM Transactions on Storage (“TOS”) journal from 2009 to 2022; in 2022, I was named the Editor-in-Chief for ACM’s TOS journal.

25. I am a named inventor on four patents, two titled “Systems and Methods for Detection of New Malicious Executables” (U.S. Patent No. 7,487,544, issued February 3, 2009; and U.S. Patent No. 7,979,907, issued July 12, 2011); and two more titled “Multi-Tier Caching,” (U.S. Patent No. 9,355,109, issued May 31, 2016; and U.S. Patent 9,959,279, issued May 1, 2018).

26. I have been disclosed as a testifying expert in 22 cases (including *inter partes* review (“IPR”) proceedings) in the past four years. I have been deposed 13 times and testified in trial twice.

27. I have personal knowledge of the facts and opinions discussed in this declaration and believe them to be true. If called upon to do so, I would testify competently thereto. I have been warned that willful false statements and the like are punishable by fine or imprisonment, or both. I have had no contact with any of the named inventors of the ’456 Patent.

28. My consulting company, Zadoks Consulting Services, is being compensated for my time at my current standard consulting rate. I am also being reimbursed for expenses that I may incur during the course of this work. My compensation is not contingent upon the results of my study and analysis, the substance of my opinions, or the outcome of any proceeding involving the Challenged Claim(s).

29. I have no financial interest in the Petitioner. I have no financial interest in the outcome of this matter or in any litigation involving the '456 patent.

II. Materials Considered

30. I understand that the '456 Patent has a filing date of August 29, 2014, claiming priority to U.S. Provisional Application Nos. 60/622,146 (the "'146 App."), filed October 25, 2004, and 60/718,185 (the "'185 App."), filed September 16, 2005. EX1001 ('456 Patent); EX1014 ('146 App.); EX1015 ('185 App.).

31. Each claim of the '456 Patent requires "logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant." *See* EX1001 ('456 Patent), Claims 1, 13, 25. The "logically combining" limitation is disclosed no earlier than the '185 App., filed on September 16, 2005. EX1015 ('185 App.), 211. But I understand that even assuming that the '456 Patent is entitled to the earliest possible priority date—October 25, 2004—the references identified below are prior art. I have considered issues from the perspective of a person of

ordinary skill in the art as described in §VII below as of October 25, 2004, but my opinion would hold equally even if the priority date of September 16, 2005 is assumed.

32. I have reviewed the following documents in preparing this declaration. I understand that these are prior art to the '456 Patent.

- U.S. Patent Application Publication No. 2003/0065656 (“Torre”) (EX1003), published on April 3, 2003.
- “The Security of All-Or-Nothing Encryption: Protecting Against Exhaustive Key Search,” by Anand Desai, Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, LNCS 1880, 359–375, 2000 (“Desai”) (EX1004), published on August 7, 2000.
- U.S. Patent Application Publication No. 2003/0091191A1 (“Watanabe”) (EX1005), published on March 15, 2003.
- U.S. Patent Application Publication No. 2004/0049687 (“Orsini”) (EX1006), published on March 11, 2004.

33. I have also reviewed all of the exhibits cited in this declaration, including the following:

Exhibit No.	DESCRIPTION
1001	U.S. Patent No. 9,135,456 ('456 Patent)
1003	U.S. Patent Publication No. 2003/0065656 (“Torre”)

Exhibit No.	DESCRIPTION
1004	Anand Desai, “The Security of All-Or-Nothing Encryption: Protecting Against Exhaustive Key Search,” <i>Advances in Cryptology - CRYPTO 2000</i> , 20th Annual International Cryptology Conference, LNCS 1880, 359–375, 2000 (“Desai”)
1005	U.S. Patent Application Publication No. 2003/0091191A1 (“Watanabe”)
1006	U.S. Patent Publication No. 2004/0049687 (“Orsini”)
1007	Declaration of Dr. Sylvia Hall-Ellis (“Hall-Ellis Decl.”)
1008	File History of U.S. Patent No. 9,135,456 (“456 File History”)
1009	File History of U.S. Application No. 13/371,363 (“363 File History”)
1010	Ronald L. Rivest, “All-or-Nothing Encryption and the Package Transform,” <i>Fast Software Encryption ’97</i> , Lecture Notes in Computer Science Vol. 1267, E. Biham ed., Springer-Verlag, 1997 (“Rivest”)
1011	Doug Burger, “Memory Systems,” <i>ACM COMPUTING SURVEYS</i> , Vol. 28, No. 1, March 1996 (“Burger”)
1012	“Understanding Computers: An Overview for Records and Archives Staff,” <i>INTERNATIONAL RECORDS MANAGEMENT TRUST</i> , 1999 (“Understanding Computers”)
1013	<i>Google, LLC v. Security First Innovations, LLC</i> , IPR2024-00212, Exhibit 1043 (Patent Owner’s Proposed Claim Constructions In <i>Security First Innovations, LLC v. Google, LLC</i> , No. 2:23-cv-00097 (E.D. Va.)) (“PO’s Google Litigation Construction”)
1014	U.S. Provisional Application No. 60/622,146 (“146 App.”)
1015	U.S. Provisional Application No. 60/718,185 (“185 App.”)
1016	Microsoft Computer Dictionary, 5th ed., 2002, excerpts (“Microsoft Computer Dictionary”)
1017	Bruce Schneier, <i>Applied Cryptography</i> , 2nd ed., 1996, excerpts (“Schneier”)
1018	Yitzhak Birk, “Random RAIDs with Selective Exploitation of Redundancy for High Performance Video Servers,” <i>IEEE</i> 1997 (“Birk”)
1019	Daniel J. Costello et al., “Some Thoughts on the Equivalence of Systematic and Non-Systematic Convolutional Encoders,” August 2, 2001 (“Costello”)

Exhibit No.	DESCRIPTION
1020	P. Venkat Rangan et al., “Designing File Systems for Digital Video and Audio,” ACM SIGOPS Operating Systems Review, Volume 25, Issue 5, 81–94 (1991) (“Rangan”)
1021	Comparison of the ’456 Patent’s Specification and Orsini
1022	“Primer: Reed-Solomon Error Correction Codes (ECC),” AHA PRODUCTS GROUP (Aug. 8, 2004) (“Primer: Reed-Solomon ECC”)
1023	Norman Matloff, “Introduction to Digital Logic,” September 4, 2003 (“Matloff”)
1024	File History of U.S. Patent Application No. 11/258,839 (“’839 File History”)
1025	U.S. Patent Publication No. 2003/0016596A1 (“Chiquoine”)
1026	Wen-Jiin Tsai et al., “Dynamic Buffer Management for Near Video-On-Demand Systems,” Tsai et al., Multimedia Tools and Applications 6,61–83 (1998) (“Tsai”)
1027	John Kubiawicz, et al., “OceanStore: An Architecture for Global-Scale Persistent Storage,” ACM 2000 (“Kubiawicz”)
1028	Curriculum Vitae of Dr. Erez Zadok

III. Summary of Opinions

34. It is my opinion that claims 1–30 of the ’456 Patent are obvious in view of Grounds identified below:

Ground	References	Challenged Claims
I	Torre, Desai	1–3, 10–15, 22–27, 30.
II	Torre, Desai, Watanabe	1–3, 10–12, 25–27, 30.
III	References in Ground I or II, and Orsini	4–9, 16–21, 28–29.

IV. Relevant Law

35. I am not an attorney. For the purposes of this declaration, I have been informed about certain aspects of the law that are relevant to my opinions. My understanding of the law is as follows:

A. Claim Construction

36. I have been informed that claim construction is a matter of law and that the final claim construction will ultimately be determined by the Board. For the purposes of my analysis in this proceeding and with respect to the prior art, I have been informed that IPRs are currently reviewed under “the *Phillips* standard.”

37. I have been informed that under the *Phillips* standard, claim terms are given their plain and ordinary meaning as understood by a person of ordinary skill in the art at the time of the invention in light of the claim language and the patent specification.

38. I have been informed that embodiments described in the specification are generally encompassed by the claims.

39. I have been informed that the patentee can serve as their own lexicographer. As such, if a claim term is provided with a specific definition in the specification, then I should interpret that claim term in light of the particular definition provided by the patentee.

B. Obviousness

40. I have been informed and understand that a patent claim can be considered to have been obvious to a person of ordinary skill in the art at the time the application was filed. This means that, even if all of the requirements of a claim are not found in a single prior art reference, the claim is not patentable if the differences between the subject matter in the prior art and the subject matter in the claim would have been obvious to a person of ordinary skill in the art at the time the application was filed.

41. I have been informed and understand that a determination of whether a claim would have been obvious should be based upon several factors, including, among others:

- the level of ordinary skill in the art at the time the application was filed;
- the scope and content of the prior art;
- what differences, if any, existed between the claimed invention and the prior art; and
- any relevant objective considerations of non-obviousness.

42. I have been informed and understand that the teachings of two or more references may be combined in the same way as disclosed in the claims, if such a combination would have been obvious to one having ordinary skill in the art.

43. In determining whether a combination based on multiple references would have been obvious, it is appropriate to consider, among other factors:

- whether the teachings of the prior art references disclose known concepts combined in familiar ways, which, when combined, would yield predictable results;
- whether a person of ordinary skill in the art would have implemented a predictable variation, and would have seen the benefit of doing so;
- whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by those skilled in the art;
- whether a person of ordinary skill would have recognized a reason to combine known elements in the manner described in the claim;
- whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; and
- whether the claim applies a known technique that had been used to improve a similar device or method in a similar way.

44. I understand that one of ordinary skill in the art has ordinary creativity and is not an automaton.

45. I understand that in considering obviousness, it is important not to determine obviousness using the benefit of hindsight derived from the patent being considered.

C. Ordinary Skill in the Art

46. I understand that, in the context of an invalidity analysis, a person having ordinary skill in the art (“POSITA”) is a hypothetical person who looks to prior art at the time of the invention.

47. I further understand that the factors that may be considered in determining the level of ordinary skill in the art include: (1) the problems

encountered in the art; (2) the prior art solutions to the problems encountered in the art; (3) the rapidity of innovation; (4) the sophistication of the technology; and (5) the education level of active workers in the field. I understand that these factors need not all be taken into account for the analysis and that one or more of these factors may control.

V. Technology Overview

48. In this section, I provide a brief overview of the background art as understood by a POSITA as of the earliest possible priority date of the '456 Patent, *i.e.*, September 16, 2005.

A. General Computer Operations

49. Computers are hardware and software devices that execute instructions. The hardware often includes a central processing unit (“CPU”) or other microprocessors, typically called “processors,” required volatile memory (*e.g.*, RAM (random-access memory), or DRAM (dynamic random-access memory)), non-volatile (persistent) storage (*e.g.*, an internal hard disk or flash drive), and input/output (“I/O”) devices (*e.g.*, network interface card, mouse, keyboard, display). RAM, hard disks, and flash drives are considered, generally, “storage” or “memory” devices because they can store and hold bits of information (whether persistently or not). Computers can have multiple storage devices. Processors are

coupled to main memory using a memory bus and coupled to peripherals such as hard disks and network interface cards using one or more I/O busses.

50. Memory and storage devices have considerable differences in their speeds, capacities, and costs. Non-volatile storage devices such as hard disks have the largest capacity, are least expensive per gigabyte, and are slowest (typically operating at millisecond speeds). Volatile DRAM devices are smaller, more expensive, and faster than hard disks (typically operating at microsecond speeds). A CPU's own memory caches and operations are the smallest, most expensive, and fastest (typically operating at nanosecond speeds). Overall, CPUs are about 1,000× faster than DRAM, which in turn is about 1,000× faster than hard disks.

51. Computer programs are often written in a high-level human-readable language (*e.g.*, C, C++, Java, Perl, PHP, Python), then translated using a compiler or script processor to machine instructions understood by the CPU (*e.g.*, Intel or AMD processor). Machine instructions—generally called “software”—are stored in files on persistent media (*e.g.*, HDD or ROM), loaded into DRAM, and then loaded into the CPU where they can be executed. All modern computers operate in this manner, regardless of their purpose: a small laptop, a server in a data center, a mainframe computer, a personal workstation, a proxy device, a gateway, a firewall, a router, an appliance, a virtual machine, a handheld or mobile device, a smart card,

an SD card, etc. Computer systems may run multiple software programs, from background servers and daemons to user applications.

52. Software itself may be distributed or located in a single location, use various modules or units such as libraries and other components. In general, “software” is “[c]omputer programs; instructions that make hardware work. Two main types of software are system software (operating systems), which controls the workings of the computer, and applications, such as word processing programs, spreadsheets, and databases, which perform the tasks for which people use computers. Two additional categories, which are neither system nor application software but contain elements of both, are network software, which enables groups of computers to communicate, and language software, which provides programmers with the tools they need to write programs. In addition to these task-based categories, several types of software are described based on their method of distribution. These include packaged software (canned programs), sold primarily through retail outlets; freeware and public domain software, which are distributed free of charge; shareware, which is also distributed free of charge, although users are requested to pay a small registration fee for continued use of the program; and vaporware, software that is announced by a company or individuals but either never makes it to market or is very late. *See also* application, canned software, freeware, network software, operating system, shareware, system software, vaporware.

Compare firmware, hardware, liveware.” EX1016 (Microsoft Computer Dictionary), 489.

53. Users (including administrators) can login to one or more computers (*e.g.*, desktop, laptop, server) and execute one or more applications (*e.g.*, word processing, web browsing, backup tool). Some computers offer external interfaces and APIs to communicate with certain devices and especially peripherals. For example, both a USB storage device as well as a printer use a well-defined “protocol” to allow users to communicate with them through the main computer.

54. All computers include internal volatile memory (*e.g.*, RAM). A “primary storage” (or “internal memory”) is “Random access memory (RAM); the main general-purpose storage region to which the microprocessor has direct access. A computer’s other storage options, such as disks and tape, are called secondary storage or (sometimes) backing storage.” EX1016 (Microsoft Computer Dictionary), 281, 419. A “secondary storage” is “[a]ny data storage medium other than a computer’s random access memory (RAM)—typically tape or disk. *Compare* primary storage.” EX1016 (Microsoft Computer Dictionary), 469.

55. A “file server” is “[a] file-storage device on a local area network that is accessible to all users on the network. Unlike a disk server, which appears to the user as a remote disk drive, a file server is a sophisticated device that not only stores files but manages them and maintains order as network users request files and make

changes to them. To deal with the tasks of handling multiple—sometimes simultaneous—requests for files, a file server contains a processor and controlling software as well as a disk drive for storage. On local area networks, a file server is often a computer with a large hard disk that is dedicated only to the task of managing shared files. *Compare* disk server.” EX1016 (Microsoft Computer Dictionary), 212–213.

56. File servers can transmit audio/video files. A “video server” is “[a] server designed to deliver digital video-on-demand and other broadband interactive services to the public over a wide area network.” EX1016 (Microsoft Computer Dictionary), 552.

57. There are many audio/video file formats, comprising a sequence of segments or frames. For example, the “.mpeg” format designates “[t]he file extension that identifies video and sound files compressed in the MPEG format specified by the Moving Pictures Experts Group. *See also* MPEG.” EX1016 (Microsoft Computer Dictionary), 350. “MPEG-2” is “[a]n extension of the MPEG-1 standard designed for broadcast television, including HDTV. MPEG-2 defines a higher bandwidth of up to 40 Mbps, five audio channels, a wider range of frame sizes, and interlaced video. *See also* HDTV, MPEG (definition 1). *Compare* MPEG-1, MPEG-3, MPEG-4.” EX1016 (Microsoft Computer Dictionary), 350.

58. Similarly, an “.avi” is “[t]he file extension that identifies an audiovisual interleaved data file in the Microsoft RIFF format.” EX1016 (Microsoft Computer Dictionary), 46. An “AVI” denotes the “[a]cronym for **A**udio **V**ideo **I**nterleaved. A Windows multimedia file format for sound and moving pictures that uses the Microsoft RIFF (Resource Interchange File Format) specification.” EX1016 (Microsoft Computer Dictionary), 46.

B. Networking

59. A Local Area Network (LAN) is typically a network at a local site where network latencies are short and physical distances between computers are small. A LAN and its computers are often protected from the rest of the Internet using a firewall. A LAN is “Acronym for **l**ocal **a**rea **n**etwork. A group of computers and other devices dispersed over a relatively limited area and connected by a communications link that enables any device to interact with any other on the network. LANs commonly include PCs and shared resources such as laser printers and large hard disks. The devices on a LAN are known as nodes, and the nodes are connected by cables through which messages are transmitted. *See also* baseband network, broadband network, bus network, collision detection, communications protocol, contention, CSMA/CD, network, peer-to-peer architecture, ring network, star network. *Compare* WAN.” EX1016 (Microsoft Computer Dictionary), 304, 315.

60. Another form of a local area network is one that operates inside a single computer. A “localhost” is “[t]he name that is used to represent the same computer on which a TCP/IP message originates. An IP packet sent to localhost has the IP address 127.0.0.1 and does not actually go out to the Internet. *See also* IP address, packet (definition 1), TCP/IP.” EX1016 (Microsoft Computer Dictionary), 316. Often, when users/processes want to communicate with other users/processes, using TCP/IP communications on the same machine, they can use the localhost address: it is considered a trusted, high-speed internal network.

61. Conversely, a Wide-Area Network (WAN) refers to computers across the entire Internet, which are often distant from each other, exhibit longer latencies, and afford little protection. A “wide area network” is “[a] geographically widespread network, one that relies on communications capabilities to link the various network segments. A WAN can be one large network, or it can consist of a number of linked LANs (local area networks).” EX1016 (Microsoft Computer Dictionary), 561, 566.

62. A firewall typically intercepts all traffic between computers in a LAN and the WAN, monitors and controls access, and can filter out undesired traffic, and thus acts as a proxy device. Often, LAN computers are far more trusted than WAN computers; therefore, a firewall can be said to separate the trusted network (LAN or localhost) from the untrusted network (WAN). EX1016 (Microsoft Computer

Dictionary), 214-215 (“**firewall** *n.* A security system intended to protect an organization’s network against external threats, such as hackers, coming from another network, such as the Internet. Usually a combination of hardware and software, a firewall prevents computers in the organization’s network from communicating directly with computers external to the network and vice versa. Instead, all communication is routed through a proxy server outside of the organization’s network, and the proxy server decides whether it is safe to let a particular message or file pass through to the organization’s network. *See also* proxy server.”).

63. A server is a computer or software that provides a service to other computers, called clients. There are many servers on the Internet, for example, Web servers and Mail servers. EX1016 (Microsoft Computer Dictionary), 474 (“**server** *n.* 1. On a local area network (LAN), a computer running administrative software that controls access to the network and its resources, such as printers and disk drives, and provides resources to computers functioning as workstations on the network. 2. On the Internet or other network, a computer or program that responds to commands from a client. For example, a file server may contain an archive of data or program files; when a client submits a request for a file, the server transfers a copy of the file to the client. *See also* client/server architecture. *Compare* client (definition 3).”

64. A client is typically an end-point computer or software, sometimes used by a user, that communicates across LAN and WAN networks; a client may contact servers to request some service. A user's desktop, laptop, and handheld device are examples of clients. (Note that a computer can be both a client and a server, a form of a proxy.) EX1016 (Microsoft Computer Dictionary), 102 (“**client** *n.* [] 2. A process, such as a program or task, that requests a service provided by another program—for example, a word processor that calls on a sort routine built into another program. The client process uses the requested service without having to ‘know’ any working details about the other program or the service itself. *Compare* child (definition 1), descendant (definition 2). 3. On a local area network or the Internet, a computer that accesses shared network resources provided by another computer (called a *server*). *See also* client/server architecture, server.”).

65. A client/server architecture describes how client computers interact with server computers. EX1016 (Microsoft Computer Dictionary), 102 (“**client/server architecture** *n.* An arrangement used on LANs (local area networks) that makes use of distributed intelligence to treat both the server and the individual workstations as intelligent, programmable devices, thus exploiting the full computing power of each. This is done by splitting the processing of an application between two distinct components: a ‘front-end’ client and a ‘back-end’ server. The client component is a complete, stand-alone personal computer (not a ‘dumb’

terminal), and it offers the user its full range of power and features for running applications. The server component can be a personal computer, a minicomputer, or a mainframe that provides the traditional strengths offered by minicomputers and mainframes in a time-sharing environment: data management, information sharing between clients, and sophisticated network administration and security features. The client and server machines work together to accomplish the processing of the application being used. Not only does this increase the processing power available over older architectures but it also uses that power more efficiently. The client portion of the application is typically optimized for user interaction, whereas the server portion provides the centralized, multiuser functionality. *See also* distributed intelligence.”).

1. Networked and Distributed Storage Systems

66. File systems can appear as local volumes (*e.g.*, the C:\ drive in Windows), but they can also be networked, where the physical storage resides on a remote file server. Network file systems allow files and volumes to be shared and accessible from multiple networked computers. EX1016 (Microsoft Computer Dictionary), 363 (“**Network File System** *n.* *See* NFS.”), 365 (“**NFS** *n.* Acronym for **Network File System**. A distributed file system that allows users to access remote files and directories on a network as if they were local. NFS is compatible

with Microsoft Windows and UNIX-based systems, including Linux and Mac OS X.”).

67. NFS is an example of a network-attached storage, used to hold, transmit, and stream any files. A “NAS” is “[a]cronym for Network-Attached Storage. A platform-independent storage appliance connected to a network. NAS uses a storage unit with a built-in server that can communicate with clients over a network. NAS devices are popular for ease of maintenance, manageability, and scalability. *Compare* SAN.” EX1016 (Microsoft Computer Dictionary), 357, 362.

68. For larger, more demanding network storage systems, a storage-area network is used. A “storage area network” or “SAN” is “[a] high-speed network that provides a direct connection between servers and storage, including shared storage, clusters, and disaster-recovery devices. A storage area network, or SAN, includes components such as hubs and routers that are also used in local area networks (LANs), but it differs in being something of a ‘subnetwork’ dedicated to providing a high-speed connection between storage elements and servers. Most SANs rely on fiber-channel connections that deliver speeds up to 1000 Mbps and can support up to 128 devices. SANs are implemented to provide the scalability, speed, and manageability required in environments that demand high data availability. *Acronym: SAN. Also called: system area network.*” EX1016 (Microsoft Computer Dictionary), 463, 498-499.

69. A NAS, NFS, or SAN storage can be used by any computer over a network. Such network storage is designed for sharing, or for multiple computers to access the same network storage concurrently. However, it is also possible for such network storage or portion of it to be dedicated for use by a single computer. In that case, such network storage acts similarly to a dedicated internal storage such as an internal hard disk.

70. When the amount of data to access grew large enough, and especially if the data had to be concurrently accessed from multiple users/applications, storage researchers and companies leveraged the principles of distributed systems to improve the performance, resiliency, and fault-tolerance of the storage system.

71. A single computer's performance is limited by the physical maximum speeds of its network, processor, memory, and I/O devices. A distributed system, however, improves overall performance by using multiple computers concurrently to process many requests, thus increasing the aggregate throughput manyfold.

72. A single computer also represents a single point of failure: if that one computer is down, off, or broken into, then the service it offers is unavailable. Even regular networks can fail and packets get lost. A distributed system, conversely, improves resiliency and security by ensuring that the failure of any one (or even a few) components do not affect the overall service provided. This principle is called "fault tolerance." EX1016 (Microsoft Computer Dictionary), 207 ("**fault tolerance**

n. The ability of a computer or an operating system to respond to a catastrophic event or fault, such as a power outage or a hardware failure, in a way that ensures that no data is lost and any work in progress is not corrupted. This can be accomplished with a battery-backed power supply, backup hardware, provisions in the operating system, or any combination of these. In a fault-tolerant network, the system has the ability either to continue the system's operation without loss of data or to shut the system down and restart it, recovering all processing that was in progress when the fault occurred.”).

73. NFS is but one example of a networked or distributed file system. Many complex software systems have been designed to operate in a distributed manner, where they are more resilient to failures and scale better than software running on a single node.

74. Generally, a “distributed network” is “[a] network in which processing, storage, and other functions are handled by separate units (nodes) rather than by a single main computer.” EX1016 (Microsoft Computer Dictionary), 168.

75. The “Distributed Computing Environment” is “[a] set of standards from the Open Group (formerly the Open Software Foundation) for development of distributed applications that can operate on more than one platform. *Acronym*: DCE. *See also* distributed processing.” EX1016 (Microsoft Computer Dictionary), 167.

76. “[D]istributed processing” is “[a] form of information processing in which work is performed by separate computers linked through a communications network. Distributed processing is usually categorized as either plain distributed processing or true distributed processing. Plain distributed processing shares the workload among computers that can communicate with one another. True distributed processing has separate computers perform different tasks in such a way that their combined work can contribute to a larger goal. The latter type of processing requires a highly structured environment that allows hardware and software to communicate, share resources, and exchange information freely.” EX1016 (Microsoft Computer Dictionary), 168.

77. Finally, a “distributed file system” is “[a] file management system in which files may be located on multiple computers connected over a local or wide area network.” EX1016 (Microsoft Computer Dictionary), 168.

C. Data Security and Privacy

78. All modern systems include measures of security to protect their data from unauthorized access. EX1016 (Microsoft Computer Dictionary), 212 (“**file protection** *n.* A process or device by which the existence and integrity of a file are maintained. Methods of file protection range from allowing read-only access and assigning passwords to covering the write-protect notch on a disk and locking away floppy disks holding sensitive files.”).

79. The POSIX standard, first started in the late 1980s, codifies the access application program interfaces (APIs) that operating systems export to user applications. EX1016 (Microsoft Computer Dictionary), 414 (“**POSIX** *n.* Acronym for **P**ortable **O**perating **S**ystem **I**nterface for **U**NIX. An Institute of Electrical and Electronics Engineers (IEEE) standard that defines a set of operating-system services. Programs that adhere to the POSIX standard can be easily ported from one system to another. POSIX was based on UNIX system services, but it was created in a way that allows it to be implemented by other operating systems.”).

80. A simple data protection scheme defines whether users can only read files, only write files, or have both read and write access. EX1016 (Microsoft Computer Dictionary), 398 (“**permission** *n.* In a networked or multiuser computer environment, the ability of a particular user to access a particular resource by means of his or her user account. Permissions are granted by the system administrator or other authorized person. Several levels of access can be given: read only, read and write (view and change), or read, write, and delete. *Also called:* Access permission.”).

81. In the basic POSIX model, for example, a file has three sets of permissions (read, write, or execute) for three classes of users (owner, group member, and everyone else).

82. More sophisticated schemes generalize these simpler permission models to allow for lists of users, lists of groups, as well as logical operations on lists (*e.g.*, negation, conjunction, and disjunction). These are called “access control lists,” or ACLs. EX1016 (Microsoft Computer Dictionary), 14 (“**access control** *n.* The mechanisms for limiting access to certain items of information or to certain controls based on users’ identities and their membership in various predefined groups. Access control is typically used by system administrators for controlling user access to network resources, such as servers, directories, and files. *See also* access privileges, system administrator.”), 14 (“**access control list** *n.* A list associated with a file that contains information about which users or groups have permission to access or modify the file. *Acronym:* ACL.”). ACLs existed in Unix systems since at least the 1990s (and I personally used them, see §I above).

83. Another common technique, useful to protect user’s privacy, is “encryption”, or “[t]he process of encoding data to prevent unauthorized access, especially during transmission. Encryption is usually based on one or more keys, or codes, that are essential for decoding, or returning the data to readable form. The U.S. National Bureau of Standards created a complex encryption standard, Data Encryption Standard (DES), which is based on a 56-bit variable that provides for more than 70 quadrillion unique keys to encrypt documents. *See also* DES.” EX1016 (Microsoft Computer Dictionary), 142, 192.

84. To “encrypt” is “[t]o encode (scramble) information in such a way that it is unreadable to all but those individuals possessing the key to the code. Encrypted information is known as cipher text. *Also called:* encipher, encode.” EX1016 (Microsoft Computer Dictionary), 192.

85. A “block cipher” is “[a] private key encryption method that encrypts data in blocks of a fixed size (usually 64 bits). The encrypted data block contains the same number of bits as the original. *See also* encryption, private key.” EX1016 (Microsoft Computer Dictionary), 65.

86. “DES” is an “[a]cronym for Data Encryption Standard. A specification for encryption of computer data developed by IBM and adopted by the U.S. government as a standard in 1976. DES uses a 56-bit key. *See also* encryption, key (definition 3).” EX1016 (Microsoft Computer Dictionary), 142, 153.

87. Most encryption schemes use an “encryption key,” “[a] sequence of data that is used to encrypt other data and that, consequently, must be used for the data’s decryption. *See also* decryption, encryption.” EX1016 (Microsoft Computer Dictionary), 192.

88. Some encryption schemes do not use a key, such as “ROT13 encryption,” “[a] simple encryption method in which each letter is replaced with the letter of the alphabet 13 letters after the original letter, so that A is replaced by N, and so forth; N, in turn, is replaced by A, and Z is replaced by M. ROT13 encryption

is not used to protect messages against unauthorized readers; rather, it is used in newsgroups to encode messages that a user may not want to read, such as sexual jokes or spoilers. Some newsreaders can automatically perform ROT13 encryption and decryption at the touch of a key.” EX1016 (Microsoft Computer Dictionary), 458.

89. Even schemes that use a key require that the key be protected to maintain privacy. The principle of “privacy” is “[t]he concept that a user’s data, such as stored files and e-mail, is not to be examined by anyone else without that user’s permission. A right to privacy is not generally recognized on the Internet. Federal law protects only e-mail in transit or in temporary storage, and only against access by Federal agencies. Employers often claim a right to inspect any data on their systems. To obtain privacy, the user must take active measures such as encryption. *See also* encryption, PGP, Privacy Enhanced Mail. *Compare* security.” EX1016 (Microsoft Computer Dictionary), 422.

90. Indeed, in public-key cryptography, the “private key” is “[o]ne of two keys in public key encryption. The user keeps the private key secret and uses it to encrypt digital signatures and to decrypt received messages. *See also* public key encryption. *Compare* public key.” EX1016 (Microsoft Computer Dictionary), 422.

91. There are many forms of verification that can be applied to data files. One example is to verify the integrity of data using cryptographic hashing algorithms

such as SHA. “SHA” is an “[a]cronym for Secure Hash Algorithm. A technique that computes a 160-bit condensed representation of a message or data file, called a *message digest*. The SHA is used by the sender and the receiver of a message in computing and verifying a digital signature, for security purposes. *See also* algorithm, digital signature.” EX1016 (Microsoft Computer Dictionary), 477. The term “data integrity” represents “[t]he accuracy of data and its conformity to its expected value, especially after being transmitted or processed.” EX1016 (Microsoft Computer Dictionary), 143.

92. Users can also verify the authenticity of files’ sources using digital signatures. A “digital signature” is “[a] security mechanism used on the Internet that relies on two keys, one public and one private, that are used to encrypt messages before transmission and to decrypt them on receipt.” EX1016 (Microsoft Computer Dictionary), 159.

93. Digital signatures often combine cryptographic techniques and hashing algorithms to enable bi-directional authentication and secure communications. “Public key cryptography” or “public key encryption” is “[a]n asymmetric scheme that uses a pair of keys for encryption: the public key encrypts data, and a corresponding secret key decrypts it. For digital signatures, the process is reversed: the sender uses the secret key to create a unique electronic number that can be read by anyone possessing the corresponding public key, which verifies that the message

is truly from the sender. *See also* private key, public key.” EX1016 (Microsoft Computer Dictionary), 429.

94. In fact, digital signatures were so common that in 2000 the U.S. Government published a standard describing their use. The “Digital Signature Standard” is “[a] public key cryptographic standard issued in 1994 by the United States National Institute of Standards and Technology (NIST) to authenticate electronic documents. The DSS uses a Digital Signature Algorithm (DSA) to generate and verify digital signatures based on a public key, which is not secret, and a private key, which is known or held only by the person generating the signature. A digital signature serves to authenticate both the identity of the signer and the integrity of the transmitted information. *Acronym: DSS. See also* public key encryption.” EX1016 (Microsoft Computer Dictionary), 159-160.

95. Data protection techniques such as encryption often use XOR operations. The “exclusive OR” or “XOR” is “[a] Boolean operation that yields ‘true’ if and only if one of its operands is true and the other is false. See the table. *Acronym: EOR. Also called: XOR. See also* Boolean operator, truth table. *Compare* AND, OR.” EX1016 (Microsoft Computer Dictionary), 199-200, 579 (including table below).

Table E.1 Exclusive OR.

<i>a</i>	<i>b</i>	<i>a XOR b</i>
0	0	0
0	1	1
1	0	1
1	1	0

96. Indeed “XOR encryption” is “[s]hort for Exclusive-OR encryption. A simple encryption scheme using the ‘exclusive-or’ concept, in which a decision is based on only one of two conditions being met. Using a provided key, XOR encryption performs an exclusive-or process on each byte of data to be encrypted. Because XOR encryption is not a strong security tool used alone, it is typically used as an additional level of security for Internet transmission of sensitive information.” EX1016 (Microsoft Computer Dictionary), 579.

1. Hash Functions and Their Uses

97. Hashing algorithms are mathematical functions that take as input (typically) a long piece of data and produce a shorter output number that can (almost always) uniquely identify the input data. Depending on the specific context, hash functions are sometimes called checksums, fingerprints, message digests, or digital signature functions.

98. Hash functions are one-way: it is typically considered cryptographically hard to produce the original input data from the hash itself (*e.g.*, invert it). *See* EX1016 (Microsoft Computer Dictionary), 247-248 (“**hash**² *vb.* To

be mapped to a numerical value by a transformation known as a hashing function. Hashing is used to convert an identifier or key, meaningful to a user, into a value for the location of the corresponding data in a structure, such as a table. []”), 477 (“**SHA** *n.* Acronym for **Secure Hash Algorithm**. A technique that computes a 160-bit condensed representation of a message or data file, called a *message digest*. The SHA is used by the sender and the receiver of a message in computing and verifying a digital signature, for security purposes. *See also* algorithm, digital signature.”), 133 (“**CRC** *n.* Acronym for **cyclical (or cyclic) redundancy check**. A procedure used in checking for errors in data transmission. CRC error checking uses a complex calculation to generate a number based on the data transmitted. The sending device performs the calculation before transmission and includes it in the packet that it sends to the receiving device. The receiving device repeats the same calculation after transmission. If both devices obtain the same result, it is assumed that the transmission was error free. The procedure is known as a redundancy check because each transmission includes not only data but extra (redundant) error-checking values. Communications protocols such as XMODEM and Kermit use cyclical redundancy checking.”).

99. Hash functions have been used for many years in various settings from security, cryptography, integrity, data deduplication, and others. For example, the Tripwire system I personally used in the early 1990’s hashes files and their meta-

data and compares those hashes against the files later on; if a mismatch is detected, it means that the file was changed, which could indicate that an intruder broke into the system and installed malware such as Trojans. *See* §I above (Background and Qualifications).

100. In another example, vendors hash known malicious binaries and distribute those to customers. The hashes are small, can easily be distributed, and also do not identify the actual binary (because customers do not like exposing their own files even to their own security vendors). On the customer's computers, existing and incoming binaries can be compared: any matches found would indicate that the customer has or received malware. This malware can be cleaned (if possible), quarantined (*e.g.*, saved in a special folder), deleted, and/or reported.

101. Hashes are useful beyond security applications. Storage systems and file systems can store numerous files. Many of those files, or parts thereof, can be duplicates (*e.g.*, due to users copying files repeatedly, backup procedures, and more). Storage systems therefore have employed data de-duplication techniques to remove these duplicates: this can save a lot of space and reduce costs. Because files' names do not have to be unique and can change over time, deduplication systems often use a different identifier for the file's actual data—unique hash computed from the actual data (or parts thereof). In that way, the hash can uniquely identify the file's data regardless of the file's name or location.

2. Secret Splitting and Sharing

102. Secret splitting is a technique to break up a piece of data and distribute it such that no one person has the entire secret.

103. “Imagine that you’ve invented a new, extra gooey, extra sweet, cream filling or a burger sauce” and that “you have to keep it secret. You could tell only your most trusted employees the exact mixture of ingredients, but what if one of them defects to the competition? There goes the secret....” EX1017 (Schneier), 70.

104. “This calls for **secret splitting**. There are ways to take a message and divide it up into pieces [551]. Each piece by itself means nothing, but put them together and the message appears. If the message is the recipe and each employee has a piece, then only together can they make the sauce. If any employee resigns with his single piece of the recipe, his information is useless by itself.” EX1017 (Schneier), 70.

105. Schneier describes a simple secret-splitting scheme using XOR among several people: Alice, Bob, Carol, Dave, and Trent. EX1017 (Schneier), 70–71. Schneier then outlines problems with this simple scheme.

106. “However, this protocol has a problem: If any of the pieces gets lost and Trent isn’t around, so does the message. If Carol, who has a piece of the sauce recipe, goes to work for the competition and takes her piece with her, the rest of them are out of luck. She can’t reproduce the recipe, but neither can Alice, Bob, and Dave

working together. Her piece is as critical to the message as every other piece combined. All Alice, Bob, or Dave know is the length of the message-nothing more.” EX1017 (Schneier), 71.

107. Schneier describes an improve scheme called “Secret Sharing.” Suppose “[y]ou’re setting up a launch program for a nuclear missile. You want to make sure that no single raving lunatic can initiate a launch. You want to make sure that no two raving lunatics can initiate a launch. You want at least three out of five officers to be raving lunatics before you allow a launch.” EX1017 (Schneier), 71.

108. “This is easy to solve. [] Give each of the five officers a key and require that at least three officers stick their keys in the proper slots before you’ll allow them to blow up whomever we’re blowing up this week.” EX1017 (Schneier), 71.

109. “We can get even more complicated. Maybe the general and two colonels are authorized to launch the missile, but if the general is busy playing golf then five colonels are required to initiate a launch. Make the launch controller so that it requires five keys. Give the general three keys and the colonels one each. The general together with any two colonels can launch the missile; so can the five colonels. However, a general and one colonel cannot; neither can four colonels.” EX1017 (Schneier), 71.

110. “A more complicated sharing scheme, called a **threshold scheme**, can do all of this and more-mathematically. At its simplest level, you can take any

message (a secret recipe, launch codes, your laundry list, etc.) and divide it into n pieces, called **shadows** or shares, such that any m of them can be used to reconstruct the message. More precisely, this is called an **(m,n) -threshold scheme.** EX1017 (Schneier), 71.

111. “With a (3,4)-threshold scheme, Trent can divide his secret sauce recipe among Alice, Bob, Carol, and Dave, such that any three of them can put their shadows together and reconstruct the message. If Carol is on vacation, Alice, Bob, and Dave can do it. If Bob gets run over by a bus, Alice, Carol, and Dave can do it. However, if Bob gets run over by a bus while Carol is on vacation, Alice and Dave can’t reconstruct the message by themselves.” EX1017 (Schneier), 71.

112. “General threshold schemes are even more versatile. Any sharing scenario you can imagine can be modeled.” EX1017 (Schneier), 72.

113. The need for secret-splitting was so great that “[t]his idea was invented independently by Adi Shamir [1414] and George Blakley [182] and studied extensively by Gus Simmons [1466]. Several different algorithms are discussed in Section 23.2.” EX1017 (Schneier), 72.

D. Recovering from Data Loss

114. Storage systems are supposed to be reliable: ideally data should never be lost or corrupted. In reality, however, the risk of data loss depends on many

factors including hardware, software, configurations, workloads, and even human errors.

115. Hardware can fail. Memory bits can be corrupted by magnetic fields and even cosmic rays. Temperatures affect the stability of bits in volatile memory. Electro-magnetic storage media are susceptible to heat, humidity, vibrations (even loud music played in close proximity to a storage device), magnetism, the quality of power feeds, and more. Mechanical parts inside disks wear out over time and with use. Poor quality electronic components anywhere on a computer system can ultimately affect data reliability elsewhere.

116. Software may contain bugs. Such bugs can result in returning the wrong data to users, writing the wrong data in the wrong place, writing only some of the data, or even not writing user-requested data at all (aka “lost writes”). Bugs can reside everywhere: inside the firmware of any component, in any of the operating system components, in libraries, and applications.

117. User applications and workloads exercise the system in different ways, whether directly or indirectly. Such access can trigger bugs and affect how quickly the hardware wears out. Generally, higher levels of I/O activity tend to increase wear-out more quickly.

118. Finally, disasters can destroy data on storage hardware, from man-made disasters (*e.g.*, wars, terrorism acts) to natural disasters (*e.g.*, earthquakes, fires, or

floods). It should be noted that human errors also play a crucial role in data reliability: a significant portion of errors that result in lost data can be attributed to misconfigured systems and human errors.

119. To reduce the risk of data loss, users can use various techniques to survive one or more failures (*i.e.*, recover from disasters). The general idea is to provide extra levels of data redundancy: the more levels of redundancy that are employed, the higher the probability of recovering such data in the event of a disaster. That is partially what RAID was designed for. Indeed, RAID stands is “[a]cronym for **r**edundant **a**rray of **i**ndependent (or **i**nexpensive) **d**isks. A data storage method in which data is distributed across a group of computer disk drives that function as a single storage unit. All the information stored on each of the disks is duplicated on other disks in the array. This redundancy ensures that no information will be lost if one of the disks fails. RAID is generally used on network servers where data accessibility is critical and fault tolerance is required. There are various defined levels of RAID, each offering differing trade-offs among access speed, reliability, and cost. *See also* disk controller, error-correction coding, Hamming code, hard disk, parity bit, server (definition 1).” EX1016 (Microsoft Computer Dictionary), 437.

120. Moreover, RAID level 5 (RAID5) provides one level of redundancy by consuming one of N drives’ capacity to store parity information, distributed across

all N drives. If one drive fails, the system is said to operate in “degraded mode.” Then, RAID5 software uses the parity information to reconstruct the lost drive’s data, thus no data appears lost to the user. The cost of operating in degraded mode is that overall storage system performance is lower because RAID5 has to work harder to reconstruct lost data on the fly.

121. In RAID5 systems, it is, therefore, important to replace a failed drive promptly so that the RAID system can reconstruct the redundant data (as discussed below) and stop operating in degraded mode. If a RAID5 system that is operating in degraded mode loses *another* drive, then *all* data on the storage system is now lost—a catastrophic data loss event (a disaster). These RAID variants provide a safe window of time in which to replace failed drives. Recovering such data now requires restoring from backups, if any.

122. RAID6 offers two levels of redundancy. Here, up to two out of N drives can fail without catastrophic data loss. Other forms of RAID offer three levels of redundancy: up to 3 of N drives can fail without catastrophic data loss.

123. One can design storage systems with any level of redundancy, even up to an extreme where all data is replicated on all N drives (*i.e.*, RAID1 mirroring). RAID1 provides $N - 1$ levels of redundancy but only $1/N^{\text{th}}$ of the overall storage capacity. RAID1 can also improve read performance because data-read requests can be load-balanced across N drives.

124. Conversely, RAID0 “striping” effectively concatenates N drives into a single logical drive whose capacity is the sum of all N drives. RAID0 maximizes storage capacity and can improve performance by distributing writes across all drives. RAID0, however, provides *no* levels of redundancy. RAID0’s risk of data loss is even higher than that of a single drive because with RAID0, if *any* single drive fails, all data is lost.

125. Users can combine RAID levels to try to trade off capacity, performance, and risk of data loss—at the cost of increased complexity. For example, RAID10 is often a combination of RAID0 (mirroring) and RAID1 (striping), attempting to balance the pros and cons of both.

126. Generally, for a given set of drives, the more levels of redundancy, the less overall usable capacity users get, as redundancy information consumes some space. Also, higher levels of redundancy tend to require more computationally complex and intensive techniques, possibly increasing CPU utilization.

127. When a drive fails, it has to be replaced. Users have to purchase a suitable replacement drive or have purchased one ahead of time and placed it on reserve. Once a suitable replacement drive is available, users have to remove the failed drive from the storage system and insert the new drive. It should be noted that a fatal human error that sometimes happens at this stage is when users remove the

wrong (i.e., “good”) drive in a degraded array—potentially causing catastrophic data loss.

128. Once a replacement drive is inserted, the RAID software has to initialize the new drive with appropriate data and parity information. That process is called *resilvering*. Resilvering can take time and consume significant amount of I/O: data often has to be read from other drives, parities calculated, and then written to the replacement drive. During resilvering time, a degraded array may continue to serve I/O requests to users and applications. The more user activity there is, the more it interferes with and slows the resilvering process; this can lengthen the time to complete resilvering and hence increase the risk of catastrophic data loss.

129. Until a drive is physically replaced and fully resilvered, the storage system is still running in degraded mode and hence vulnerable to catastrophic data loss should enough additional drives fail. For that reason, users who are more concerned with the risk of data loss would employ higher levels of redundancy and also reserve several drives in the storage system as *hot spares*. Hot-spare drives are automatically brought into service when one drive fails, without requiring time-delayed, manual human intervention.

130. RAID disks are not the only way to implement different disaster recoverability levels. In its simplest form, the ability to recover data after a disaster is a probability function that increases with two factors. First, the more copies that

are kept, the higher the chance that one could recover at least one copy of the data. Second, the more copies that are distributed distant from each other (even geographically), the higher the chance that one could recover at least one: it makes no sense to keep source data and all backups in the same physical facility, because a building fire, earthquake, major flood, and even a terrorism act could destroy all of them.

131. The parities that RAID techniques use are examples of a more generalized theory of error correction. An “error-correcting code” (ECC), used in “error-correction coding,” is “[a] method for encoding that allows for detection and correction of errors that occur during transmission. Data is encoded in such a way that transmission errors may be detected and corrected by examination of the encoded data on the receiving end. Most error-correction codes are characterized by the maximum number of errors they can detect and by the maximum number of errors they can correct. Error-correction coding is used by most modems. *Also called:* error-correcting code. *See also* error detection and correction. *Compare* error-detection coding.” EX1016 (Microsoft Computer Dictionary), 196-197.

132. RAID techniques and ECC schemes can be configured to be able to recover an original data with only a subset (or threshold) number of parts available. These techniques are the basis for secret splitting as discussed in §V.C.2 above.

133. Performance or resource consumption has often been a trade-off with reliability. For example, one could make more mirror copies of a piece of data and distribute them geographically: that would increase reliability of the data and its resilience to failures; however, it would consume more storage resources and run slower as more copies have to be retrieved from different nodes on the network

134. RAID is a form of error-correcting codes. When one uses RAID with multiple storage devices, especially slower hard disks, it is possible that some drives would run much slower than others. In that case, one optimization implemented was to retrieve just a subset of the RAID data blocks, from those drives that have responded fastest—meaning to skip the drives that are the slowest to respond; and, once enough blocks are read, the full data could be reconstructed in memory before returning it to the user or application.

135. For example, Birk recognizes that “[b]y using the redundant information to avoid accessing an overloaded disk drive, the occasional transient imbalance in disk load due to the randomization is partly prevented and, when occurring, can be circumvented.” EX1018 (Birk), 13. Birk considers this problem in the context of “High Performance Video Servers”. EX1018 (Birk), 13.

136. Birk discusses techniques for “Exploitation of redundancy for load balancing”. EX1018 (Birk), 16–17. He notes that, “in 1975, Maxemchuk proposed to partition a message into m packets, compute r redundant ones such that the

original message can be reconstructed from any sufficiently large subset of the $m + r$ packets, send those packets over different paths and use the earliest arrivers to reconstruct the message [11]” and that “this substantially reduced delay and provided fault-tolerance.” EX1018 (Birk), 16–17. Moreover, “[s]imilar observations were later made by Rabin [12] and dubbed ‘information dispersal algorithm’.” EX1018 (Birk), 17.

137. Birk contemplates that an “approach, cast in terms of disk arrays, would entail submitting requests to all the disk drives holding a given parity group, and using the first ones that arrive to reconstruct the data.” EX1018 (Birk), 17.

138. Birk, however, improve on that approach to minimize the number of read requests that have to be issued in the first place. Birk “exploit[s] the redundant chunk if and only if the queue length to one of the disks holding a data chunk is longer than a certain value and the queue to the disk holding the parity chunk is sufficiently shorter. We refer to our approach as *selective exploitation of redundancy*, and apply it in conjunction with randomized layout.” EX1018 (Birk), 17–18.

139. Similarly, Chiquoine recognizes that disks in a RAID array may be slow (or degraded). “Individual disks of a RAID storage configuration will occasionally stall or respond slowly to an access request due to disk surface defects and bad block re-vectoring. ... During a slow response, the entire RAID

configuration may wait while one disk transmits requested data. Thus, a single slowly responding disk can cause a long latency for a read operation from the RAID configuration.” EX1025 (Chiquoine), [0004].

140. Chiquoine, like Birk, also considers the problem of slow storage for video servers. “For digital video and cable systems, one slowly responding disk can cause a disaster, because data needs to arrive at a video receiver at a substantially constant rate to keep the receiver’s input buffer full. ... If a slow RAID configuration causes a transmission gap so that the receiver’s input buffer empties[,] a viewer may perceive a noticeable pause in the video being viewed. Defect-free transmission of video requires that such pauses be absent.” EX1025 (Chiquoine), [0005].

141. “A reconstructor performs an XOR operation on parity and data to reconstruct the missing data corresponding to the bad or reconstructing drive. Thus, by skipping the bad drive and not waiting for a slow or degraded node or drive to respond before trying to reconstruct data improves the performance of the RAID array....” EX1025 (Chiquoine), [0010].

142. “Thus, by skipping the bad drive and not waiting for a slow drive to respond before trying to reconstruct data, will improve the performance of the RAID array towards normal performance.” EX1025 (Chiquoine), [0038].

143. The idea of reconstructing data by retrieving data from the fastest-responding servers was not limited to RAID but also use in distributed storage systems where data is protected using encryption and redundancy techniques.

144. “OceanStore is a utility infrastructure designed to span the globe and provide continuous access to persistent information. Since this infrastructure is comprised of untrusted servers, data is protected through redundancy and cryptographic techniques. To improve performance, data is allowed to be cached anywhere, anytime. Additionally, monitoring of usage patterns allows adaptation to regional outages and denial of service attacks; monitoring also enhances performance through pro-active movement of data.” EX1027 (Kubiatowicz), 1 (Abstract).

145. “To reconstruct archival copies, OceanStore sends out a request keyed off the GUID of the archival versions. Note that we can make use of excess capacity to insulate ourselves from slow servers by requesting more fragments than we absolutely need and reconstructing the data as soon as we have enough fragments. As the request propagates up the location tree (Section 4.3), fragments are discovered and sent to the requester. This search has nice locality properties since closer fragments tend to be discovered first.” EX1027 (Kubiatowicz), 8.

E. All-Or-Nothing-Transform (“AONT”)

146. A common approach for attackers seeking to access encrypted data is a brute-force search, wherein the attacker will attempt to decrypt the data using all possible keys until, for example, the resulting data is coherent or matches a known target set of data. *See* EX1010 (Rivest), 1.

147. Block ciphers, such as DES, are commonly used encryption methods that “can be vulnerable to exhaustive key-search attacks.” EX1004 (Desai), 6. *See also* EX1010 (Rivest), 2 (“The problem with most popular encryption modes is that the adversary can obtain one block of plaintext by decrypting *just one* block of ciphertext.”) (emphasis in original). This is due to (1) “their relatively small key-sizes” (EX1004 (Desai), 6), which mean there are fewer key variations that an attacker needs to test; and (2) the “separable” nature of the encrypted data blocks, which means that “an adversary can determine one block of plaintext by decrypting just one block of ciphertext” (EX1010 (Rivest), 2), without needing to test or even necessarily have access to any of the other encrypted data blocks. *See also* EX1010 (Rivest), 2 (“We illustrate this point with cipher-block chaining (CBC mode) ... any one of the s message blocks can be obtained with the decryption of just one ciphertext block. This makes the adversary’s key-search problem relatively easy, since decrypting a single ciphertext block is generally enough to test a candidate key.”).

148. The AONT technique was introduced by Dr. Ronald Rivest in a seminal paper titled “All-or-Nothing Encryption and the Package Transform,” presented at Fast Software Encryption ’97 and published in Lecture Notes in Computer Science Vol. 1267, edited by E. Biham, Springer-Verlag, 1997 (“Rivest”) (EX1010). Rivest introduces the AONT technique as “a means to make brute-force search more difficult, by appropriately pre-processing a message before encrypting it.” EX1010 (Rivest), 1. Rivest discloses that implementing AONT as a pre-processing step to encryption results in a new mode of encryption that requires that “one must decrypt the *entire* ciphertext before one can determine *even one* message block.” EX1010 (Rivest), 1. Rivest explains that as a result, “brute-force searches against all-or-nothing encryption are slowed down by a factor equal to the number of blocks in the ciphertext.” EX1010 (Rivest), 1.

149. Applying AONT before encryption is beneficial because it effectively hides data structure and patterns even before encryption, further adding obfuscation to otherwise plaintext.

150. AONT involves “mapping a message sequence $m_1, m_2, \dots, m_s$ into a pseudo-message sequence $m'_1, m'_2, \dots, m'_s$.” EX1010 (Rivest), 3. AONT has three main characteristics: (1) the transformation is reversible (“given the pseudo-message sequence, one can obtain the original message sequence”); (2) both the transformation and its inverse are “computable in polynomial time” (*i.e.*, are

“efficiently computable”); and (3) it would be “computationally infeasible to compute any function of any message block in if any one of the pseudo-message blocks is unknown.” EX1010 (Rivest), 3. *See also* EX1004 (Desai), 7 (“[A]n AONT is an efficiently computable transformation ... Given the output sequence of blocks, one can easily obtain the original sequence of blocks. Given all but one of the output sequence of blocks, it is computationally infeasible to determine any function of any input block.”).

151. The first two characteristics of AONT ensure that a legitimate user can readily reverse the AONT encryption process to obtain the original data. The third characteristic serves to make the underlying encryption brute-force-resistant by requiring that “all ciphertext blocks must be decrypted in order to determine any (property of any) message block. EX1010 (Rivest), 3. In other words, any would-be attacker to first decrypt *all* the data blocks of a data set before an attempt can be made to obtain the original data. EX1010 (Rivest), 4. *See also* EX1004 (Desai), 8 (“We give a notion, called non-separability of keys, that formalizes the inability of an adversary to gain *any* information about the underlying key, without ‘decrypting’ *every* block of the ciphertext.”) (emphases in original).

152. In other words, an attacker must decrypt the entirety of a data file—as opposed to just a one-block portion of the data file—in order to test just one single candidate key. *See, e.g.*, EX1010 (Rivest), 4 (“As an example, if I send you a[n]

eight-megabyte message encrypted in all-or-nothing CBC mode with a 40-bit DES key, the adversary must decrypt the entire eight-megabyte file in order to test a single candidate 40-bit key.”).

153. “[A]n adversary attempting a brute-force attack pays a penalty of a factor of t , where t is the number of blocks in the ciphertext.” EX1010 (Rivest), 4. Accordingly, the work required for such a brute-force “exhaustive key-search attack[] on encryption modes secure in the non-separability of keys sense” would be “slowed down by a factor equal to the number of blocks” in the encrypted data set. EX1004 (Desai), 8. *See also* EX1010 (Rivest), 4.

154. Because AONT is merely a pre-processing step, “it can be used with already-existing encryption devices and software, without changing the encryption algorithm.” EX1010 (Rivest), 5. For example, AONT can be implemented to improve the security of encryption algorithms that have specific key-length requirements that cannot be changed—such as where “the size of the key space for one’s encryption is fixed, ‘marginal,’ [] can’t be improved ... [o]r, one may be encumbered by export regulations that restrict one to a 40-bit secret key.” EX1010 (Rivest), 1–2. *See also* EX1010 (Rivest), 2 (“[I]s there any way to significantly increase the difficulty for an adversary of performing a brute-force search, while keeping the key size the same and not overly burdening the legitimate

communicants? We show that the answer to the question is to the answer to the question is yes.”).

155. There are many variations of AONT encryption. As Rivest explains, “[t]here are many approaches one might take towards devising all-or-nothing transforms.” EX1010 (Rivest), 7. For example, Rivest discloses a “package transform” AONT encryption mode, which applies AONT as a pre-processing step to “cipher-block chaining” mode of encryption. EX1010 (Rivest), 3–5. Rivest discloses that the “package transform” AONT encryption mode involves a key that is “randomly chosen rather than fixed” and further requires that “the last pseudo-message block is the exclusive-or of the key and a hash of all previous pseudo-message blocks.” EX1010 (Rivest), 5.

156. Rivest further discloses that AONT encryption can use “hash function[s] h chosen randomly from a universal family of hash functions” or be based on “an FFT-like arrangement of randomized multipermutations” or “secret-sharing schemes.” EX1010 (Rivest), 7.

157. As another example, Desai discloses a simpler “CTRT” AONT encryption mode, which applies AONT as a pre-processing step to the “counter” mode of encryption. EX1004 (Desai), 13 (“counter mode (CTR)”), 16 (“We give a construction based on the CTR mode of encryption. We describe the transform $CTRT = (E-CTRT, D-CTRT) \dots$ ”).

158. Rivest’s “package transform” AONT encryption mode computes the last pseudo-message block as an exclusive-or (XOR) of the key and the hashes of the previous pseudo-blocks. EX1010 (Rivest), 5. Conversely, Desai’s “CTRT” AONT encryption mode computes the last pseudo-message block as an exclusive-or (XOR) of the key and the previous pseudo-blocks themselves. EX1004 (Desai), 16.

159. Rivest acknowledges that its “package transform” AONT encryption mode has room for improvement in terms of efficiency, “leav[ing] it as an open problem to devise an all-or-nothing encryption mode that is substantially more efficient than the scheme presented here [in Rivest].” EX1010 (Rivest), 8. Desai’s simpler AONT encryption mode—which eliminates the step of calculating the hash of all previous pseudo-message blocks—allows for more a resource-efficient process that still maintains the brute-force-resistant properties attributable to AONT encryption. EX1004 (Desai), 16.

VI. Overview Of The ’456 Patent

160. The ’456 Patent pertains to a “secure data parser” that “pares data and then splits the data into multiple portions that are stored or communicated distinctly.” EX1001 (’456 Patent), Abstract. In connection with its claims, Figure 21—which I copied and annotated below—“shows the steps of the process

performed by the secure data parser on data to store the session master key with the parsed data.” EX1001 (’456 Patent), 55:22–25.

161. For instance, Figure 21 illustrates the following steps: (1) generating a “session master key” (highlighted in magenta); (2) encrypting the data set using the session master key (highlighted in lime green); (3) “[s]eparating the resulting encrypted data [set] into four shares or portions of parsed data” (highlighted in light blue); (4) likewise dividing the session master key into “key data” shares (highlighted in red); and (5) “appending the key data [shares] to the encrypted parsed data” (highlighted in orange and magenta) to produce final data shares. EX1001 (’456 Patent), 55:22-39. These data shares, containing both encrypted content and key information, are then further processed and stored in their designated storage units. See EX1001 (’456 Patent), 58:15-45, 58:57-64.

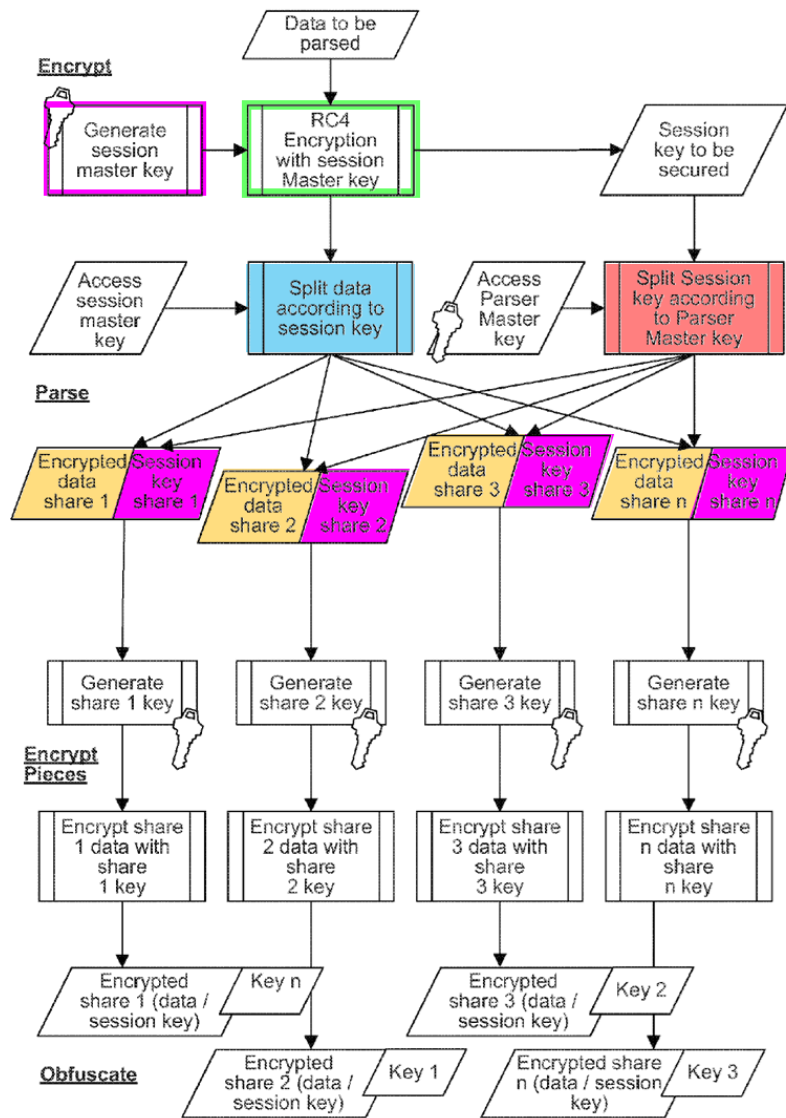


FIG. 21

EX1001 ('456 Patent), FIG. 21 (annotated).

162. The '456 Patent describes the act of dividing data into shares as a “[a] cryptographic split (cryptosplit).” EX1001 ('456 Patent), 52:40–41. According to the patent, this cryptosplit technique is designed to “provide sufficient redundancy in the shares such that only a subset of the shares are needed to reassemble or restore the data to its original or useable form.” EX1001 ('456 Patent), 53:15–19. *See also*

EX1001 ('456 Patent), 53:28–54:40. As one illustrative embodiment, the patent notes that “each unit of data is stored in two shares, the primary share and the backup share,” ensuring that even if a share is missing later, the remaining shares can still “reassemble or restore the original data with no missing data units.” EX1001 ('456 Patent), 53:28–34. *See also* EX1001 ('456 Patent), 53:34–54:40.

163. In addition, the '456 Patent notes that “an All or Nothing Transform (AoNT), such as a Full Package Transform, may be used to further secure data by transforming ... any [] suitable data elements.” EX1001 ('456 Patent), 74:45–49; *see also id.*, 74:45-75:39 (describing an example 7-step AoNT procedure).

A. Prosecution History

164. The '456 Patent stems from U.S. Application No. 14/473,813 (“’813 App.”), which was filed on August 29, 2014. That application is a continuation of U.S. Application No. 13/371,363 (“’363 App.”), filed on February 10, 2012. In turn, the '363 App. is a continuation of U.S. Application No. 11/258,839 (the “’839 App.”), filed on October 25, 2005. The '456 Patent also claims priority to two provisional applications: 60/622,146, filed October 25, 2004, and 60/718,185, filed September 16, 2005. EX1001 ('456 Patent), Cover; EX1009 ('363 File History).

165. Each claim of the '456 Patent includes the limitation “logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant.” *See* EX1001 ('456 Patent), Claims 1, 13, 25. Support for the

“logically combining” limitation appears, at the earliest, in Provisional Application No. 60/718,185, which was filed on September 16, 2005. EX1015 (’185 App.), 211. Nothing in the ’146 App. supports the “logically combining” limitation. Consistent with that understanding, I have been asked to assume that the earliest possible priority date of the ’456 Patent is September 16, 2005.

166. On November 24, 2014, Examiner Nguy issued a Non-Final Office Action in which the then-pending claims were rejected as being anticipated by U.S. Patent Publication No. 2004/0049687 (“Orsini”) and as being obvious in view of a combination of U.S. Patent Publication No. 2004/0003272 (“Bantz”) and U.S. Patent No. 6,807,649 (“Murthy”). EX1008 (’456 File History), 300–320. The Examiner also rejected the claims on the basis that the use of the term “substantially” rendered them indefinite. In addition, the claims were rejected for non-statutory double patenting in view of co-pending application numbers 13/371,363 and 13/371,364.

167. Following that rejection, the applicant conducted an interview with the Examiner and submitted a terminal disclaimer. These actions ultimately resulted in the allowance of the ’456 Patent. EX1008 (’456 File History), 373, 422, 426, 466.

VII. Level of Ordinary Skill in the Art

168. In my opinion, a person of ordinary skill in the relevant field or art (“POSITA”) as of the earliest claimed priority date of the ’456 Patent would have had a Bachelor’s degree in Computer Science, Computer Engineering, Electrical

Engineering, or an equivalent field, and about 2-3 years of experience in the fields of data storage and security. Less professional experience can be substituted by additional education, and vice versa.

169. I met and/or exceeded these requirements for a POSITA at the time of the earliest claimed priority date of the '456 Patent.

VIII. Claim Construction

170. I have reviewed the claims, specification, and the file history of the '456 Patent. EX1001 ('456 Patent), EX1008 ('456 File History). I have also reviewed the file history of the '146 App., the '185 App., and the '839 App. to which the '456 Patent claims priority. EX1014 ('146 App.), EX1015 ('185 App.), EX1024 ('839 File History).

171. I have been asked to determine if there are any particular claim terms that should be construed for the Board to understand how the prior art references read on the challenged claims. I have not identified any such claim terms.

172. Thus, I have afforded the claim terms their ordinary and customary meaning.

IX. Specific Grounds For Challenge

A. Ground I: Torre and Desai Render Obvious Claims 1–3, 10–15, 22–27, And 30.

173. Claims 1–3, 10–15, 22–27, and 30 would have been obvious in view of Torre and Desai.

174. In this Ground, I introduce illustrations to show how the Torre/Desai combination renders Claims 1–3, 10–15, 22–27, and 30 obvious.

1. Desai Color-Coding Scheme

175. The illustrations in this section have been color-coded in accordance with the color-coding scheme summarized in the table below.

Color	Disclosures in Torre and/or Desai
red	Torre's shreds
brown	Torre's Reed-Solomon encoding
orange	Torre's redundancy information
olive	Torre's storage units
lime green	Torre's input data
royal blue	Torre's pass-through data blocks
violet	Desai's encrypted data blocks $x[I] \dots x[n]$
pink	Desai's encrypted data block $x[n+1]$
teal	Desai's encrypted data blocks $x[I] \dots x[n+1]$
dark blue	Desai's encryption key

2. Overview of Torre (EX1003)

176. Torre is highly similar to the '456 Patent. Torre describes “a system and method for shredding and deshredding [data] ... to store and retrieve [the] data while emphasizing considerations such as redundancy, integrity, and security.” EX1003 (Torre), [0049]. Torre explains that the input data is transformed “into smaller pieces of data called shreds” (EX1003 (Torre), [0051]) and “writing the shredded input data ... [in] multiple storage units” (EX1003 (Torre), [0052]). In addition, “[s]hredding is done in a manner that adds redundancy to the shreds so that

the input data can be recovered from a fewer number of shreds than the total number stored.” EX1003 (Torre), [0051].

177. The system’s “shredder” processes input data through a series of transformation and processing steps, as illustrated in Figure 4 below. EX1003 (Torre), [0063] (“As mentioned, numerous passes of shredding can be implemented such as the YES branch of decision step 126 in FIGS. 3 and 4. An implementation using numerous passes of shredding is found in FIG. 5 showing a multilevel shredding approach. Input data 139 is sent to a shredder pool 140, which contains a first shredder 142 that receives the input data. The shredder 142 is coupled by an interconnect 144 to a plurality of additional shredders 146 in the shredder pool 140 that successively shred the shredded data outputted by the first shredder 142 based upon their hierarchical position in the levels of multilevel shredders. The levels of additional multilevel shredders 146 are also coupled by interconnects 144.”). *See also* EX1003 (Torre), [0055]–[0057].

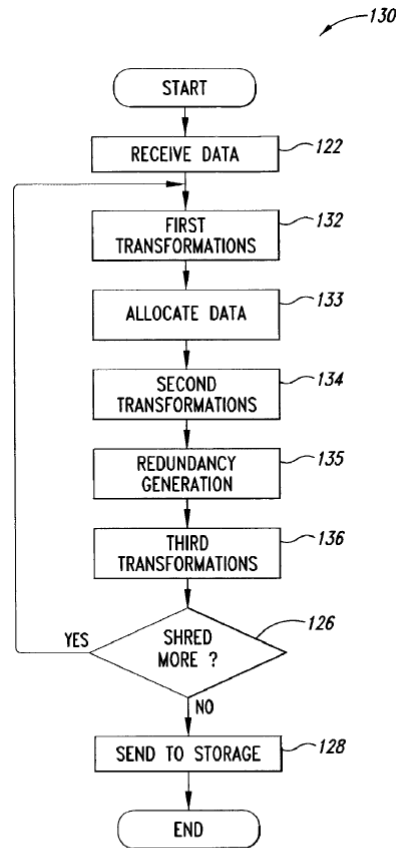


Fig. 4

EX1003 (Torre), Figure 4.

178. When input data is received (step 122), the shredder “performs first transformations (step 132),” which “could include one or more of compression, encryption, or signature generation functions.” EX1003 (Torre), [0061]. These initial transformations prepare the data for further processing.

179. After the first transformations, the system “allocates data transformed by the first transformations into various different blocks of data (step 133)” referred

to as “pre-shred[s]” (EX1003 (Torre), [0061], [0080]) by “split[ting] up the [transformed] data.” EX1003 (Torre), [0062] (“Allocation (step 133) is not necessarily found in all implementations. Those implementations that use allocation (step 133) split up the data transformed by the first transformations (step 132) to be further processed in parallel by multiple processors or sequentially by one processor with the second transformations (step 134), the redundancy generation (step 135), and the third transformations (step 136).”).

180. Torre then applies a second transformation (step 134), which “further splits the units of data into smaller sized units of data and generates additional redundant data (step 135) so that fewer pieces of stored data are necessary to restore the stored input data.” EX1003 (Torre), [0061]. Finally, the shredder performs a third transformation (step 136) to complete the shredding process. EX1003 (Torre), [0061]–[0062] (“The third transformations (step 136) can include functions to partition the shredded data based upon the number of storage units available. For instance, if the shredded data has eight shreds, but there are only seven units of storage available, either more than one of the shreds may be placed on one of the units of storage or since the shreds are redundant, one of the shreds may simply be dropped. The third transformations (step 136) can also include signature generation to further improve upon data integrity.”).

181. The shredder may consist of multiple processing components arranged in tiers. For example, the shredder architecture shown in Figure 7 below includes several distinct elements. First, a first-tier component carries out the first transformation step 132 and the allocation step 133 using a “first transformer 174” and an “allocator 176,” respectively. Second, multiple second-tier components each take in a pre-shred from the first tier and perform the second transformation step 134 and the redundancy generation step 135, using a “second transformer 178” and a “redundancy generator 180,” respectively. Finally, third-tier components perform the third transformation step 136. EX1003 (Torre), [0073]–[0074].

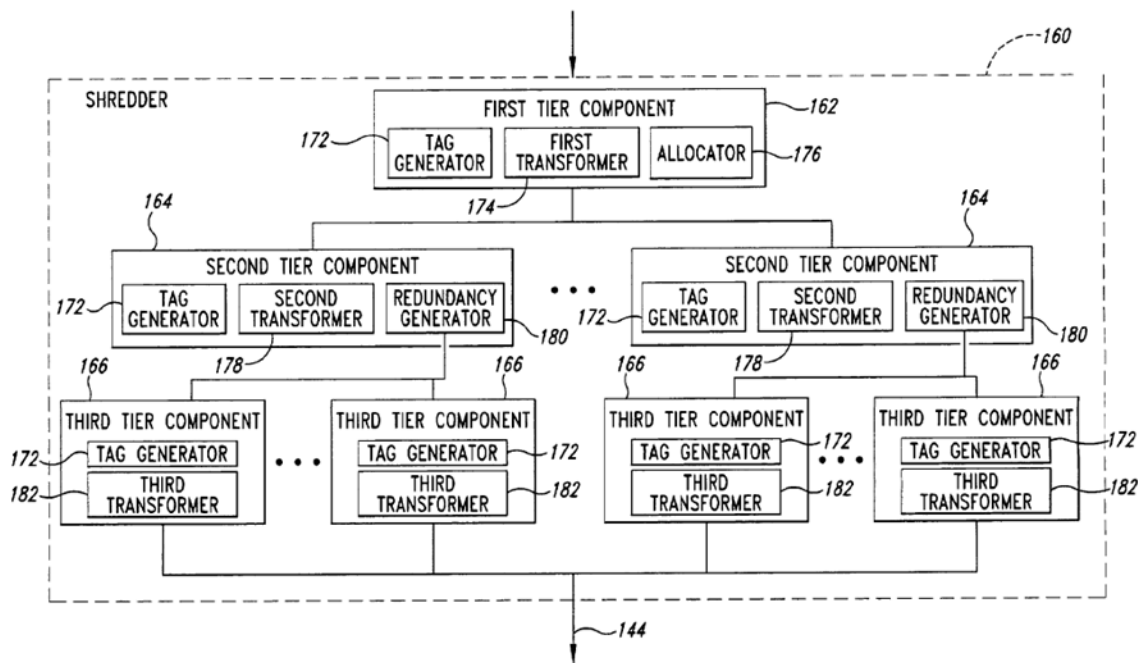


Fig. 7

EX1003 (Torre), Figure 7.

182. As input data moves through Torre's various processing steps of the shredder, the shredder attaches "tags" to the processed data "to keep track of what is been done to the pieces of the input data 139 as the pieces of the input data move through the shredder." EX1003 (Torre), [0078]–[0079]. These tags contain metadata that describes the transformations applied to each piece of data. For example, "the tags 110 can contain information about its associated piece of the input data 139 describing such details as data geometries, data relationships, data nesting, structure, coding structure, data alignment, other data descriptions including the size and length, and data qualities such as hash." EX1003 (Torre), [0078]. Torre explains that "[i]nformation [in the tags] describing how the shreds were created is used to reassemble the shreds through processes inverse to those used to create the shreds." EX1003 (Torre), [0053].

183. Torre describes the process of reconstructing shredded data as "deshredding" and depicts this process in Figure 2, shown below. EX1003 (Torre), [0010]. As explained in the patent, during deshredding, "the shredded input data 106 is sent to a deshredder 112 that performs inverse transforms 114 (A'B° C'D' for the first shred 107 and D'E'F'G'H' for the Nth shred) that are inverses of the specific transforms 104 used to create each particular shred 107 and that are identified by the tag 110 contained within the particular shred." EX1003 (Torre), [0058].

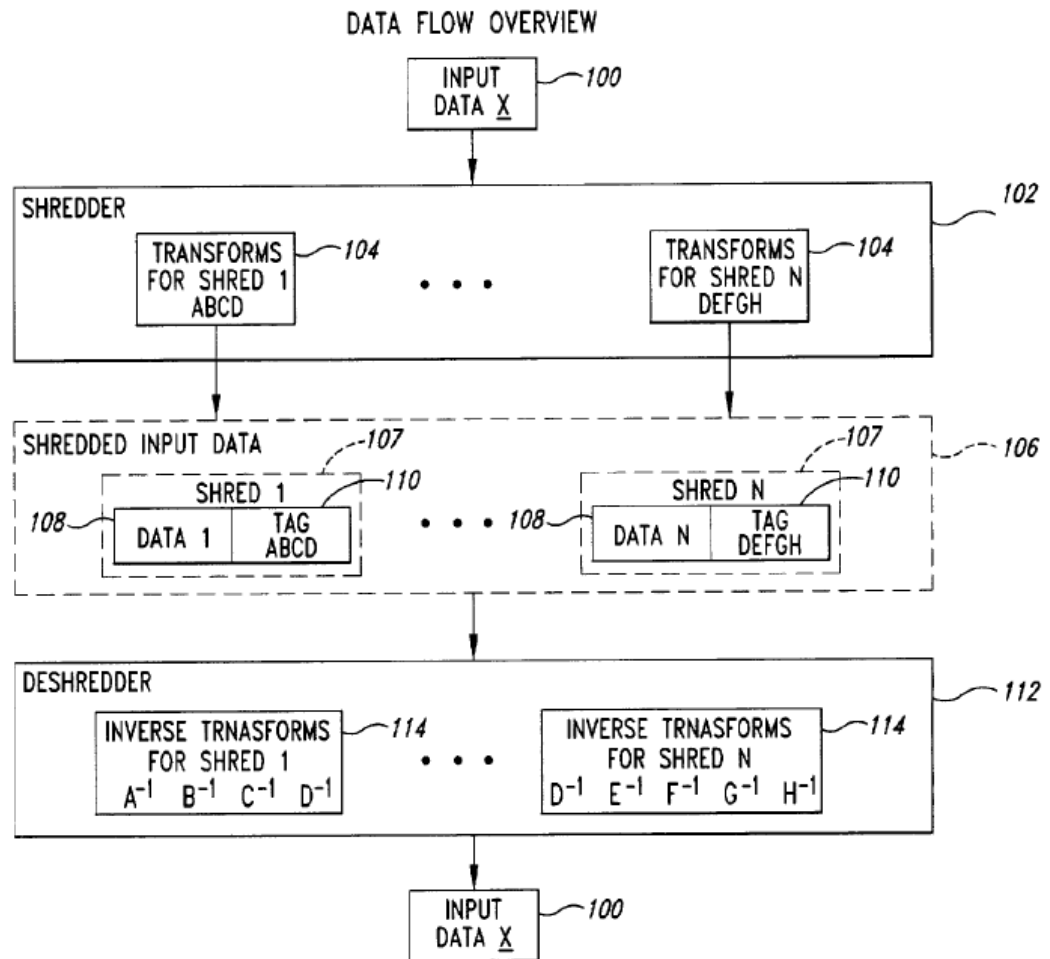


Fig. 2

EX1003 (Torre), FIG. 2.

184. Torre discloses example implementations of a shredder and a deshredder in Figures 32 and 33, respectively. Figure 32 (below) depicts a shredder system where: (1) a first transformer 464 “receives the input data 139 consisting in this example as a data file 460” and “generate[s] a signature associated with the received data to generate a pre-shred 466 containing the received data”; (2) a redundancy generator 468 creates six pre-shreds 470 by “us[ing] a 4 input-6 output Reed Solomon redundancy generation function”; and (3) a third transformer

produces “shreds 474, which includes the pre-shred data.” EX1003 (Torre), [0131]. According to Torre, “[t]he shreds 474 are then sent to the storage pool 148 where in this instance is made up of six servers 476.” EX1003 (Torre), [0132]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

SHREDDING BY FIRST AND THIRD TRANSFORMERS WITHOUT ALLOCATION

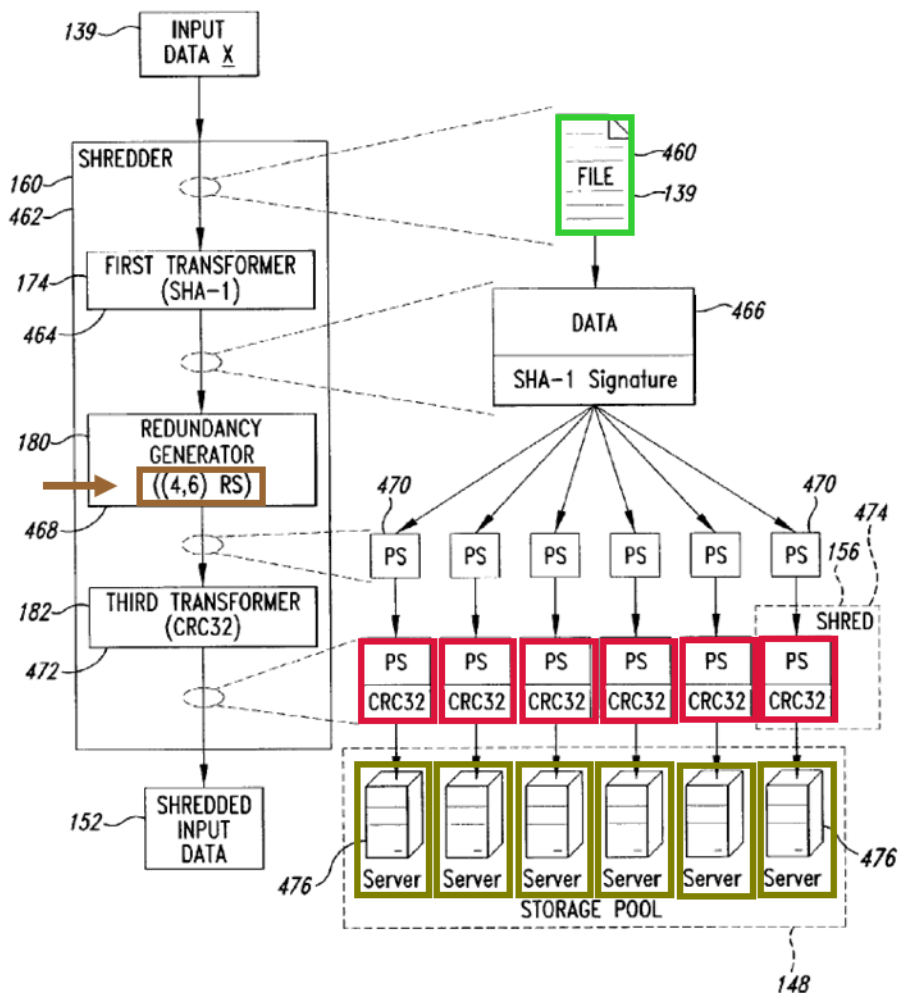


Fig. 32

EX1003 (Torre), FIG. 32 (annotated).

185. In contrast, Figure 33 (below) presents “a deshredder 477” that is described as “[c]omplementary to the shredder” shown in Figure 32. EX1003 (Torre), [0133]–[0134]. Within the deshredder 477: (1) “the third inverse transformer(s) 478” “receive the four **shreds** 474 from four of the [six] **servers** 476”; (2) the “inverse redundancy generator 480” then “perform[s] a[n] **inverse Reed Solomon redundancy function** on the four pre-shreds to reconstruct the pre-shred 466 containing data”; and (3) the “first inverse transformer” outputs the original data file 460, which corresponds to **the input data** 139. EX1003 (Torre), [0133]–[0134]. This is shown in Torre’s Figure 33 below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

DESHREDDING BY FIRST AND THIRD INVERSE TRANSFORMERS WITHOUT DEALLOCATOR

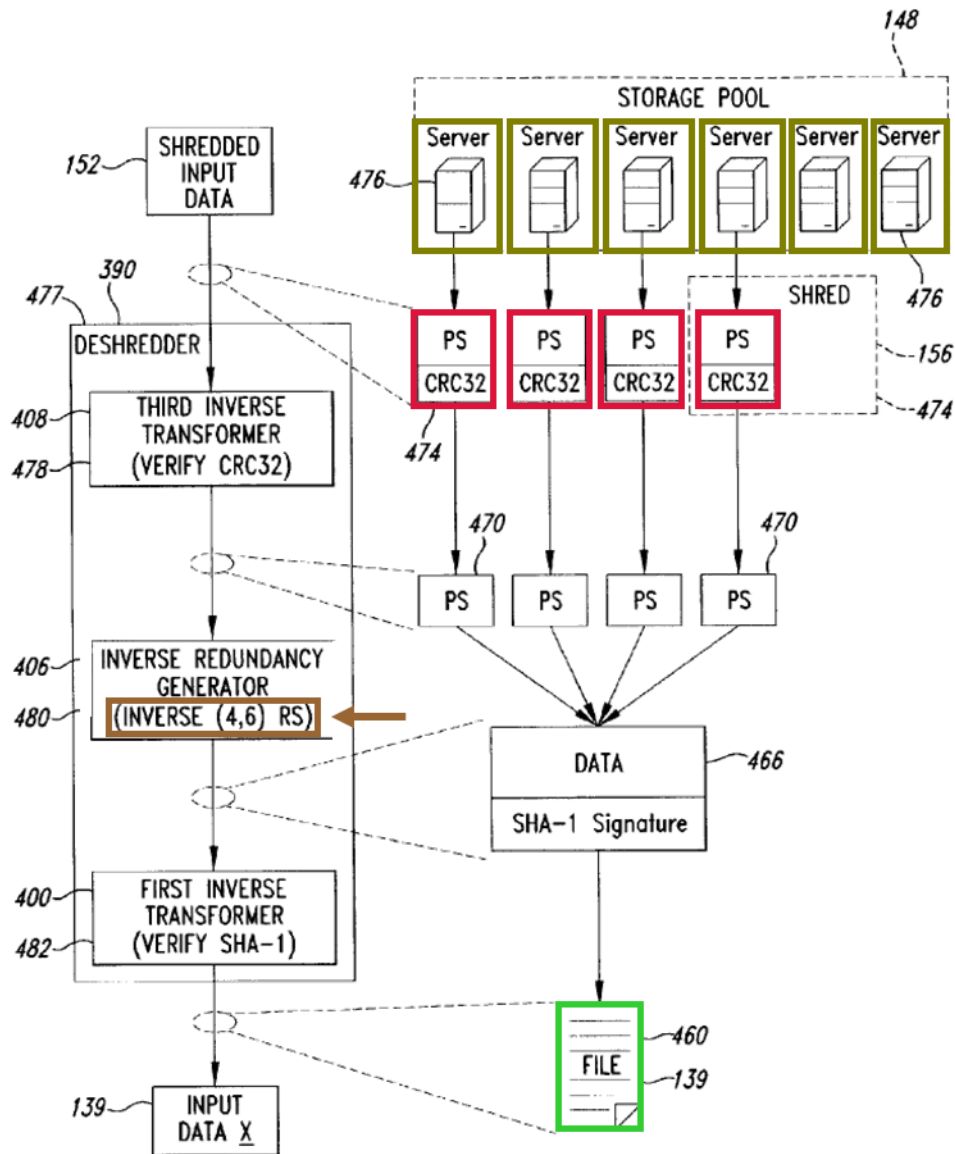
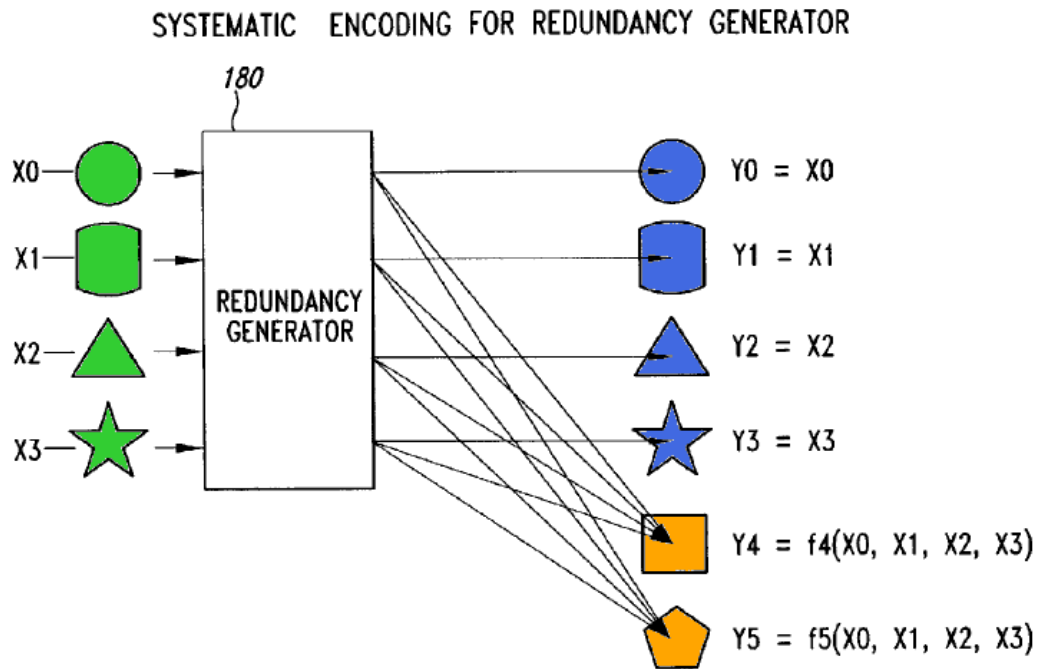


Fig. 33

EX1003 (Torre), FIG. 33 (annotated).

a. Overview of Torre Figure 21 Embodiment (“Torre System-1”)

186. Torre also clarifies that both the redundancy generator and the corresponding inverse redundancy generator—such as redundancy generator 468 in Figure 32 and inverse redundancy generator 480 in Figure 33—can support a variety of redundancy schemes. For instance, Torre describes the use of systematic encoding by the redundancy generator. EX1003 (Torre), [0116]. Figure 21 illustrates “[a]n example of systematic encoding” in which “the redundancy generator 180 receives **four inputs of data** and passes through the **four inputs of received data** to the output of the redundancy generator.” EX1003 (Torre), [0117]. Additionally, “[t]he redundancy generator 180 also generates **two outputs of redundant data** from **the received data** and outputs the **two outputs of redundant data** along with the **four outputs of the received data**.” This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.



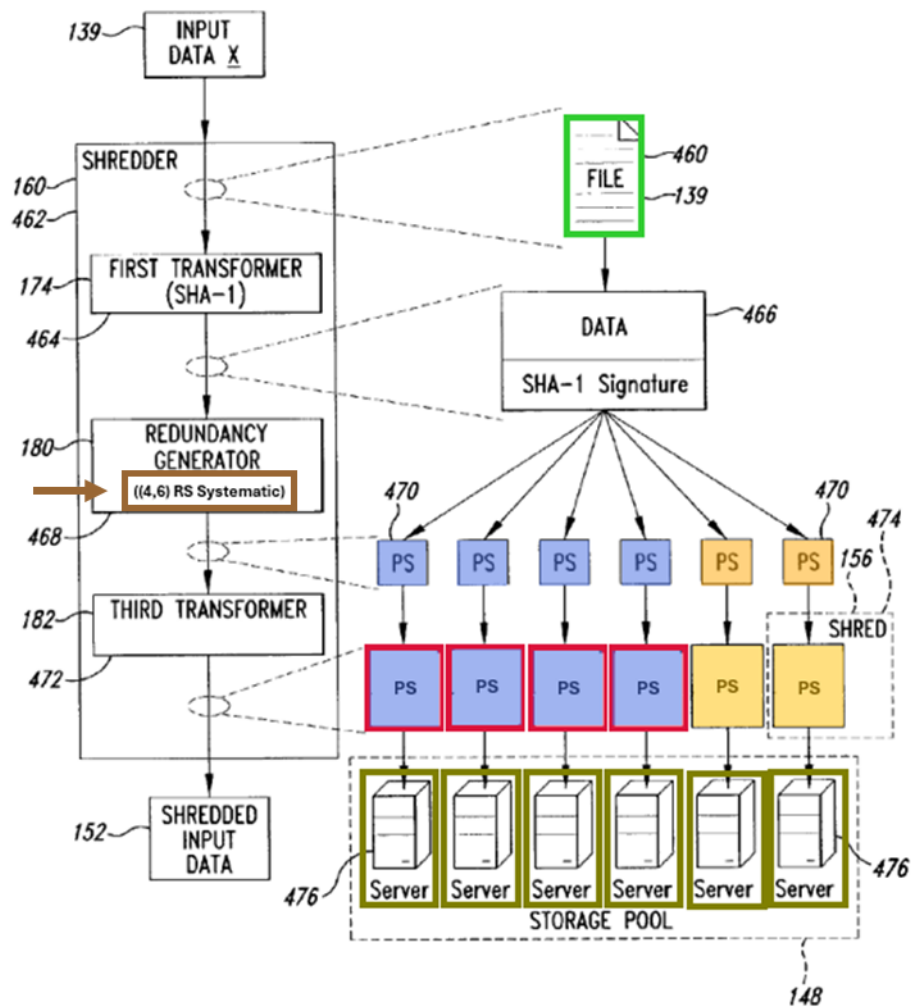
EX1003 (Torre), FIG. 21 (annotated).

187. A person of ordinary skill in the art (POSITA) would have recognized that this form of redundancy generator implementation is applicable to the embodiment shown in Torre's Figure 32, as that figure also identifies the redundancy generator using the label "180." See EX1003 (Torre), Figure 32. The figure below highlights the relevant features described by Torre in connection with the shredder of Figure 32, but it has been modified—consistent with Torre's teachings—so that the redundancy generator employs **Reed-Solomon systematic encoding**, as demonstrated in Torre's Figure 21. In this modified version, the redundancy generator produces pre-shreds 470 that include **both pass-through data blocks** from **the received data** and **redundant data**. This modified system is referred to here as

“Torre System-1.” This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre System-1

SHREDDING BY FIRST AND THIRD TRANSFORMERS WITHOUT ALLOCATION



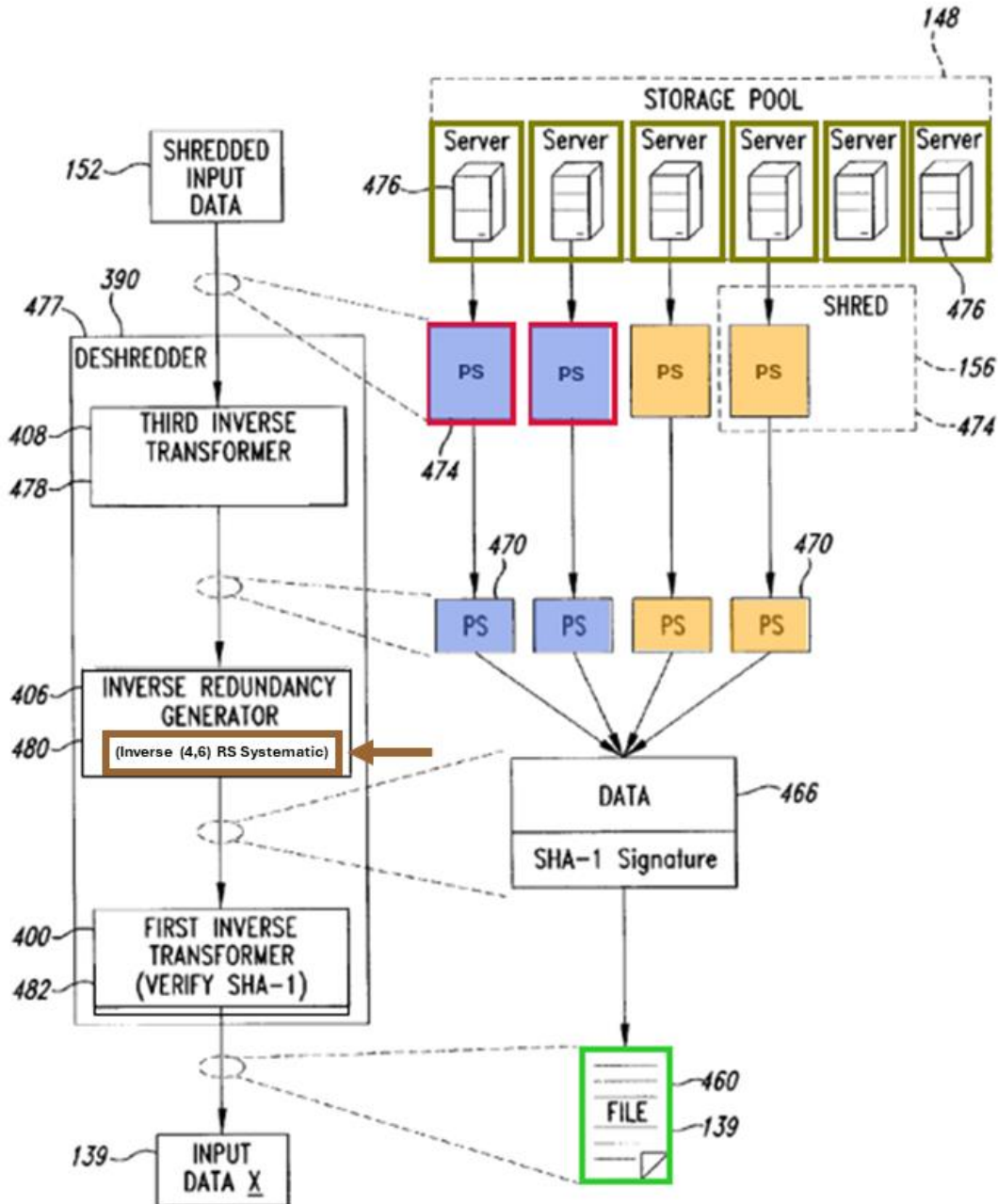
EX1003 (Torre), FIG. 32 (annotated).

188. Torre similarly explains that this type of variation may also be applied to the inverse redundancy generator shown in Figure 33. The figure below depicts

the components Torre associates with Figure 33 but has been modified—consistent with Torre’s teachings—so that the inverse redundancy generator applies **inverse Reed-Solomon systematic encoding** in order to reconstruct the pre-shred 466. This modified configuration is also referred to as “Torre System-1.” This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre System-1

DESHREDDING BY FIRST AND THIRD INVERSE TRANSFORMERS WITHOUT DEALLOCATOR



EX1003 (Torre), Fig. 33 (annotated).

b. Overview of Torre Figure 22 Embodiment (“Torre System-2”)

189. In addition to systematic encoding, Torre also states that the redundancy generator may implement non-systematic encoding. EX1003 (Torre), [0116]. Figure 22 provides “[a]n example of non-systematic encoding,” in which “the redundancy generator 180 generates six outputs of **redundancy-generated data** from four inputs of **received data**.” EX1003 (Torre), [0117]. The “generat[ed] six outputs of redundancy-generated data”—*i.e.*, redundancy generator outputs Y0–Y5—are visually represented as directional vectors in Figure 22. EX1003 (Torre), FIG. 22. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

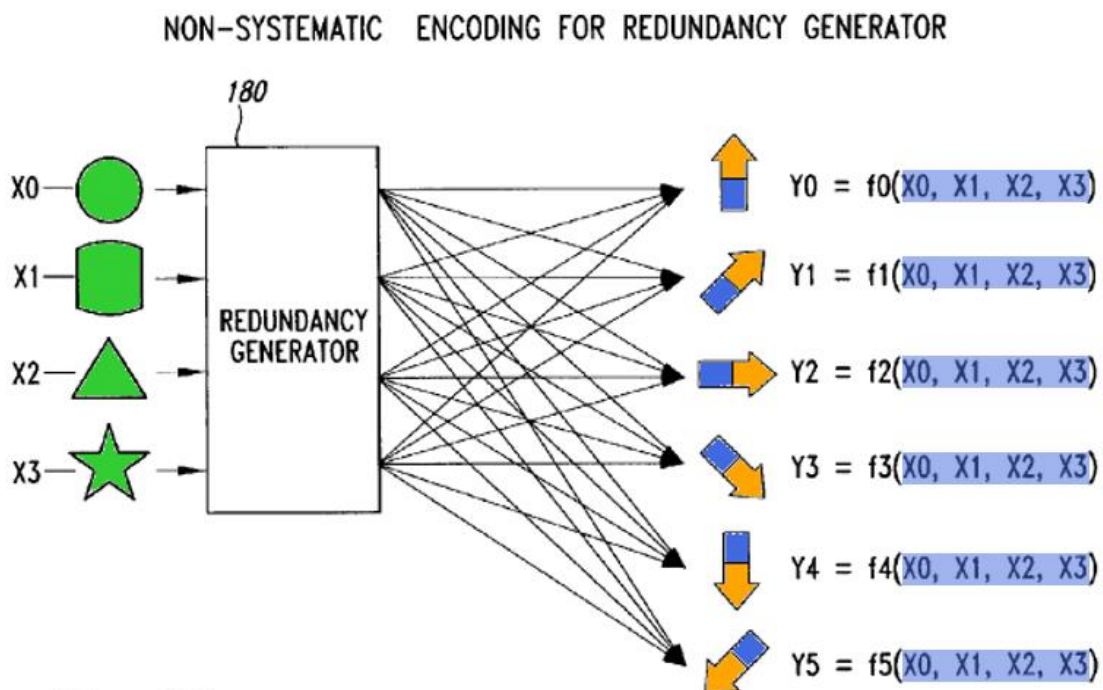


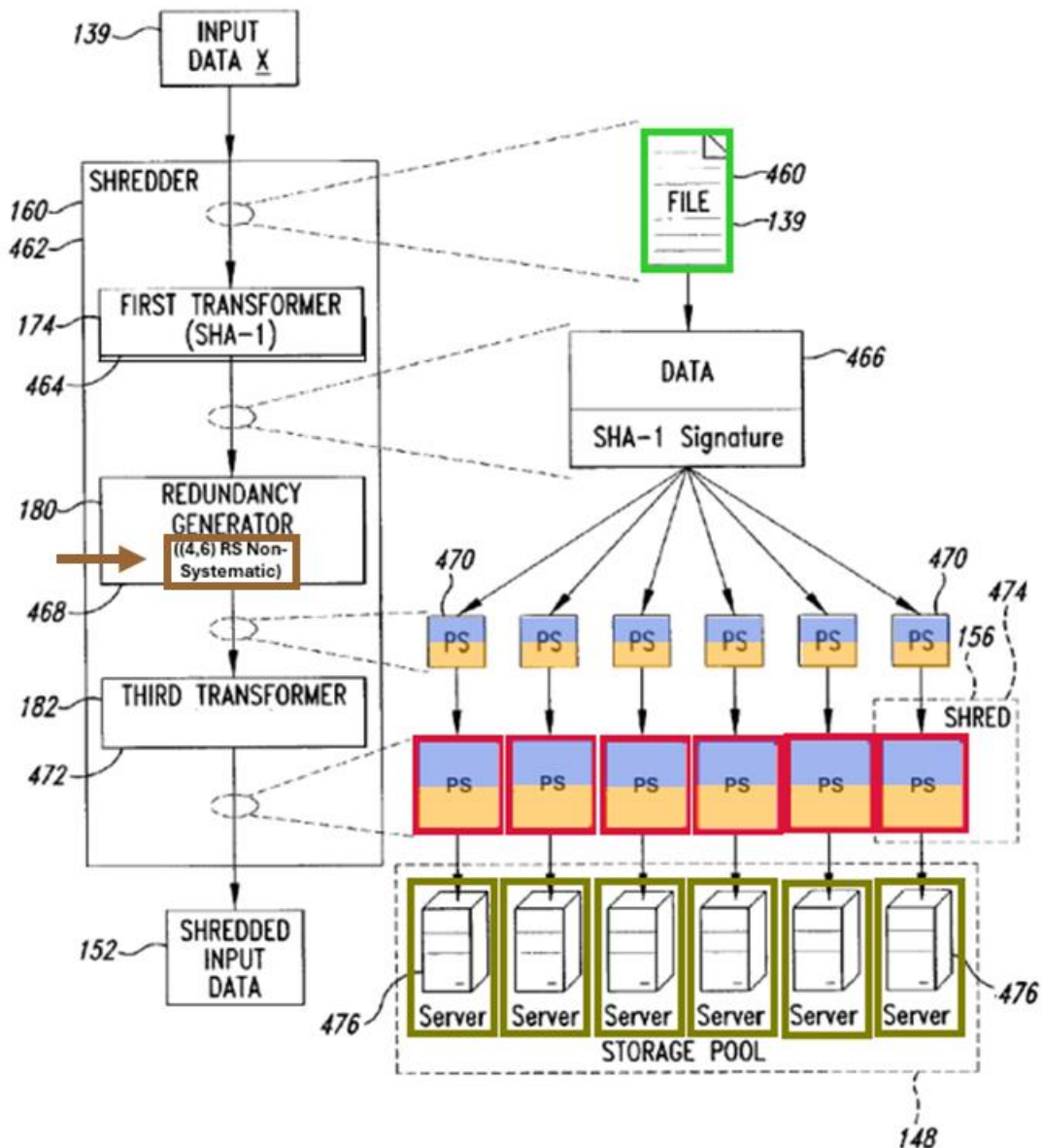
Fig. 22

EX1003 (Torre), FIG. 22 (annotated).

190. A person of ordinary skill in the art (POSITA) would have recognized that this form of redundancy generator implementation is applicable to the embodiment shown in Torre's Figure 32; that figure also identifies the redundancy generator using the label "180." *See* EX1003 (Torre), Figure 32. The figure below highlights the relevant features described by Torre in connection with the shredder of Figure 32, but it has been modified—consistent with Torre's teachings—so that the redundancy generator employs **Reed-Solomon *non-systematic* encoding**, as demonstrated in Torre's Figure 22. In this modified version, the redundancy generator shown here (as brown-annotated "(4,6) RS") applies **Reed-Solomon (RS) *non-systematic* encoding**. As a result, it generates pre-shreds 470 that include both **pass-through data blocks** from **the received data** and **redundant data**. This configuration is referred to here as "Torre System-2." This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre System-2

SHREDDING BY FIRST AND THIRD TRANSFORMERS WITHOUT ALLOCATION

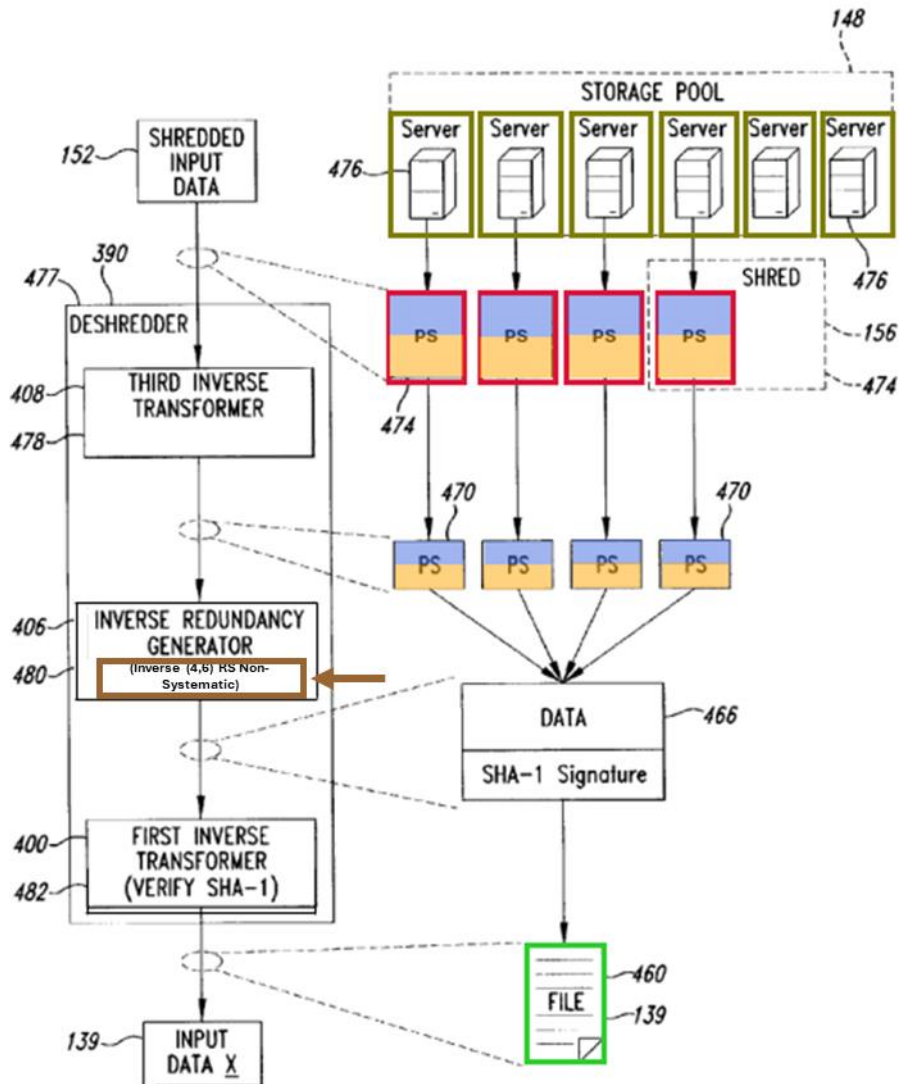


EX1003 (Torre), FIG. 32 (annotated).

191. Torre similarly explains that this type of variation may also be applied to the inverse redundancy generator shown in its Figure 33. The figure below depicts the components Torre associates with Figure 33 but has been modified—consistent with Torre’s teachings—so that the inverse redundancy generator applies **inverse Reed-Solomon *non-systematic* encoding** in order to reconstruct the pre-shred 466. This modified configuration is also referred to as “Torre System-2.” This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre System-2

DESHREDDING BY FIRST AND THIRD INVERSE TRANSFORMERS WITHOUT DEALLOCATOR



EX1003 (Torre), FIG. 33 (annotated).

192. Torre System-1 and Torre System-2 are functionally identical in all respects except for the particular implementation of the redundancy generator. Therefore, in the sections that follow, where the analysis does not depend on the

specific behavior of the redundancy generator, both systems will be referred to collectively as the “Torre System.”

3. Overview of Desai (EX1004)

193. Desai discusses a variant of a method known as the “all-or-nothing transform” (“AONT”). As explained in §V.E above, this concept was originally introduced by Dr. Rivest in his “All-or-Nothing Encryption and the Package Transform” paper. See §V.E above [All-Or-Nothing Transform (“AONT”)]; EX1010. In general, AONT involves applying a transformation to a data set—such as a sequence of messages (*e.g.*, $m_1, m_2, \dots, m_s$)—to produce a modified version of that data set (*e.g.*, pseudo-message sequence $m'_1, m'_2, \dots, m'_s$). See EX1010 (Rivest), 3. The defining feature of this transformation process is that the transformed data must be entirely available before the transformation can be reversed to recover the original message sequence. In other words, unless *all* elements of the transformed sequence (*e.g.*, $m'_1, m'_2, \dots, m'_s$) are at hand for the reverse transformation process, the original data (*e.g.*, $m_1, m_2, \dots, m_s$) cannot be retrieved. See EX1010 (Rivest), 3; EX1004 (Desai), 7.

194. AONT may be used in combination with a follow-on encryption step. For instance, after transforming the original data set using AONT, the resulting modified data set can then be encrypted using an encryption technique of the user’s choice, such as cipher block encryption. One advantage of applying AONT prior to

encryption is that it conceals underlying data structures and patterns even before the encryption is performed, thereby enhancing obfuscation beyond what would be achieved by encrypting plaintext alone.

195. Desai describes “a simplified version of Rivest’s ‘package transform’ construction of an AONT,” which is built upon “an efficient block-cipher-based construction of an AONT.” EX1004 (Desai), 9, 16. *See also* EX1004 (Desai), 7 (“As defined in [Rivest], an AONT is an efficiently computable transformation, mapping sequences of blocks to sequences of blocks”), 22. This “simplified” version of AONT allows for “all-or-nothing encryption modes that cost only two times the cost of normal CBC encryption,” which differs from Rivest’s original “package transform,” where “the resulting modes had cost about three times the cost of CBC.” EX1004 (Desai), 9.

196. Desai describes a specific implementation of AONT that is “based on the CTR [i.e., counter] mode of encryption.” This construction, which Desai refers to as “CTRT,” comprises two separate algorithms: an “encoding algorithm” known as “*E*-CTRT” and a corresponding “decoding algorithm” referred to as “*D*-CTRT.” EX1004 (Desai), 15–16:

Algorithm $\mathcal{E}$-CTRT$^{F,F^{-1}}$ ($x[1] \dots x[n]$) $K' \leftarrow \{0, 1\}^l$ $K = K' \bmod 2^k \quad // (K = k)$ for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$ $x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$ return $x'[1] \dots x'[n+1]$	Algorithm $\mathcal{D}$-CTRT$^{F,F^{-1}}$ (x') Parse x' as $x'[1] \dots x'[n+1]$ $K' = x'[1] \oplus \dots \oplus x'[n+1]$ $K = K' \bmod 2^k \quad // (K = k)$ for $i = 1, \dots, n$ do $x[i] = x'[i] \oplus F_K(i)$ return $x[1] \dots x[n]$
--	---

EX1004 (Desai), 16.

197. Turning first to the encoding algorithm, it operates on an input message x , which is provided as a sequence of data blocks $x[1]$ through $x[n]$. The algorithm produces, as output, a corresponding sequence of encrypted data blocks: $x'[1]$ through $x'[n]$, as well as an additional block $x'[n+1]$. EX1004 (Desai), 15–16. The encoding process also requires a parameter “ F ,” which refers to a cipher characterized by a key length k and block length l . EX1004 (Desai), 16–17. (Note: is math and computer theory, the tick symbol x' attached to a variable x denotes a variant of x ; x' is pronounced “X prime.”)

198. Desai describes the $\mathcal{E}$ -CTRT $^{F,F^{-1}}$ transformation algorithm in the following manner, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{E}\text{-CTRT}^{F,F^{-1}}(x[1] \dots x[n])$
Step E1	$K' \leftarrow \{0, 1\}^l$
Step E2	$K = K' \bmod 2^k \quad // (K = k)$
Step E3	for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$
Step E4	$x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$
Step E5	return $x'[1] \dots x'[n+1]$

EX1004 (Desai), 16 (annotated and encryption each step labeled E1 through E5).

199. In **step E1** of the algorithm, a key K' is generated as a random bit string having a length of l . EX1004 (Desai), 16–17.

200. In **step E2**, the algorithm calculates K' modulo 2^k to derive the truncated encryption key K , where k represents the key length supported by the cipher F . EX1004 (Desai), 16 (“We describe the transform $\text{CTRT} = (\mathbf{E}\text{-CTRT}, \mathbf{D}\text{-CTRT}) \dots$ using an ideal cipher F with key length $k \dots$ ”).

201. In **step E3**, the algorithm encrypts each of the data blocks in the sequence $x[1]$ through $x[n]$ by applying the cipher F —using the truncated key K —to each individual block. This process produces the encrypted data blocks $x'[1]$ through $x'[n]$. EX1004 (Desai), 16 (“The message x to be transformed is regarded as a sequence of l -bit blocks, $x = x[1] \dots x[n]$.”).

202. In **step E4**, the algorithm performs an **exclusive-OR operation** (“XOR,” represented by the symbol “ $\oplus$ ”) between the key K' and each of the encrypted data

blocks $x'[1]$ through $x'[n]$. This operation yields a new data block, $x'[n+1]$. EX1004 (Desai), 16.

203. In step E5, the algorithm “return[s] $x'[1]$... $x'[n]$, $x'[n+1]$.” EX1004 (Desai), 16.

204. Turning to the decoding algorithm, it takes as input the encrypted message x' , which consists of a sequence of encrypted data blocks $x'[1]$ through $x'[n]$, along with the additional block $x'[n+1]$. The algorithm then reconstructs and outputs the original sequence of data blocks $x[1]$ through $x[n]$. EX1004 (Desai), 16.

205. Desai outlines the $D\text{-CTRT}^{F,F^{-1}}$ transformation algorithm as follows, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $D\text{-CTRT}^{F,F^{-1}}(x')$
Step D1	Parse x' as $x'[1] \dots x'[n+1]$
Step D2	$K' = x'[1] \oplus \dots \oplus x'[n+1]$
Step D3	$K = K' \bmod 2^k$ // ($ K = k$)
Step D4	for $i = 1, \dots, n$ do $x[i] = x'[i] \oplus F_K(i)$
Step D5	return $x[1] \dots x[n]$

EX1004 (Desai), 16 (annotated and each decryption step labeled D1 through D5).

206. In **step D1**, the algorithm “[p]arse[s] x' as $x'[1] \dots [x'[n], x'[n+1]$.”

EX1004 (Desai), 16.

207. In **step D2**, the algorithm recovers K' by computing the XOR operation (denoted by the symbol “ $\oplus$ ”) between $x'[1]$ through $x'[n]$ and $x'[n+1]$. EX1004 (Desai), 16.

208. In **step D3**, the algorithm calculates K' modulo 2^k to derive the truncated encryption key K , where k corresponds to the key length supported by the cipher F . EX1004 (Desai), 16 (“We describe the transform CTRT = (E -CTRT, D -CTRT) ... using an ideal cipher F with key length k ...”).

209. In **step D4**, the algorithm decrypts the encrypted data blocks $x'[1]$ through $x'[n]$ by applying the cipher F —using the recovered key K —to each individual block. This step restores the original input data blocks $x[1]$ through $x[n]$. EX1004 (Desai), 16.

210. In **step D5**, the algorithm “return[s] the data set $x[1] \dots x[n]$.” EX1004 (Desai), 16.

211. Desai states that its “CTRT is strong enough that when used to realize the all-or-nothing encryption paradigm, the resulting mode will have the properties we desire,” including “key-search resistance” and “non-separability of keys.” EX1004 (Desai), 12–13, 16–17. Desai further explains that although “AONT is

distinct from an encryption scheme,” “if the output of an AONT is encrypted ... then the resulting scheme will not only be secure as an encryption scheme but also be strongly non-separable.” This means that “it should be infeasible to determine even one message block (or any property of a particular message block) without decrypting all [encrypted] blocks.” EX1004 (Desai), 7.

212. Desai further notes that when CTRT is used, “[a]s long as at least one block of the [AONT] output is missing, the key K used to ‘encrypt’ the message blocks is information theoretically hidden,” and additionally states that “CTRT would also have been secure in the sense given by Rivest [20].” EX1004 (Desai), 11–12, 17.

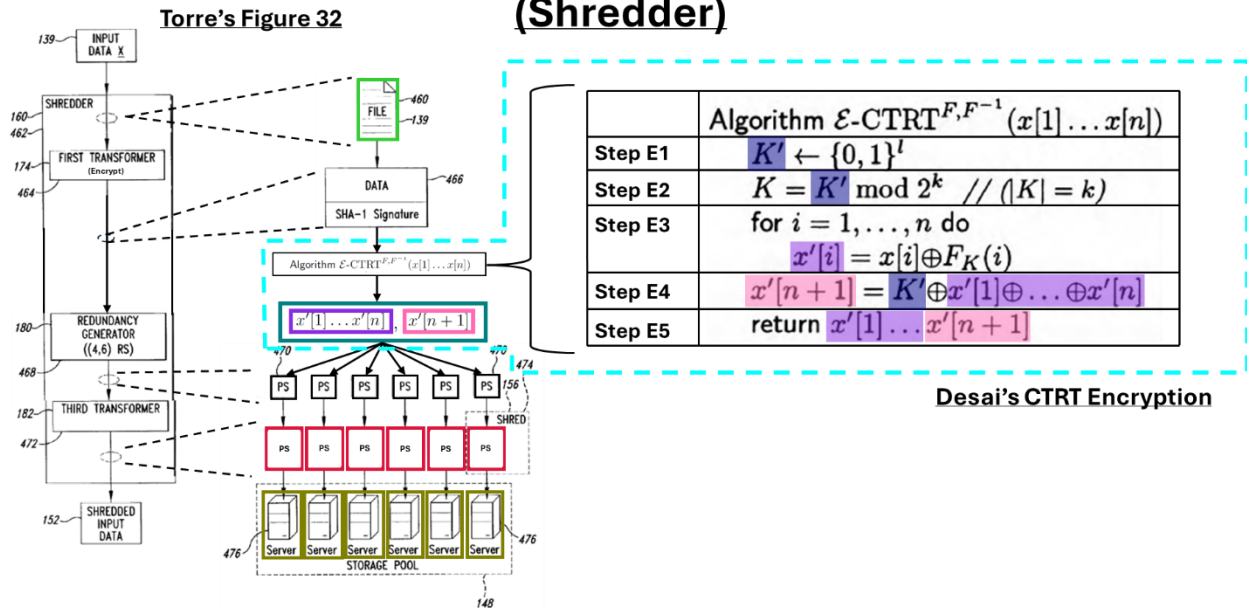
4. Combining Torre With Desai (“Torre/Desai System”)

213. Torre is concerned with the secure storage of input data by parsing and dividing that data into multiple shreds. *See* §IX.A.2 above [Torre]. A person of ordinary skill in the art would have found it obvious to enhance Torre’s shredder—whether in Torre System-1 or Torre System-2—by incorporating Desai’s CTRT technique in order to further strengthen the security of the stored data.

214. In particular, it would have been obvious to implement Desai’s E -CTRT^{F, F⁻¹} transformation algorithm within Torre’s shredder—specifically placing it between Torre’s first transformer and the redundancy generator. In this configuration, Desai’s CTRT algorithm would transform the data 466 (which

corresponds to the data blocks $x[1]$ through $x[n]$) into a **data set** consisting of the **encrypted data blocks** $x'[1]$ through $x'[n]$, along with the **additional data block** $x'[n+1]$. The figure below illustrates the shredder portion of the combined Torre/Desai System, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

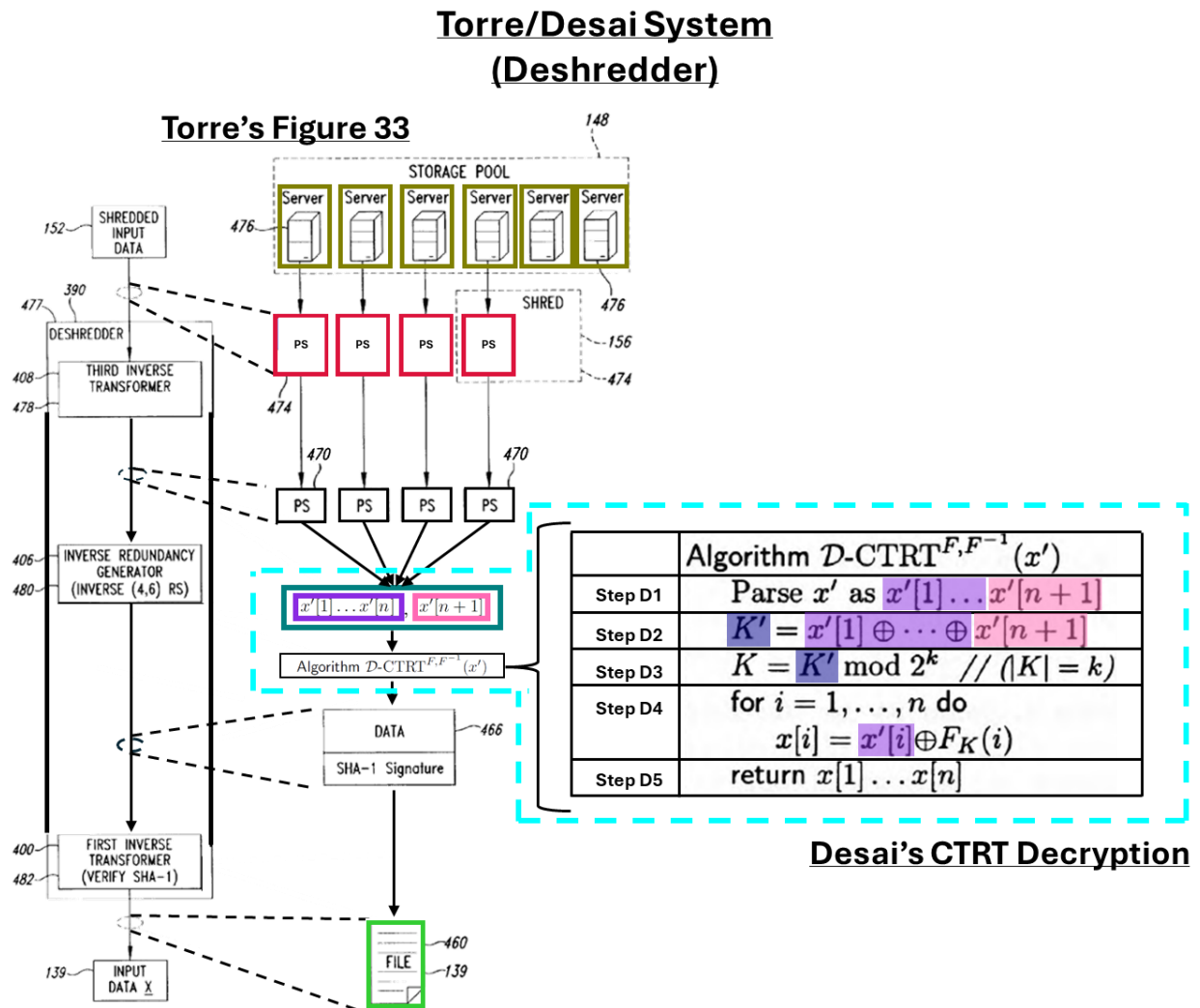
Torre/Desai System (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

215. Similarly, it would have been obvious to modify Torre's deshredder in a corresponding manner. Specifically, a person of ordinary skill in the art would have recognized that Desai's $D\text{-CTRT}^{F,F^{-1}}$ transformation algorithm could be implemented within Torre's deshredder—positioned between Torre's first inverse transformer and the inverse redundancy generator. In this setup, the algorithm would

operate on the encrypted data blocks $x'[1]$ through $x'[n]$ and the additional data block $x'[n+1]$ to reconstruct the original data 466, represented by the data blocks $x[1]$ through $x[n]$. The figure below illustrates the deshredder component of the combined Torre/Desai System, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.



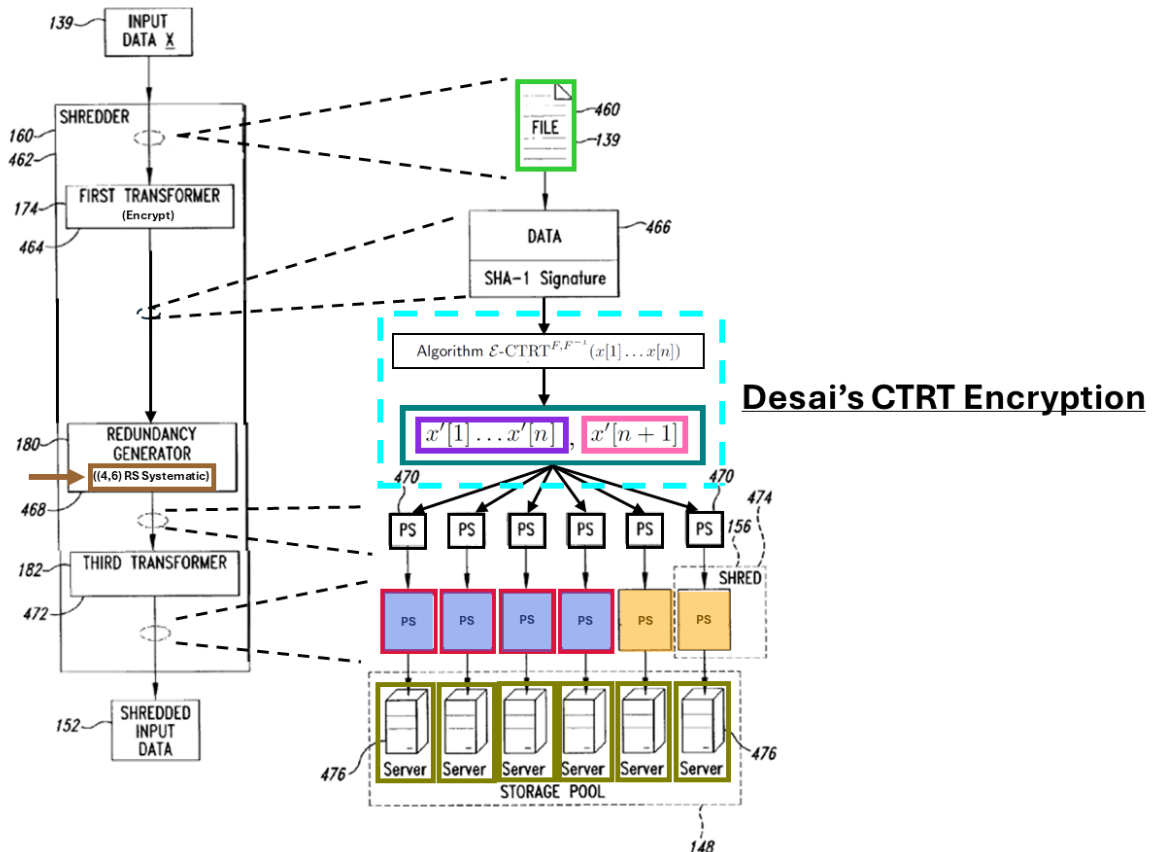
EX1003 (Torre), FIG. 33 (annotated); EX1004 (Desai), 16.

a. Overview of Torre/Desai System-1

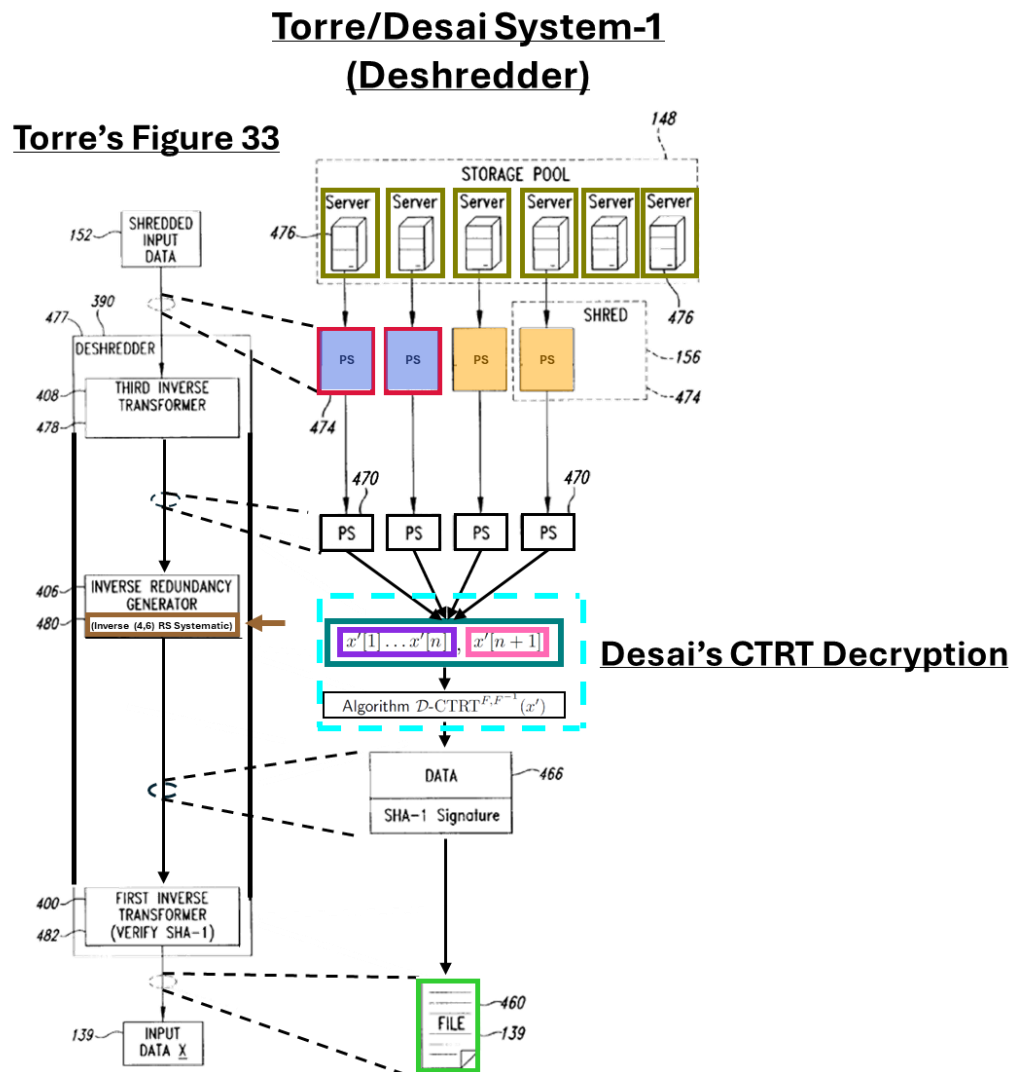
216. For the same reasons discussed in §IX.A.4 above, it would have been obvious to a person of ordinary skill in the art to incorporate Desai's CTRT technique into Torre System-1 in order to enhance the security of the stored data. This modified configuration is referred to herein as "Torre/Desai System-1." This is shown in the figures below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-1
(Shredder)

Torre's Figure 32



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.



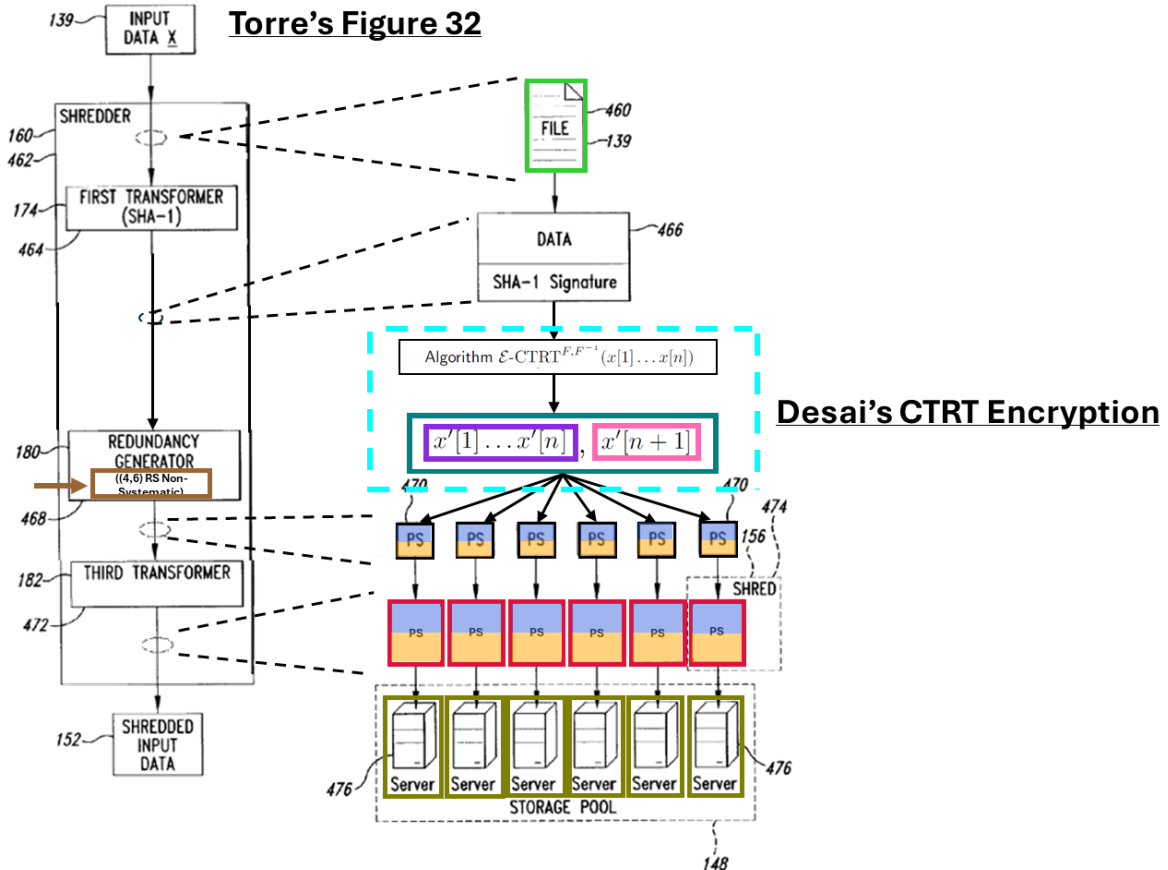
EX1003 (Torre), FIG. 33 (annotated); EX1004 (Desai), 16.

b. Overview of Torre/Desai System-2

217. For the same reasons, a person of ordinary skill in the art would have found it obvious to enhance Torre System-2 by integrating Desai's CTRT technique, thereby improving the security of the stored data. This configuration is referred to

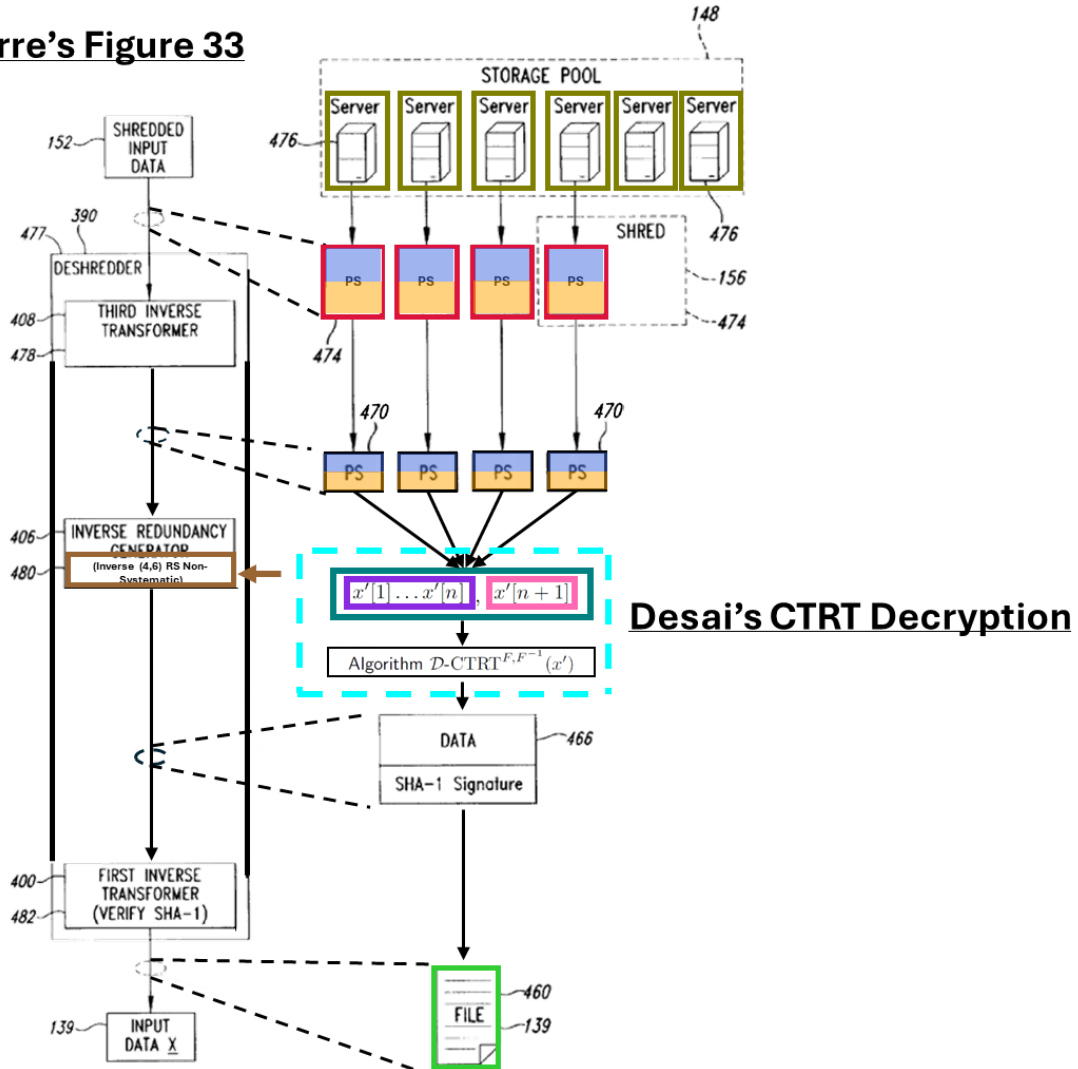
herein as “Torre/Desai System-2.” This is shown in the figures below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-2 **(Shredder)**



EX1003 (Torre), FIG. 32; EX1004 (Desai), 16.

Torre's Figure 33



EX1003 (Torre), FIG. 33; EX1004 (Desai), 16.

c. Reasons That A POSITA Would Have Been Motivated To Combine Torre and Desai To Arrive At Torre/Desai System-1 and Torre/Desai System-2

218. A person of ordinary skill in the art (POSITA) would have been motivated to combine Torre with Desai as implemented in the Torre/Desai System for at least the following five reasons.

219. *First*, both Torre and Desai are directed toward addressing substantially the same problem using conceptually similar techniques. Specifically, each is concerned with securely encrypting a data file as data blocks in such a way that the original file can later be reconstructed if a minimum number of blocks are available for decryption.

220. Torre, for example, discloses “[a] system and method for shredding and deshredding [data] ... to store and retrieve [the] data while emphasizing considerations such as redundancy, integrity, and security.” EX1003 (Torre), [0049]. Torre further explains that “[s]hredding is done in a manner that adds redundancy to the shreds so that the input data can be recovered from a fewer number of shreds than the total number stored.” EX1003 (Torre), [0051]. Torre also states that “[w]hen the shredded input data that has been stored is later desired, it is retrieved from storage (step 88)” and “deshredded to reassemble the original input data ...” EX1003 (Torre), [0053].

221. Similarly, Desai explains that “the preferred approach might be to squeeze a little more security out of the existing encryption modes using some efficient pre-processing techniques.” EX1004 (Desai), 6. Desai further states that “it should be infeasible to determine even one message block ... without decrypting all the [encrypted] blocks Given the output sequence of [encrypted] blocks, one can easily obtain the original sequence of blocks.” EX1004 (Desai), 7. Desai also

says that that the encryption “Algorithm $E\text{-CTRT}^{F,F^{-1}}(x[1] \dots x[n])$ ” will “return $x'[1] \dots x'[n+1]$ ” and the decryption “Algorithm $D\text{-CTRT}^{F,F^{-1}}(x')$ ” will “parse x' as $x'[1] \dots x'[n+1]$ ” and “return $x[1] \dots x[n]$.” EX1004 (Desai), 16. *See also* §§IX.A.2 above and IX.A.3 above.

222. ***Second***, a person of ordinary skill in the art would have been motivated to incorporate Desai’s CTRT algorithms into Torre’s system because Torre itself expressly encourages such enhancements. Torre states that its “shredding and the shredding system and method discussed herein is envisioned to be used with ***any other encryption method that provides a desired level of security*** dependent upon such other factors as proper key management.” EX1003 (Torre), [0085]. Torre also highlights that “[s]ecurity is also becoming more necessary for conventional systems to prevent unauthorized disclosure of stored or transmitted data.” EX1003 (Torre), [0005]. Furthermore, Torre notes that “[c]ertain of the transforms 104 may be selected to address performance, ***security***, and/or space efficiency issues.” EX1003 (Torre), [0055].

223. Based on these disclosures, a POSITA seeking to enhance security—particularly to prevent unauthorized disclosure—would have been motivated to adopt a technique for encrypting data at the block level. A POSITA would have recognized that Desai teaches such a technique by using an AONT pre-processing step, which offers improved security.

224. **Third**, a person of ordinary skill in the art would have been further motivated to incorporate Desai's CTRT algorithms into Torre's system based on Desai's own teachings. Desai identifies a known vulnerability of block cipher encryption schemes—namely, their susceptibility to brute-force key search attacks. EX1004 (Desai), 6–7. To address this issue, Desai proposes the CTRT technique as a “pre-processing step” that can be used alongside existing data security methods to mitigate such vulnerabilities. EX1004 (Desai), 7; *see also* EX1004 (Desai), 6 (“[T]he preferred approach might be to squeeze a little more security out of the existing encryption modes using some efficient pre-processing techniques.”). Accordingly, a POSITA would have understood that applying Desai's CTRT method to Torre's system would enhance the overall security of the shreds stored in storage devices.

225. **Fourth**, a person of ordinary skill in the art would have been motivated to apply Desai's CTRT technique within Torre's shredder in order to add an additional layer of security. Even if an unauthorized third party were able to obtain a sufficient number of shreds (*e.g.*, “shreds 474” shown in Torre's Figure 32 and Figure 33 (EX1003 (Torre), [0132]–[0134])) to reconstruct the pre-shred (*e.g.*, “pre-shred 466” shown in Torre's Figure 32 and Figure 33 (EX1003 (Torre), [0131], [0134])), the data would remain obfuscated due to the application of CTRT. Without knowledge of how CTRT was implemented—such as the specific cipher *F*, key

length k , and block length l —a POSITA would be unable to recover any individual original data block $x[i]$, due to the non-separable nature of AONT techniques like Desai’s CTRT. This integration would therefore enhance the overall security of the system.

226. *Fifth*, a person of ordinary skill in the art would have been motivated to adopt Desai’s CTRT technique over other forms of all-or-nothing transforms, such as Rivest’s original AONT, because Desai specifically teaches that CTRT achieves comparable security while offering improved computational efficiency. For example, Desai explains that “we can now get all-or-nothing encryption modes that cost only two times the cost of normal CBC encryption, while with the package transform, the resulting modes had cost about three times the cost of CBC.” EX1004 (Desai), 9. Desai further notes that “[i]n the case of CTRT being used as the AONT, the cost of the resulting all-or-nothing codebook mode would be a factor of only two greater than CBC. From our results, we get that the resulting all-or-nothing codebook mode would be secure in the non-separability of keys sense, as well as being secure against chosen-plaintext attack.” EX1004 (Desai), 21.

227. Furthermore, Desai’s CTRT technique would have been particularly well-suited for integration into Torre’s system. Other approaches to improving resistance against key-search attacks—such as increasing the encryption key length to increase the number of potential key permutations that an attacker would be

required to test—often “necessitate changing the existing cryptographic hardware and software implementing these encryption modes.” EX1004 (Desai), 6. *See also* EX1004 (Desai), 6 (“The goal is to increase the difficulty of an exhaustive key search on symmetric encryption schemes, while keeping the key size the same ...”). In contrast, Desai’s CTRT algorithm achieves this objective by “increas[ing] the difficulty of an exhaustive key search on symmetric encryption schemes, while keeping the key size the same and not overly burdening the legitimate users.” EX1004 (Desai), 6–7.

228. Accordingly, a person of ordinary skill in the art would have been motivated to apply a known technique—namely, Desai’s CTRT algorithms—to enhance the security of Torre’s known system. This would involve using Desai’s CTRT algorithm to transform the input data, thereby further obfuscating the data and achieving non-separability. Such a combination would be expected to yield the predictable and desirable result of improved data security.

229. A person of ordinary skill in the art would have had a reasonable expectation of success in integrating Desai’s CTRT algorithms into Torre’s system for at least the following two reasons.

230. **First**, Desai teaches that “an all-or-nothing transform (AONT),” such as its CTRT algorithm, functions as a “pre-processing step” that enhances existing data security systems. EX1004 (Desai), 7 (“The all-or-nothing encryption paradigm

was suggested as a means to achieve strongly non-separable encryption modes. It involves using an all-or-nothing transform (AONT) as a pre-processing step to an ordinary encryption mode.”). Based on this disclosure, a person of ordinary skill in the art would have understood that incorporating Desai’s CTRT algorithm into Torre’s system would have involved applying a known pre-processing transformation to the input data during the shredding process and performing the corresponding reverse operations during deshredding. Doing so would yield the predictable benefit of non-separable data handling and additional input data obfuscation, thereby improving overall system security.

231. **Second**, Torre explicitly states that “the embodiments disclosed herein ... can be equivalently implemented ... as one or more *computer programs* running on one or more computers (*e.g.*, as one or more programs running on one or more data processing systems).” EX1003 (Torre), [0141]. Therefore, incorporating Desai’s CTRT technique into the Torre System would have simply required implementing software code—a task that would have been well within the capabilities of a person of ordinary skill in the art. For example, Torre notes that “designing the circuitry and/or writing the code for the software and or firmware [described by the embodiments] would be well within the skill of one of ordinary skill in the art in light of this disclosure.” EX1003 (Torre), [0141].

232. At a minimum, it would have been obvious to try implementing Desai's CTRT algorithm within Torre's system. Desai explains that block cipher encryption schemes are susceptible to brute-force key search attacks, and that this vulnerability can be addressed by using an AONT as a pre-processing step to encryption. EX1004 (Desai), 6-7. Importantly, Desai also teaches that an AONT can be applied in this manner to any known block cipher encryption without requiring any modifications to the underlying encryption algorithm. EX1004 (Desai), 6-7. Given these teachings, a person of ordinary skill in the art would have found it obvious to incorporate an AONT as a pre-processing step before the block cipher encryption operations performed by Torre's shredder in order to enhance the security of Torre's system.

233. The rationale for combining the Torre System and Desai applies equally to the combinations of Torre System-1 with Desai and Torre System-2 with Desai. This is because the integration of Desai's CTRT algorithm does not depend on or interfere with the specific implementation or operation of the redundancy generator in either Torre System-1 or Torre System-2.

234. Torre/Desai System-1 and Torre/Desai System-2 are identical in all respects except for the particular implementation of the redundancy generator. Therefore, in the sections that follow, where the analysis does not depend on the specific functionality of the redundancy generator, both configurations—

Torre/Desai System-1 and Torre/Desai System-2—will be referred to collectively as the “Torre/Desai System.”

5. Claim 1

235. Claim 1 would have been obvious in view of the Torre/Desai System.

- a. **[1Pre] A method performed by a computer system for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the method comprising:**

236. To the extent the preamble is limiting, Torre/Desai System teaches preamble [1Pre].

237. **First**, the Torre/Desai System teaches “[a] method ... for securing *a first data set*.” Torre describes “[a] system and method for shredding and deshredding ... to *store and retrieve data* while emphasizing considerations such as redundancy, integrity, and *security of data*.” EX1003 (Torre), [0049]. Additionally, Torre notes that its “shredding system and method ... is envisioned to be used with any other encryption method that provides a *desired level of security* dependent upon such other factors as proper key management.” EX1003 (Torre), [0085]. *See also* EX1003 (Torre), [0095]–[0096], [0109].

238. Torre further explains that “during allocation, certain particular bytes or other groupings of data may be rearranged based upon particular processing to occur in subsequent steps or for certain portions of the data to be secretly mapped to

certain storage units for reasons such as *enhanced security*.” EX1003 (Torre), [0109]. In the context of Figure 32, Torre further describes “receiv[ing] *the input data 139*” and processing that input data using various “transformers” for secure storage. EX1003 (Torre), [0131]–[0132].

239. *The input data* handled by Torre—and therefore by the Torre/Desai System—corresponds to the claimed “*data set*.” Torre states that “[t]he *input data* can be a *file or another form*” and may be produced “by operating software programs and by receiving or recording data such as live audio or *video*, or audio or *video* broadcast streams.” EX1003 (Torre), [0050]. This disclosure encompasses input data such as video content composed of a series of video frames. EX1020 (Rangan), 2 (“Digitization of motion video yields a sequence of frames, and that of audio yields a sequence of samples Frame is the basic unit of video. Sample is the basic unit of audio.”). A sequence of video frames constitutes a “*data set*” because it is a group of related data elements (*i.e.*, the frames). Indeed, a POSITA would have known that a “*data set*” is “[a] collection of related information made up of separate elements that can be treated as a unit in data handling.” EX1016 (Microsoft Computer Dictionary), 144 (definition 1).

240. Notably, in its litigation with Google, Patent Owner interpreted the term “*data set*” broadly to mean “collection of information for storage.” EX1013 (PO’s

Google Litigation Construction), 8. Under that construction, a set of video frames qualifies as a “data set.”

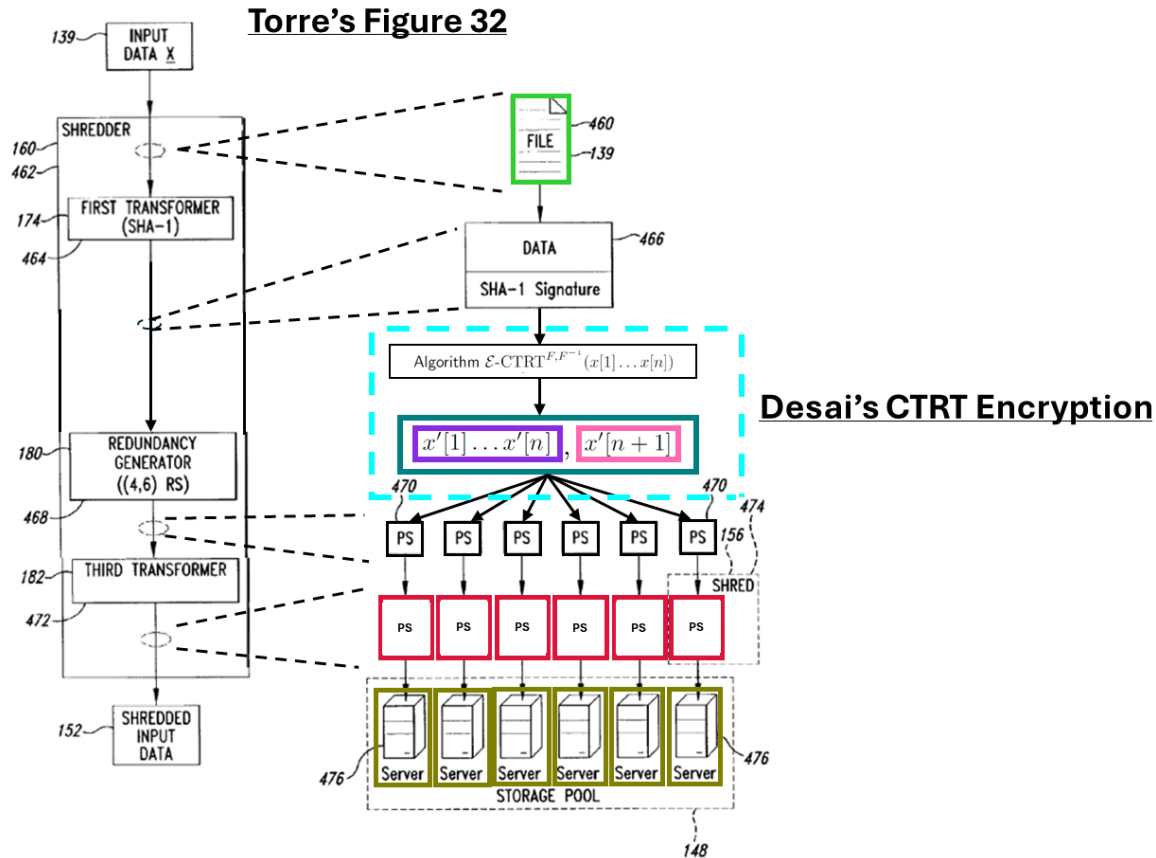
241. Furthermore, a set of video frames qualifies as a claimed “data set” under Patent Owner’s own definition of the term. For example, Patent Owner was involved in a prior litigation concerning U.S. Patent No. 11,178,116 (the “’116 Patent”). The ’116 Patent belongs to the same patent family as the ’456 Patent and largely shares the same specification. The ’116 Patent includes additional Figures 36 through 60 and corresponding written description that are not found in the ’456 Patent. *See, e.g.*, ’116 Patent, 77:51–100:11. However, it omits certain examples related to securing and splitting data that are present in the ’456 Patent. *See, e.g.*, EX1001 (’456 Patent), 75:7–39. In that prior litigation, Patent Owner adopted a broad construction of “data set” as meaning a “collection of information for storage.” EX1013 (PO’s Google Litigation Construction), 8. The “input data” in the Torre/Desai System meets that definition. As previously discussed in this section, Torre teaches that the input can be a video file (*i.e.*, a file composed of multiple video frames). *See* EX1003 (Torre), [0050]; EX1016 (Microsoft Computer Dictionary), 144 (definition 1). As each frame constitutes “information for storage,” the full video file represents a “collection of information for storage” as claimed.

242. **Second**, the Torre/Desai System also teaches that the method “provid[es] recoverability of the first data set in the event of unavailability of data

at one or more storage locations.” Torre explains that “[i]n the process of shredding, input data is generally transformed into many smaller pieces of data called **shreds**.” EX1003 (Torre), [0051]. It further states that “[s]hredding is done in a manner that adds redundancy to the **shreds** so that the **input data** can be recovered from *a fewer number of shreds* than the total number stored.” EX1003 (Torre), [0051].

243. For example, as illustrated in Figure 32, Torre teaches adding redundancy information to the **shreds** before they are stored in the **servers**—such as the six **servers 476** shown in annotated Figure 32. Specifically, “the redundancy generator 468 receives the data file 460 and uses a 4 input-6 output **Reed Solomon redundancy** generation function” EX1003 (Torre), [0131]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System (Shredder)

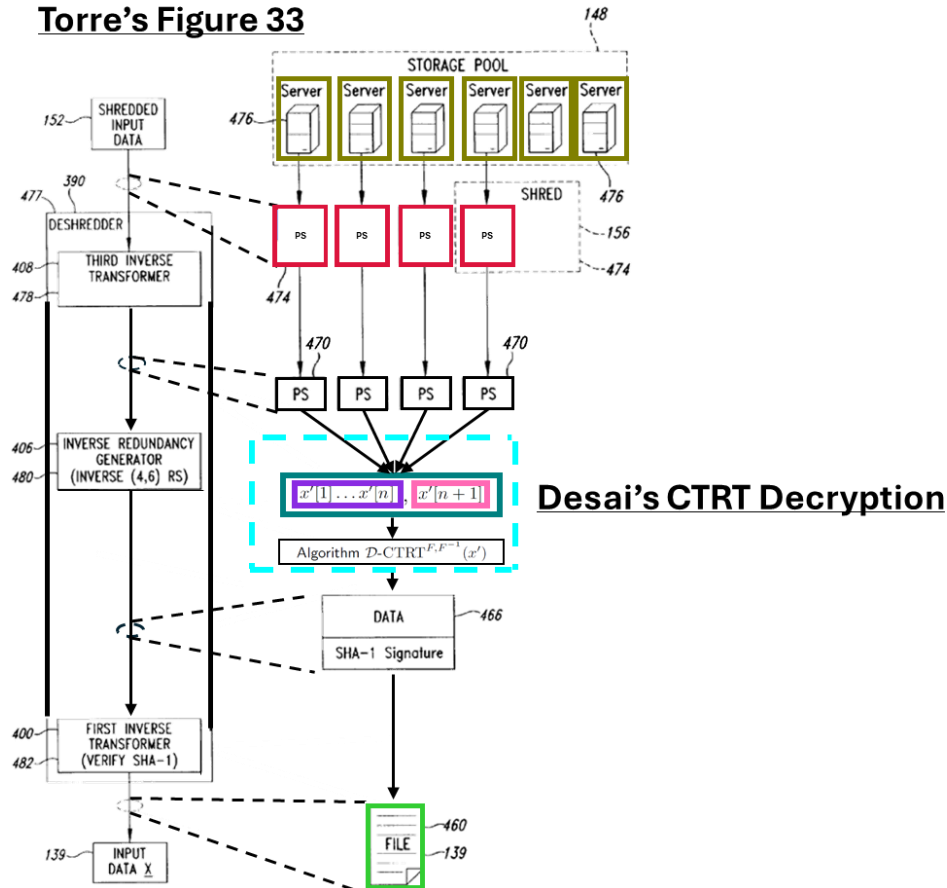


EX1003 (Torre), Fig. 32 (annotated); EX1004 (Desai), 16.

244. Torre's deshredder is designed to reconstruct the original data using only a subset of the stored **shreds**—for example, by retrieving **shreds** from just four of the six **servers** illustrated in Figure 33. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System (Deshredder)

Torre's Figure 33



EX1003 (Torre), FIG. 33 (annotated); EX1004 (Desai), 16.

245. Torre explains that one of the key advantages of being able to reconstruct the original data from fewer than all of the stored shreds is the ability to recover the original data even if one of the servers becomes nonfunctional. As Torre states, “[a] general purpose for the redundancy generator 180 is to make the shredded input data 106 fault-tolerant. The end result of the redundancy generator is to generate data redundant to the input data 139 so that *some of the shredded input*

data 106 that is stored in storage pool 148 could be lost without preventing the input data from being recovered.” EX1003 (Torre), [0111]. As a result, the storage pool 148, which contains the shredded data, “could be viewed as a giant highly fault-tolerant drive in which some of the shreds in some of the storage could be lost without losing function of the giant drive.” EX1003 (Torre), [0069].

246. **Third**, the Torre/Desai System teaches that the claimed “*method [is] performed by a computer system.*” Torre explains that the disclosed functionality may be realized “as one or more **computer programs running on one or more computers** (e.g., as one or more programs running on one or more data processing systems), as one or more programs running on one or more controllers (e.g., microcontrollers)[,] as one or more programs running on one or more processors e.g., microprocessors, as firmware, or as virtually any combination thereof.” EX1003 (Torre), [0141]. *See also* EX1003 (Torre), [0142].

247. **In sum**, the Torre/Desai System teaches “[a] *method performed by a computer system for securing a first data set [Torre’s method for shredding and deshredding input data for secure storage that runs on one or more computers] while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations [Torre’s method for deshredding input data to recover the input data using only a subset of the stored shreds if some of the servers are nonfunctional].*”

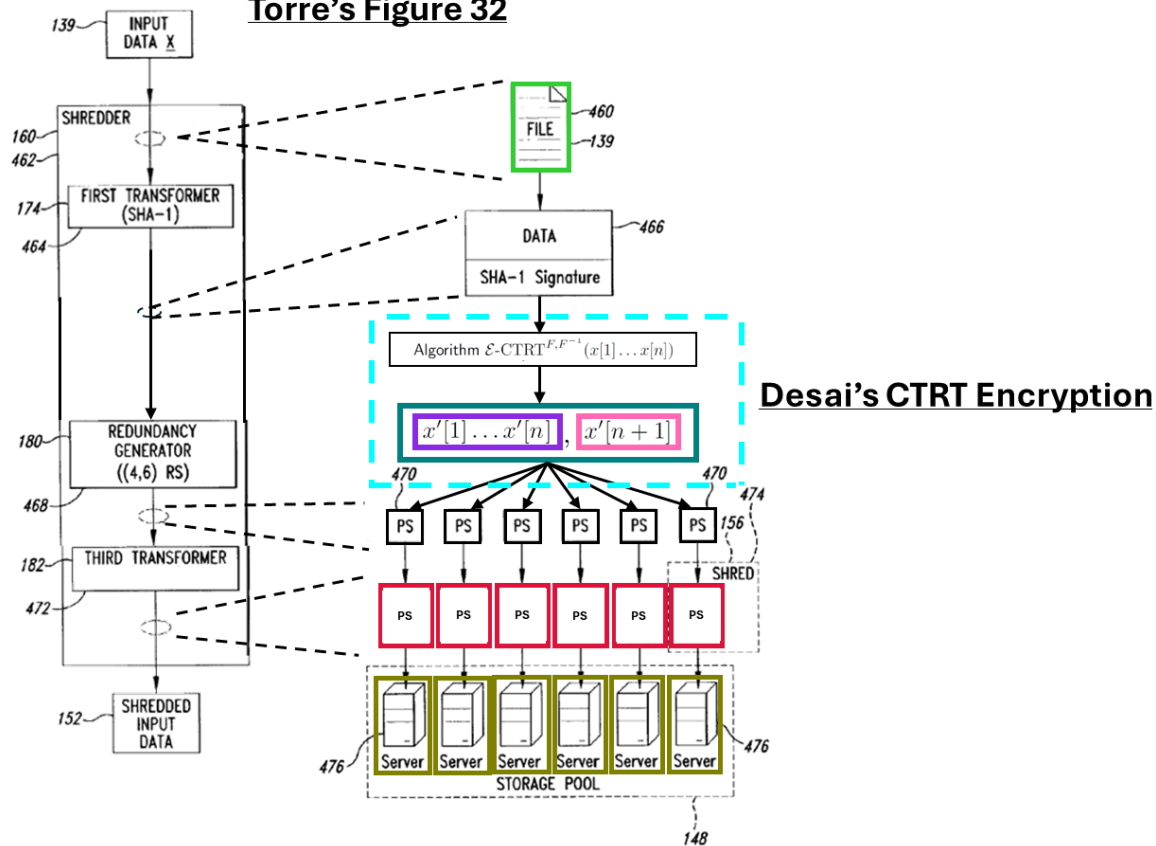
b. [1A] reading the first data set from a memory that stores the first data set;

248. Torre/Desai System teaches claim limitation [1A].

249. The Torre/Desai System teaches a shredder that receives **the first data set**. For instance, in the shredder depicted in the Torre/Desai System, “a first transformer 464” “receives **the input data 139** consisting in this example as a **data file 460**,” which corresponds to the “*first data set*.” EX1003 (Torre), [0131]. *See also* EX1003 (Torre), [0050]. Torre further clarifies that “[t]he **input data** can be a file or another form that has been generated directly or by other means,” and that “[g]eneration of data includes, but is not limited to, generation by operating software programs and by receiving or recording data such as live audio or video, or audio or video broadcast streams.” EX1003 (Torre), [0050]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System (Shredder)

Torre's Figure 32



EX1003 (Torre), FIG 32 (annotated); EX1004 (Desai), 16.

250. A person of ordinary skill in the art would have understood that Torre’s shredder can receive **the first data set**, at least in part, by retrieving it from a memory. This is because processing circuitry such as a shredder typically stores incoming data—such as a video stream—in a memory prior to processing. *See, e.g.*, EX1026 (Tsai), 2 (“For a video server, buffers are used as an intermediate to cache and deliver video data from the storage to the network interface.”); EX1011 (Burger), 1 (“A

computer's *memory system* is the repository for all the information the computer's central processing unit (CPU, or processor) uses and produces."); EX1012 (Understanding Computers), 26 ("[M]emory refers to the temporary internal storage areas within a computer The main type of memory and the most familiar to users is random access memory (RAM). RAM is the same as main memory. A computer can both write data into RAM and read data from RAM."). *See also* EX1005 (Watanabe), [0053] ("Data prepared by the user or provided by the information processing unit itself can be stored in the memory device for use ..."); §V.A above.

251. Indeed, Torre already teaches a memory that can maintain the input data. *See, e.g.*, EX1003 (Torre), [0068] ("[If] a computer has system *memory*, various portions of this *memory* can be used as the storage units 150.") Torre itself teaches the use of memory to maintain the input data. For instance, it states that "[if] a computer has system *memory*, various portions of this *memory* can be used as the storage units 150." EX1003 (Torre), [0068].

252. **In sum**, the Torre/Desai System teaches "*reading the first data set [Torre's input data 139] from a memory that stores the first data set [Torre's memory buffer storing incoming data, such as a video stream].*"

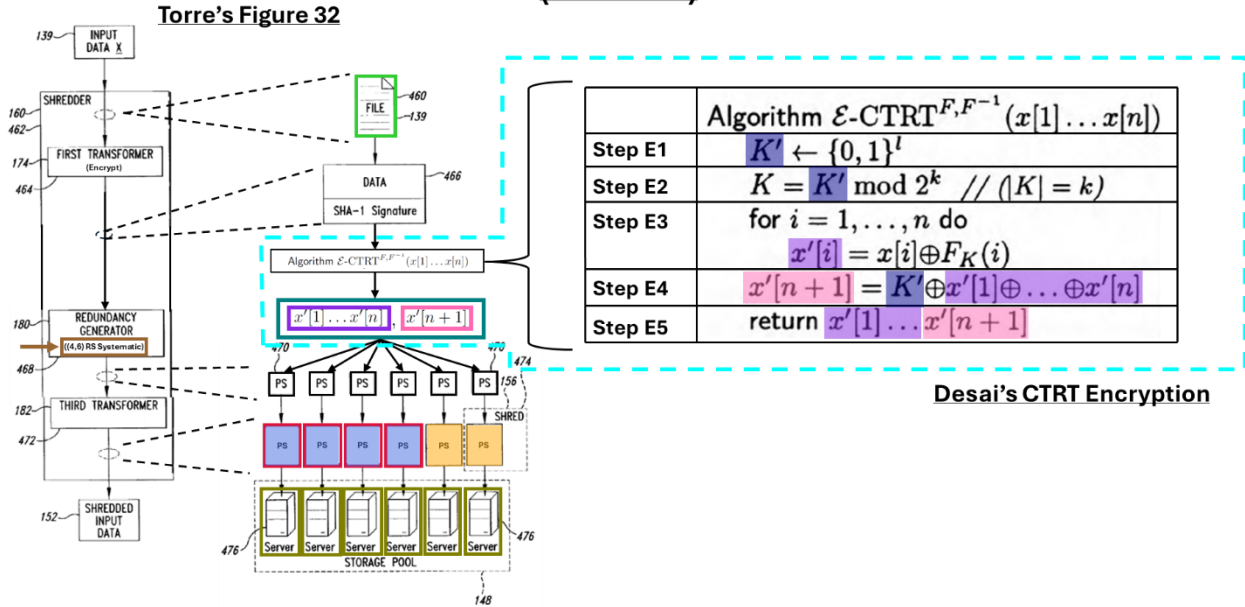
c. [1B] processing the first data set to produce a second data set,

253. Torre/Desai System teaches claim limitation [1B].

254. In the Torre/Desai System, “a first transformer 464” “receives the **input data 139 consisting in this example as a data file 460**,” which constitutes “*a first data set*.” EX1003 (Torre), [0131]. After receiving this input, the first transformer 464 “generate[s] a pre-shred 466 containing **the received data** and a tag of the SHA-1 signature.” EX1003 (Torre), [0131].

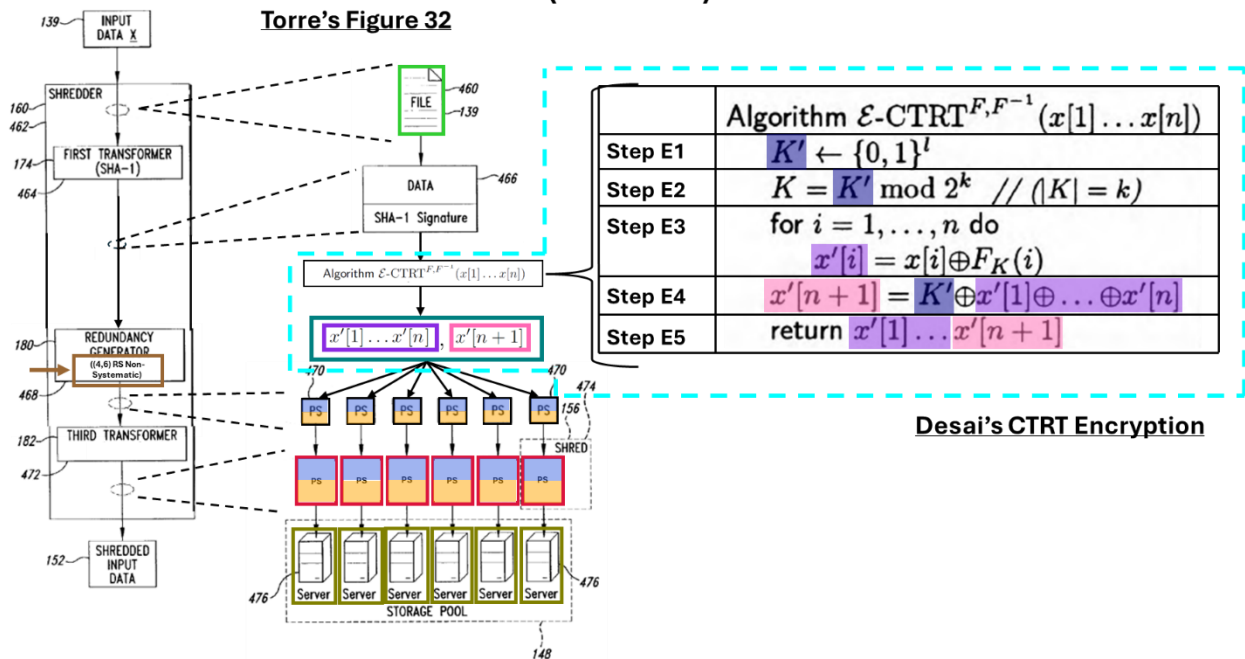
255. In the next step, the shredder in the Torre/Desai System processes the pre-shred 466 by applying Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm. In this step, the pre-shred 466—which contains **the input data** (*i.e.*, the “*first data set*”)—“is regarded as a sequence of l -bit blocks, $x = x[1] \dots x[n]$.” EX1004 (Desai), 16. The $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm then operates on these blocks to produce $x' = x'[1] \dots x'[n+l]$, which constitutes the “*second data set*.” This is shown in the figures below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-1 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

Torre/Desai System-2 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

256. **In sum**, Torre/Desai System teaches “*processing the first data set to produce a second data set* [applying Desai’s $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm to Torre’s **input data 139** to produce data blocks $x' = x'[1] \dots x'[n+1]$].”

d. [1B-1] the processing comprising ... encrypting the first data set using an encryption key to produce an encrypted data set, and

257. Torre/Desai System teaches claim limitation [1B-1].

258. As discussed above, the shredder in the Torre/Desai System includes a first transformer 464 that “generate[s] a pre-shred 466 containing **the input data**” (*i.e.*, the “*first data set*”). EX1003 (Torre), [0131]. This pre-shred 466 is then processed using Desai’s $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm, which operates on **the input data** as illustrated below. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{E}\text{-CTR}^{F,F^{-1}}(x[1] \dots x[n])$
Step E1	$K' \leftarrow \{0, 1\}^l$
Step E2	$K = K' \bmod 2^k \quad // (K = k)$
Step E3	for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$
Step E4	$x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$
Step E5	return $x'[1] \dots x'[n+1]$

EX1004 (Desai), 16. (annotated).

259. Desai's $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm “*encrypt[s] the first data set [the input data] using an encryption key [K'] to produce an encrypted data set [$x'[1] \dots x'[n]$].*” For example, the $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm operates on the pre-shred 466—which contains the input data—by treating it “as a sequence of l -bit blocks, $x = x[1] \dots x[n]$.” EX1004 (Desai), 16.

260. In **step E1**, the algorithm generates a key K' —referred to as the “encryption key”—by producing a random bit sequence of length l . EX1004 (Desai), 16.

261. In **step E2**, the algorithm computes K' modulo 2^k in order to derive the truncated encryption key K , where k corresponds to the key length supported by the cipher F . EX1004 (Desai), 16.

262. In **step E3**, the algorithm encrypts each data block in the sequence $x = x[1] \dots x[n]$ by applying the cipher F —using the key K —to each individual block. This produces the encrypted data blocks $x'[1]$ through $x'[n]$, which constitute “an encrypted data set.” EX1004 (Desai), 16.

263. **In sum**, Torre/Desai teaches “*the processing comprising ... encrypting the first data set [encrypting Torre's input data] using an encryption key [Desai's encryption key K'] to produce an encrypted data set [Desai's encrypted data blocks $x'[1]$ through $x'[n]$].*”

- e. [1B-2] the processing comprising ... logically combining at least a portion of the encrypted data set with the encryption key to produce a resultant,

264. Torre/Desai System teaches claim limitation [1B-2].

265. For example, following **step E3** as described above in §IX.A.5.d above, **step E4** of Desai's $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm performs an exclusive-OR operation ("XOR," denoted by the symbol " $\oplus$ ") between the encryption key K' and each of the encrypted data blocks $x'[1]$ through $x'[n]$ —i.e., "at least a portion of the encrypted data set." This operation yields a new data block $x'[n+1]$, which constitutes "a resultant." EX1004 (Desai), 16. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{E}\text{-CTRT}^{F,F^{-1}}(x[1] \dots x[n])$
Step E1	$K' \leftarrow \{0, 1\}^l$
Step E2	$K = K' \bmod 2^k \quad // (K = k)$
Step E3	for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$
Step E4	$x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$
Step E5	return $x'[1] \dots x'[n+1]$

EX1004 (Desai), 16 (annotated).

266. The XOR operation performed in **step E4** corresponds to the claimed "logically combining" operation, as XOR is a logical operator. See, e.g., EX1023

(Matloff), 3, 5-6 (“The term combinational **logic** refers to circuitry that transforms bits, as opposed to storing bits ... Here we have inputs X and Y, output Z, with Z being equal to 1 if the statement ‘X = 1 or Y = 1 but not both’ is true. The term used for this is ‘exclusive-or,’ abbreviated to **XOR**.”). *See also* §V.C above.

267. Furthermore, claim 11 of the ’456 Patent expressly states that “logically combining comprises logically combining using an exclusive-OR operation.” EX1001 (’456 Patent), Claim 11.

268. **In sum**, Torre/Desai System teaches “*the processing comprising ... logically combining [combining using Desai’s XOR operator] at least a portion of the encrypted data set [Desai’s encrypted data blocks $x'[1]$ through $x'[n]$] with the encryption key [Desai’s encryption key K'] to produce a resultant [Desai’s data block $x'[n+1]$].*”

- f. **[1B-3] wherein the second data set comprises the encrypted data set and the resultant, and wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant;**

269. Torre/Desai System teaches claim limitation [1B-3].

270. **First**, Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm discloses the limitation “*wherein the second data set comprises the encrypted data set and the resultant.*” Specifically, in **step E5**, the algorithm returns $x' = x'[1] \dots x'[n+1]$ (the “second data set”) comprising of $x'[1]$ through $x'[n]$ (the “encrypted data set”) and

$x'[n+1]$ (the “*resultant*”). An annotated version of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm is shown below. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{E}\text{-CTRT}^{F,F^{-1}}(x[1] \dots x[n])$
Step E1	$K' \leftarrow \{0, 1\}^l$
Step E2	$K = K' \bmod 2^k \quad // (K = k)$
Step E3	for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$
Step E4	$x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$
Step E5	return $x'[1] \dots x'[n+1]$

EX1004 (Desai), 16 (annotated).

271. **Second**, Desai’s $D\text{-CTRT}^{F,F^{-1}}$ transformation algorithm discloses the limitation “*wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant.*” An annotated version of Desai’s $D\text{-CTRT}^{F,F^{-1}}$ transformation algorithm is provided below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{D}\text{-CTRTR}^{F,F^{-1}}(x')$
Step D1	Parse x' as $x'[1] \dots x'[n+1]$
Step D2	$K' = x'[1] \oplus \dots \oplus x'[n+1]$
Step D3	$K = K' \bmod 2^k$ // ($ K = k$)
Step D4	for $i = 1, \dots, n$ do $x[i] = x'[i] \oplus F_K(i)$
Step D5	return $x[1] \dots x[n]$

EX1004 (Desai), 16 (annotated).

272. In **step D1**, the algorithm “[p]arse[s] x' as $x'[1] \dots x'[n+1]$,” which corresponds to the “*second data set*.” EX1004 (Desai), 16. Then, in **step D2**, the algorithm recovers K' (the “*encryption key*”) by performing an XOR operation (denoted by the symbol “ $\oplus$ ”) between $x'[1]$ through $x'[n]$ (i.e., “*at least a portion of the encrypted data set*”) and $x'[n+1]$ (the “*resultant*”). EX1004 (Desai), 16.

273. In this context, the algorithm requires access to all of the encrypted data blocks $x'[1]$ through $x'[n]$, along with the resultant block $x'[n+1]$, in order to reverse the XOR operation and recover the *encryption key* K' . Desai explains that an AONT is “an efficiently computable transformation, mapping sequences of blocks to sequences of blocks,” and that “[g]iven all but one of the output sequence of blocks, it is computationally infeasible to determine any function of any input block.” EX1004 (Desai), 7.

274. Indeed, this is the defining characteristic of an all-or-nothing transform. *See, e.g.,* EX1010 (Rivest), 5 (“It is easy to see that the package transform is invertible ... We also note that if any block of the pseudo-message sequence is unknown, then K' can not be computed, and so it is infeasible to compute any message block.”), 3 (“It is computationally infeasible to compute any function of any message block if any one of the pseudo-message blocks is unknown.”).

275. **In sum**, Torre/Desai System teaches “*wherein the second data set [Desai’s data blocks $x'[1]$... $x'[n+1]$] comprises the encrypted data set [Desai’s encrypted data blocks $x'[1]$ through $x'[n]$] and the resultant [Desai’s data block $x'[n+1]$], and wherein all of the at least a portion of the encrypted data set is required to recover the encryption key based on the resultant [Desai requires performing an XOR between all of the encrypted data blocks $x'[1]$ through $x'[n]$ and the resultant data block $x'[n+1]$ to recover the encryption key K].*”

g. [1C] producing redundancy information based on information in the second data set;

276. Both Torre/Desai System-1, which utilizes systematic encoding for redundancy generation, and Torre/Desai System-2, which employs non-systematic encoding, satisfy the requirements of claim limitation [1C].

i. Torre/Desai System-1

277. Torre/Desai System-1, which applies systematic encoding for redundancy generation, teaches the limitation of “*producing redundancy*

information based on information in the second data set.” In Torre/Desai System-1, the shredder first uses Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm to process the pre-shred 466—containing **the input data**—to generate $x' = x'[1] \dots x'[n+1]$, which constitutes the “*second data set*.” The redundancy generator 468 then receives the second data set $x' = x'[1] \dots x'[n+1]$ and applies systematic encoding to produce redundancy information. For example, Torre explains that “[t]he redundancy generator 180 also generates **two outputs of redundant data** from **the received data** and outputs **the two outputs of redundant data** along with the **four outputs** of **the received data**.” EX1003 (Torre), [0117].

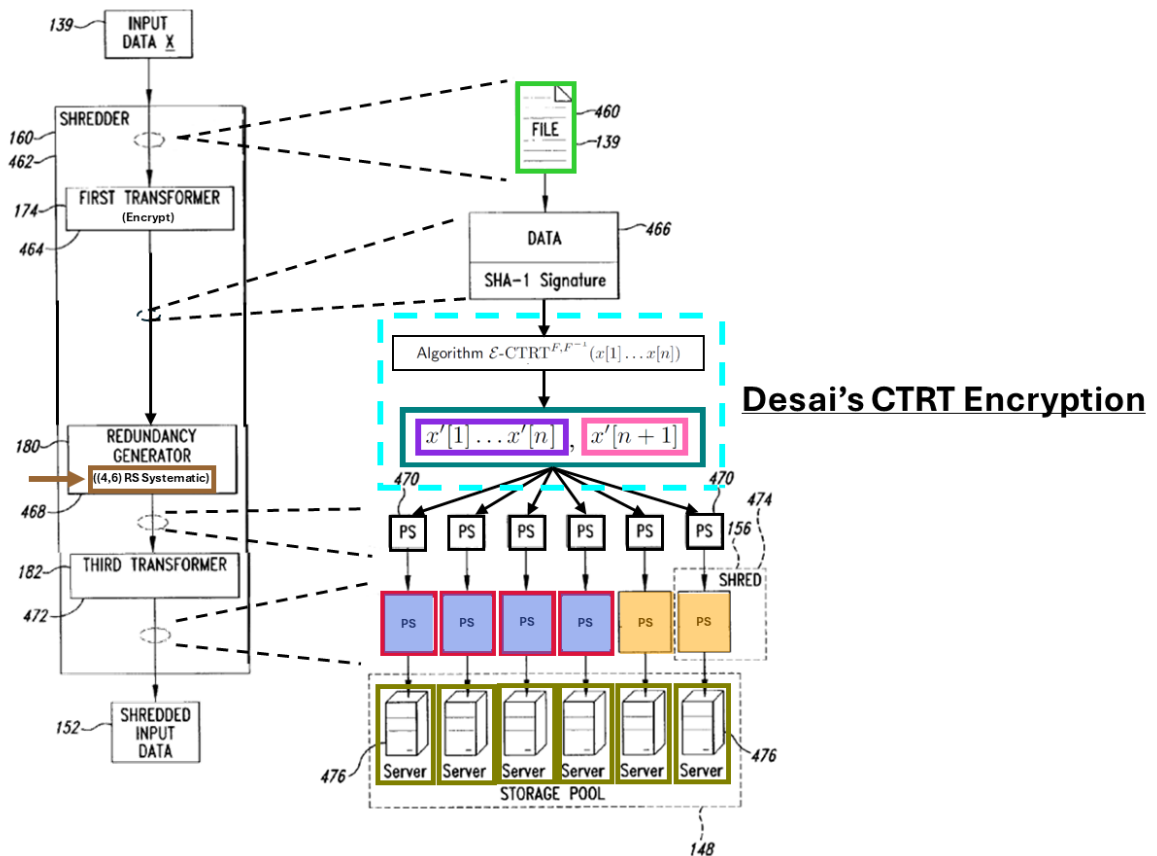
278. Torre further states that “[t]he redundancy generator 468 ... uses a 4 input-6 output **Reed Solomon redundancy** generation function with systematic encoding to generate six pre-shreds 470 with **four of the pre-shreds** containing direct pass-through data blocks of the data file and **two of the pre-shreds** being generated from combinations of data blocks from the data ***file for redundancy purposes***.” EX1003 (Torre), [0131]. *See also* Torre’s Figure 21 and §IX.A.2.a above [Torre System-1] (describing using systematic encoding for redundancy generation).

279. As illustrated in an annotated version of Figure 32 from Torre shown below, the **redundancy information** corresponds to two of the six shreds produced by the 4 input-6 output **Reed Solomon redundancy** generator, which utilizes

systematic encoding. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-1 **(Shredder)**

Torre's Figure 32



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

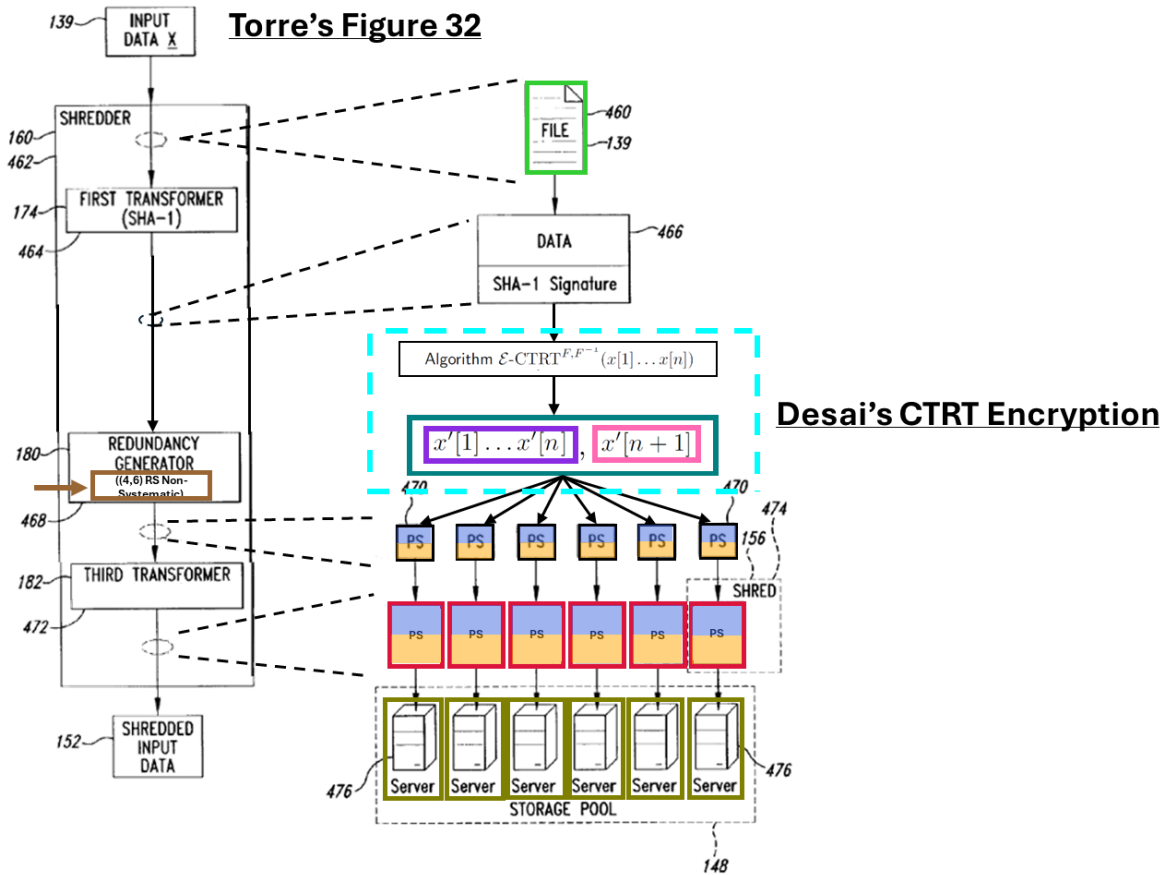
280. **In sum**, Torre/Desai System-1 teaches “*producing redundancy information [Torre’s two shreds 474 containing **redundant data**] based on information in the second data set [based on information in Desai’s data blocks $x'[1]$ through $x'[n+1]$].*”

ii. **Torre/Desai System-2**

281. Torre/Desai System-2, which uses non-systematic encoding for redundancy generation, also satisfies the limitation of “*producing redundancy information based on information in the second data set.*” In this configuration, the shredder processes the input data using Desai’s $E\text{-CTR}^{F,F-1}$ transformation algorithm to generate $x' = x'[1] \dots x'[n+1]$, which constitutes the “*second data set.*” The redundancy generator 468 then generates redundancy information based on this second data set using **non-systematic** encoding. Torre explains that, in this mode, the redundancy generator “generates six outputs of redundancy-generated data from four inputs of received data.” EX1003 (Torre), [0117]; *see also* EX1003 (Torre), FIG. 22.

282. As illustrated in the annotated version of Figure 32 provided below, the redundancy information forms part of the six shreds generated by the 4 input–6 output Reed Solomon redundancy generator operating with non-systematic encoding. *See* EX1022 (Primer: Reed-Solomon ECC); EX1019 (Costello), 8 (“In the case of a non-systematic encoder, every one of the n columns of the generator matrix generates a ‘parity’ sequence.”). This is shown in Torre’s Figure 32 below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-2 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

283. **In sum**, Torre/Desai System-2 teaches “*producing redundancy information [Torre’s six shreds 474 containing redundant data] based on information in the second data set [based on information in Desai’s data blocks $x'[1]$ through $x'[n+1]$].*”

- h. [1D] producing a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set; and

284. Both Torre/Desai System-1, which employs systematic encoding for redundancy generation, and Torre/Desai System-2, which uses non-systematic encoding, teach limitation [1D].

- i. **Torre/Desai System-1**

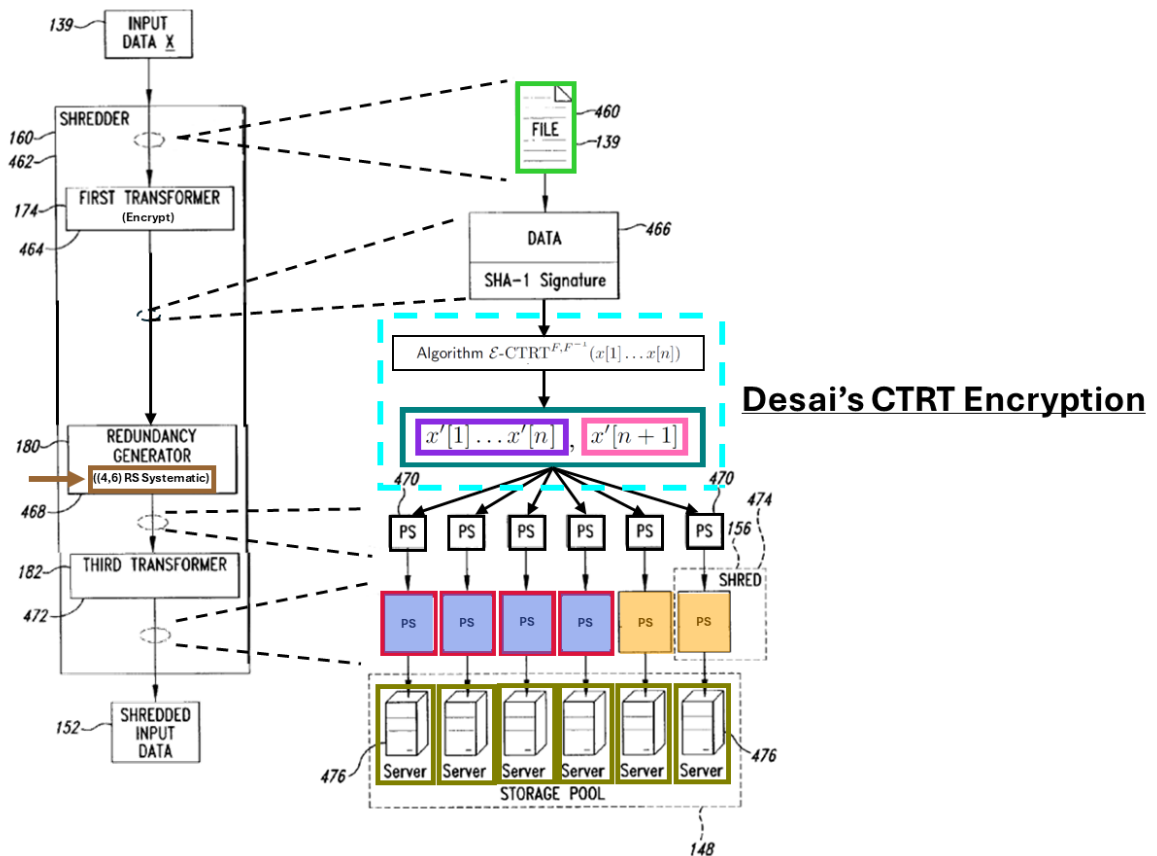
285. *First*, the shredder in Torre/Desai System-1 “*produc[es] a plurality of n shares from the second data set.*” As discussed in §IX.A.4.a above, Torre/Desai System-1 includes a redundancy generator 468 that uses systematic encoding to process the “*second data set*” $x' = x'[1] \dots x'[n+1]$. This processing step “generate[s] six pre-shreds 470 with *four of the pre-shreds* containing direct pass-through data blocks of *the data file.*” EX1003 (Torre), [0131].

286. The “*four of the pre-shreds*” referenced just above are subsequently processed by the shredder’s “third transformer(s) 472,” and “form *shreds* 474, which includes the pre-shred data.” EX1003 (Torre), [0132]. As illustrated in Figure 32 of Torre (annotated below), each resulting *shred* contains pre-shred data corresponding to one of the original *pre-shreds*. Accordingly, the third transformer produces *four shreds* 474, each of which includes “*direct pass-through data blocks*” derived from the “second data set” $x' = x'[1] \dots x'[n+1]$. EX1003 (Torre), [0132].

This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-1 **(Shredder)**

Torre's Figure 32



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

287. **Second**, in Torre/Desai System-1, “*each of the n shares comprises at least some of the second data set.*” As discussed earlier in this Section, these **four shreds** each include pre-shred data derived from a corresponding one of the **pre-shreds**, which themselves contain data from the “*second data set*” *second data set*”

$x' = x'[1] \dots x'[n+1]$. Torre explains that “[t]he redundancy generator 468 receives the data file 460 and uses a 4 input-6 output Reed Solomon redundancy generation function with systematic encoding to generate six pre-shreds 470 with four of the pre-shreds containing direct pass-through data blocks of the data file and two of the pre-shreds being generated from combinations of data blocks from the data file for redundancy purposes.” EX1003 (Torre), [0131].

288. **In sum**, Torre/Desai System-1 teaches “*producing a plurality of n shares from the second data set* [Torre’s third transformer 472 produces four shreds 474 from Desai’s data blocks $x' = x'[1] \dots x'[n+1]$], *wherein each of the n shares comprises at least some of the second data set* [each of Torre’s four shreds 474 comprise at least some of the data from Desai’s data blocks $x' = x'[1] \dots x'[n+1]$].”

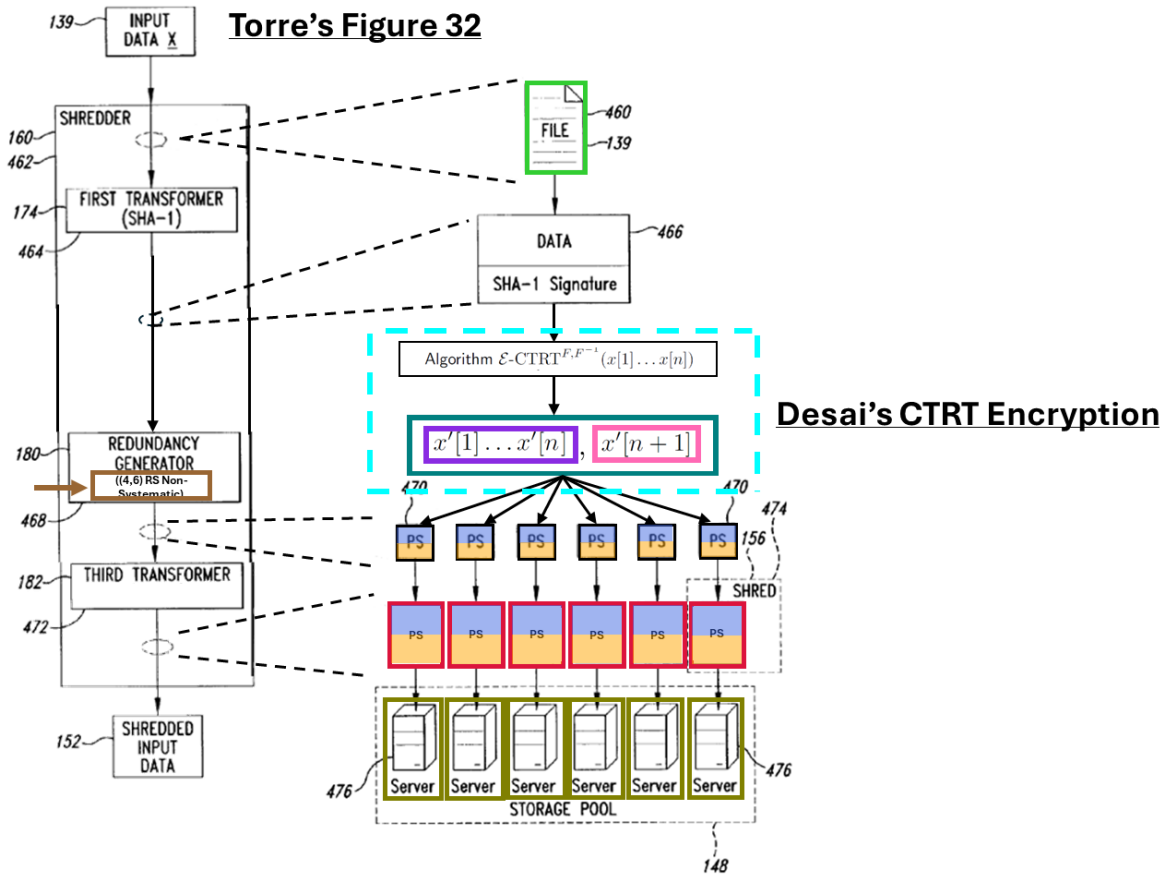
ii. Torre/Desai System-2

289. **First**, the shredder in Torre/Desai System-2 “*produc[es] a plurality of n shares from the second data set.*” As described in §IX.A.2.b above, Torre/Desai System-2 includes a redundancy generator 468 that applies non-systematic encoding to process the second data set $x' = x'[1] \dots x'[n+1]$. This processing step “generate[s] six pre-shreds 470,” **each** of which contains a combination of pass-through data blocks from the received data and redundancy information. EX1003 (Torre), [0131], [0117]. See EX1022 (Primer: Reed-Solomon ECC), EX1019 (Costello), 8 (“In the

case of a non-systematic encoder, every one of the n columns of the generator matrix generates a ‘parity’ sequence.”).

290. The “**six pre-shreds**” generated by the redundancy generator are then further processed by the shredder’s “third transformer(s) 472,” which “form **shreds** 474, which includes the pre-shred data.” EX1003 (Torre), [0132]. As depicted in Figure 32 of Torre (annotated below), each **shred** contains pre-shred data derived from a corresponding one of the **pre-shreds**. As a result, the third transformer produces **six shreds 474**, each of which includes both **pass-through data blocks** from the **received data**—pre-processed into the “*second data set*” $x' = x'[1] \dots x'[n+1]$ —as well as **redundancy information**. EX1003 (Torre), [0132]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-2 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

291. **Second**, in Torre/Desai System-2, “*each of the n shares comprises at least some of the second data set.*” As explained above in this Section, the **six shreds** include pre-shred data from a corresponding one of the **pre-shreds**, which, in turn, contain data from the “*second data set*” $x' = x'[1] \dots x'[n+1]$. EX1003 (Torre), [0131], [0117] (“An example of non-systematic encoding is shown in FIG. 22 where the redundancy generator 180 generates six outputs of redundancy-generated data from four inputs of received data.”).

292. **In sum**, Torre/Desai System-2 teaches “*producing a plurality of n shares from the second data set* [Torre’s third transformer 472 produces six **shreds** 474 from Desai’s data blocks $x' = x'[1] \dots x'[n+1]$], *wherein each of the n shares comprises at least some of the second data set* [each of Torre’s six **shreds** 474 comprise at least some of the **data** from Desai’s data blocks $x' = x'[1] \dots x'[n+1]$].”

- i. [1E] **storing the plurality of n shares and the redundancy information**, wherein at least some of the **n shares** are stored in **separate storage locations** and **the first data set** is recoverable using at least a threshold number, less than n , of the **plurality of n shares**

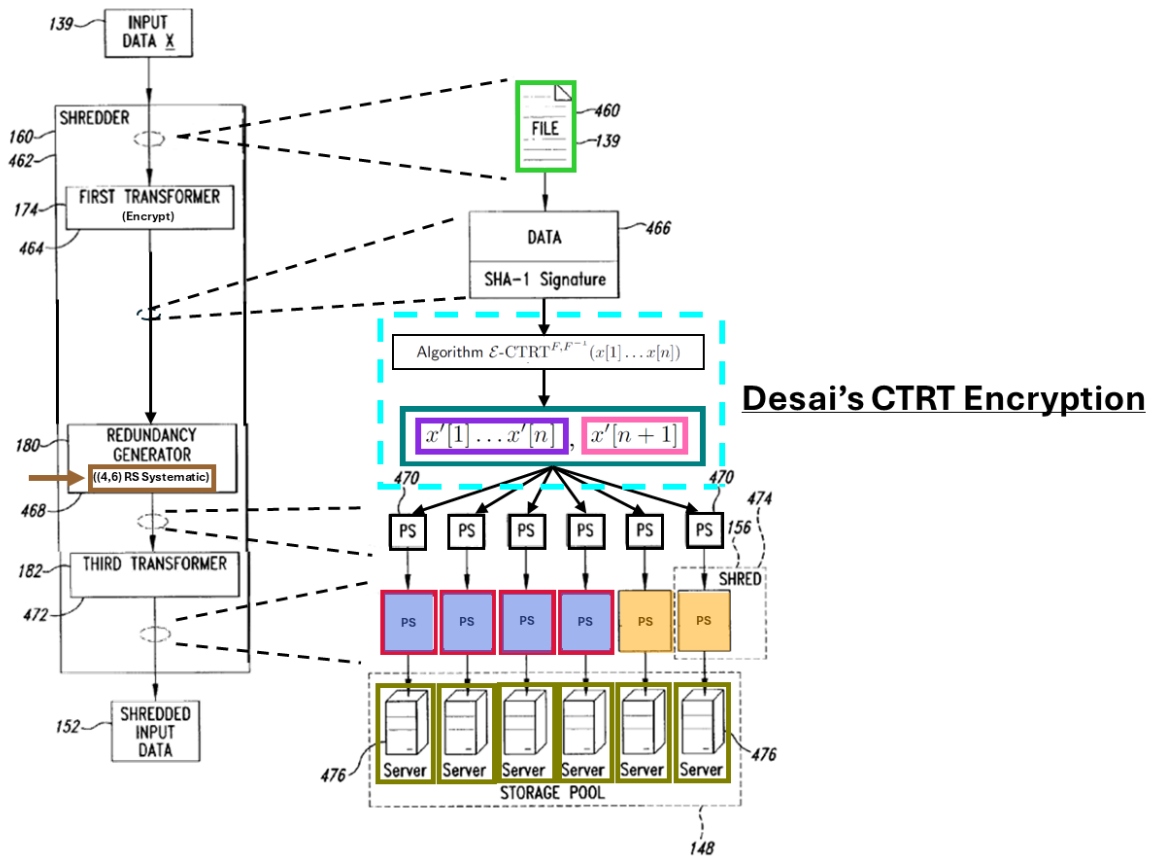
i. **Torre/Desai System-1**

293. Torre/Desai System-1 teaches claim limitation [1E].

294. **First**, Torre/Desai System-1 teaches the step of “*storing the plurality of n shares and the redundancy information, wherein at least some of the n shares are stored in separate storage locations.*” In Torre/Desai System-1, the shredder produces $N = 4$ **shreds** that include **pass-through data from the second data set** (the “*plurality of n shares*”) and two additional shreds that carry **redundancy information** (the “*redundancy information*”). These six **shreds** are then “sent to the storage pool 146 where in this instance is made up of **six servers** 476.” EX1003 (Torre), [0132]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-1 (Shredder)

Torre's Figure 32



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

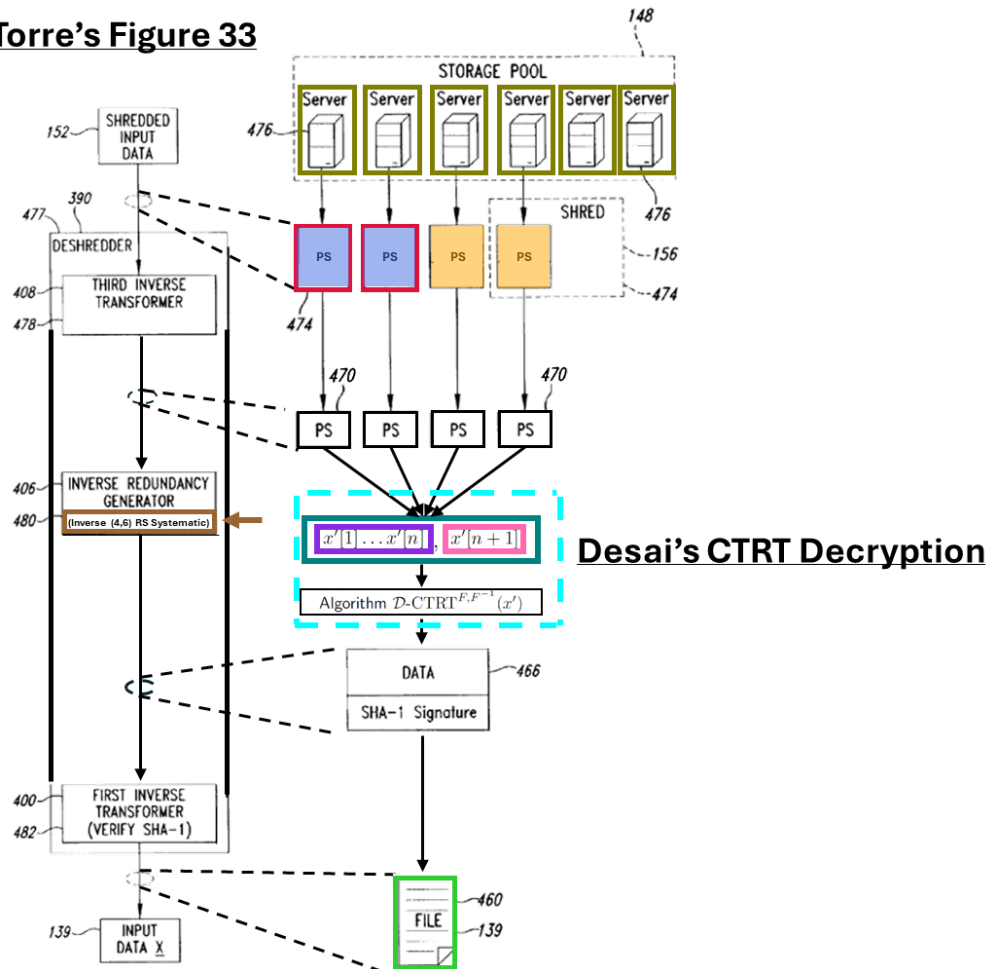
295. Torre further explains that “[f]or higher reliability, the **storage units** [such as the **servers 476** in the storage pool 148] could be **physically located separately from one another**, typically linked to the shredder by one of the interconnects 144 that is a communication network.” EX1003 (Torre), [0069]. *See also* EX1003 (Torre), [0052] (“Storage of the **shredded input data** ... may involve sending it ... to **multiple storage units** such as ... multiple hard disks on a single

server or *multiple servers*, or multiple storage devices, other than servers, directly attached to a network.”). Based on these disclosures, a POSITA would have understood that the Torre/Desai System-1 stores the *plurality of shares* in *separate servers* that are “*physically located separately from one another.*” EX1003 (Torre), [0069].

296. **Second**, Torre/Desai System-1 teaches that “*the first data set is recoverable using at least a threshold number, less than n, of the plurality of n shares.*” The deshredder in Torre/Desai System-1 is able to use only four of the six *shreds* stored in the *servers* to recover *the original input data* (i.e., the “*first data set*”). For example, Torre states that “[c]omplementary to the shredder 462, an exemplary depiction of a deshredder 477 that deshreds without de-allocation, shown in FIG. 33, receives four of the *shreds* 474 from four of the *servers* 476.” EX1003 (Torre), [0133]. Based on this, the deshredder requires only *at minimum two* of the *four shreds* containing *pass-through data from the second data set* (“*at least a threshold number [two], less than n [four], of the plurality of n shares*”) and the two additional shreds that carry *redundancy information* to reconstruct the pre-shred 466, which carries the original *input data*. This process is further illustrated in the annotated version of Figure 33 provided below. EX1003 (Torre), [0133]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-1 (Deshredder)

Torre's Figure 33



EX1003 (Torre), FIG. 33 (annotated); EX1004 (Desai), 16.

297. **In sum**, Torre/Desai System-1 teaches “*storing* [Torre stores data on servers 476] *the plurality of n shares* [Torre’s four shreds containing pass-through data from the second data set] *and the redundancy information* [Torre’s two additional shreds that carry redundancy information], *wherein at least some of the n shares are stored in separate storage locations* [Torre’s four shreds containing pass-through data from the second data set are stored in separate servers 476] *and the first*

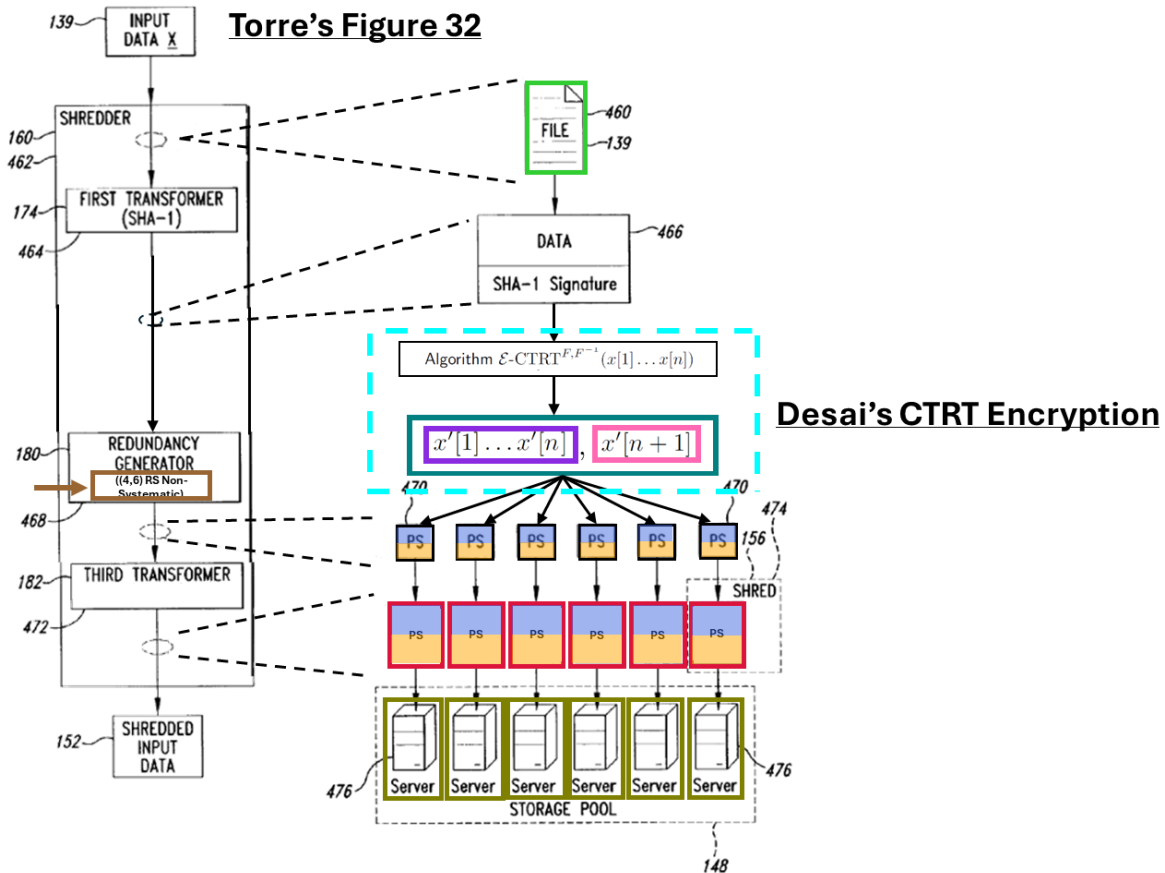
data set [Torre's input data] is recoverable using at least a threshold number, less than n , of the plurality of n shares [Torre can reconstruct the pre-shred 466 containing the original input data using only two of the four shreds containing pass-through data from the second data set]."

ii. Torre/Desai System-2

298. Torre/Desai System-2 teaches claim limitation [1E].

299. **First**, Torre/Desai System-2 teaches the limitation of "*storing the plurality of n shares and the redundancy information, wherein at least some of the n shares are stored in separate storage locations.*" In Torre/Desai System-2, the shredder produces $N = 6$ shreds, each of which includes both pass-through data from the data set and redundancy information. These six shreds are stored in six distinct servers 476, as illustrated in the annotated version of Figure 32 provided below. EX1003 (Torre), [0132]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-2 (Shredder)



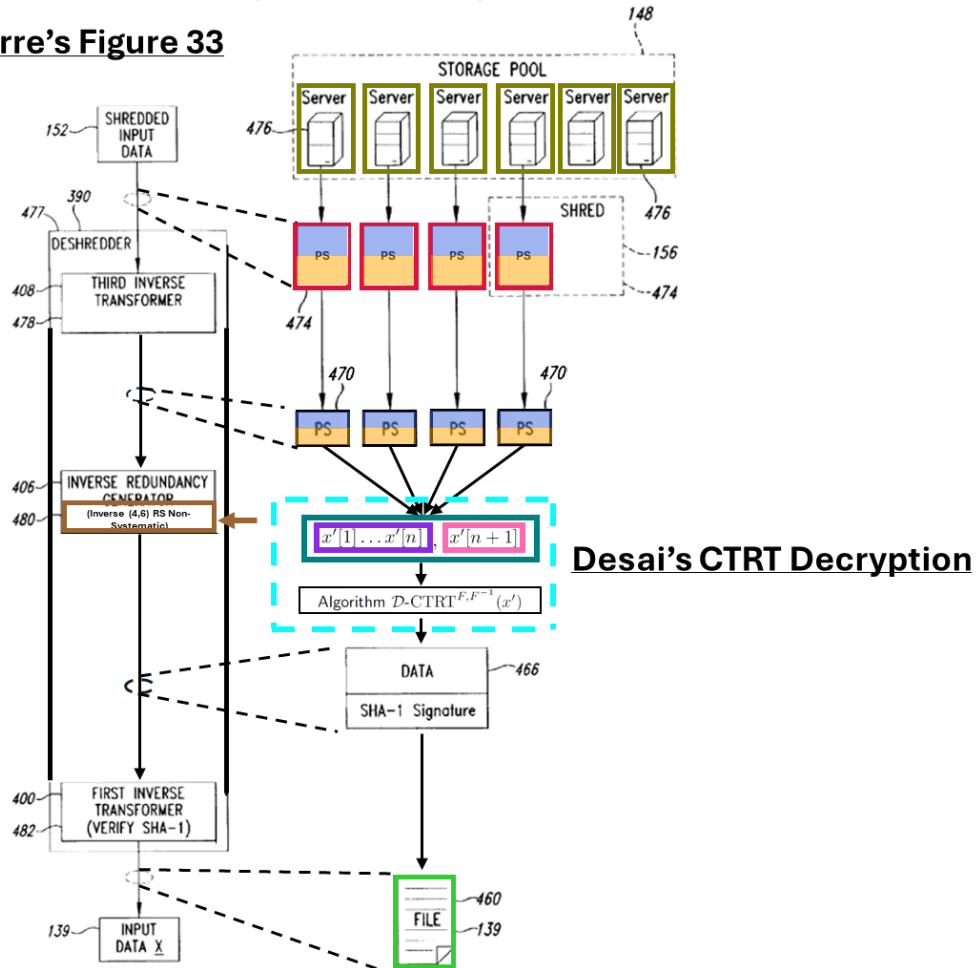
EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

300. As explained in §IX.A.5.i.i above [1E, Torre/Desai System-1], “[f]or higher reliability, the **storage units** [such as the **servers 476** in the storage pool 148] could be **physically located separately from one another**.” EX1003 (Torre), [0069]. See also EX1003 (Torre), [0052]. The same rationale applies to Torre/Desai System-2, which distributes the plurality of shares—the six **shreds**—in respective **servers**. These **servers** are “physically located separately from one another.” EX1003 (Torre), [0069].

301. **Second**, Torre/Desai System-2 teaches the limitation that “*the first data set is recoverable using at least a threshold number, less than n , of the plurality of n shares.*” Specifically, the deshredder in Torre/Desai System-2 is capable of reconstructing **the original input data** (*i.e.*, the “first data set”) using only four of **the six shreds** stored across the **servers**. EX1003 (Torre), [0133]. Thus, the system requires “*at least a threshold number [four], less than n [six], of the plurality of n shares,*” which include both **pass-through data from the data set** and **redundancy information**. These shares are used to recover the pre-shred 466 containing **the original input data**, as illustrated in the annotated figure below. EX1003 (Torre), [0133]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-2 (Deshredder)

Torre's Figure 33



EX1003 (Torre), FIG. 33 (annotated); EX1004 (Desai), 16.

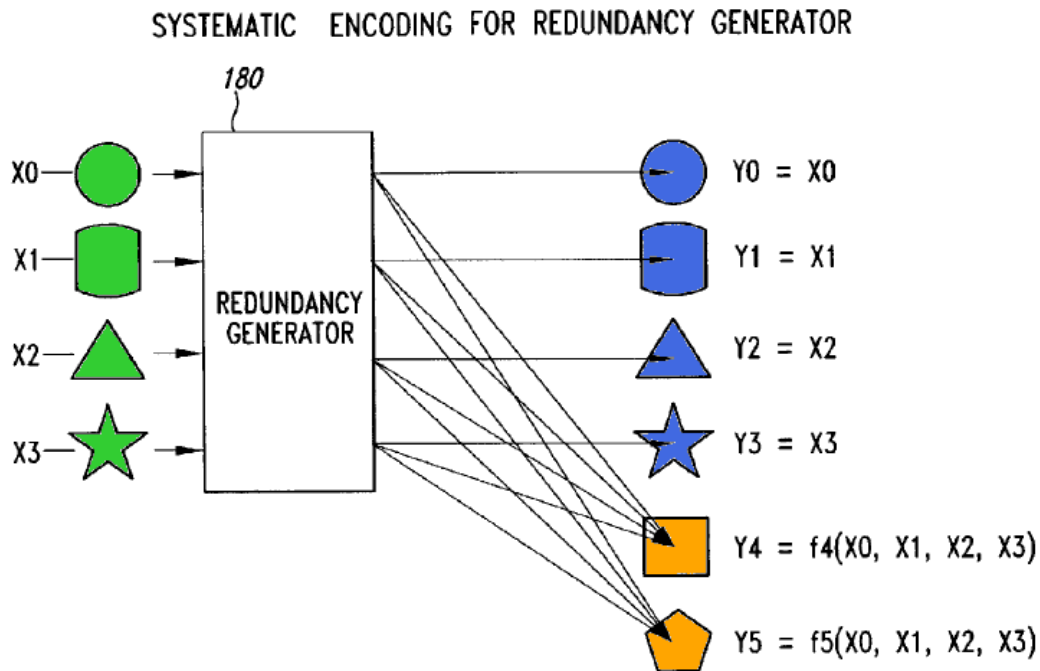
302. **In sum**, Torre/Desai System-2 teaches “*storing the plurality of n shares and the redundancy information* [Torre stores the **six shreds** which each include both **pass-through data from the data set** and **redundancy information** on **servers 476**], *wherein at least some of the n shares are stored in separate storage locations* [Torre’s **six shreds** are stored in separate **servers 476**] *and the first data set* [Torre’s **input data**] *is recoverable using at least a threshold number, less than n , of the*

plurality of n shares [Torre can reconstruct the pre-shred 466 containing **the original input data** using only **four** of the six **shreds**].”

6. **Claim 2: The method of claim 1, wherein each of the n shares further comprises at least some of the redundancy information and storing the plurality of n shares and the redundancy information comprises storing the redundancy information in the n shares.**

303. Torre/Desai System-2 teaches the additional limitation of Claim [2].

304. **First**, Torre/Desai System-2 teaches the limitation “*wherein each of the n shares further comprises at least some of the redundancy information.*” In Torre/Desai System-2, the shredder uses a redundancy generator that implements **non-systematic encoding**, which “generates six outputs of **redundancy-generated data** from four inputs of **received data**.” EX1003 (Torre), [0131], [0116]–[0117]. A person of ordinary skill in the art would have understood that, under non-systematic encoding, each output contains embedded **redundancy information**. See EX1022 (Primer: Reed-Solomon ECC), EX1019 (Costello), 8 (“In the case of a non-systematic encoder, every one of the n columns of the generator matrix generates a ‘parity’ sequence.”). This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.



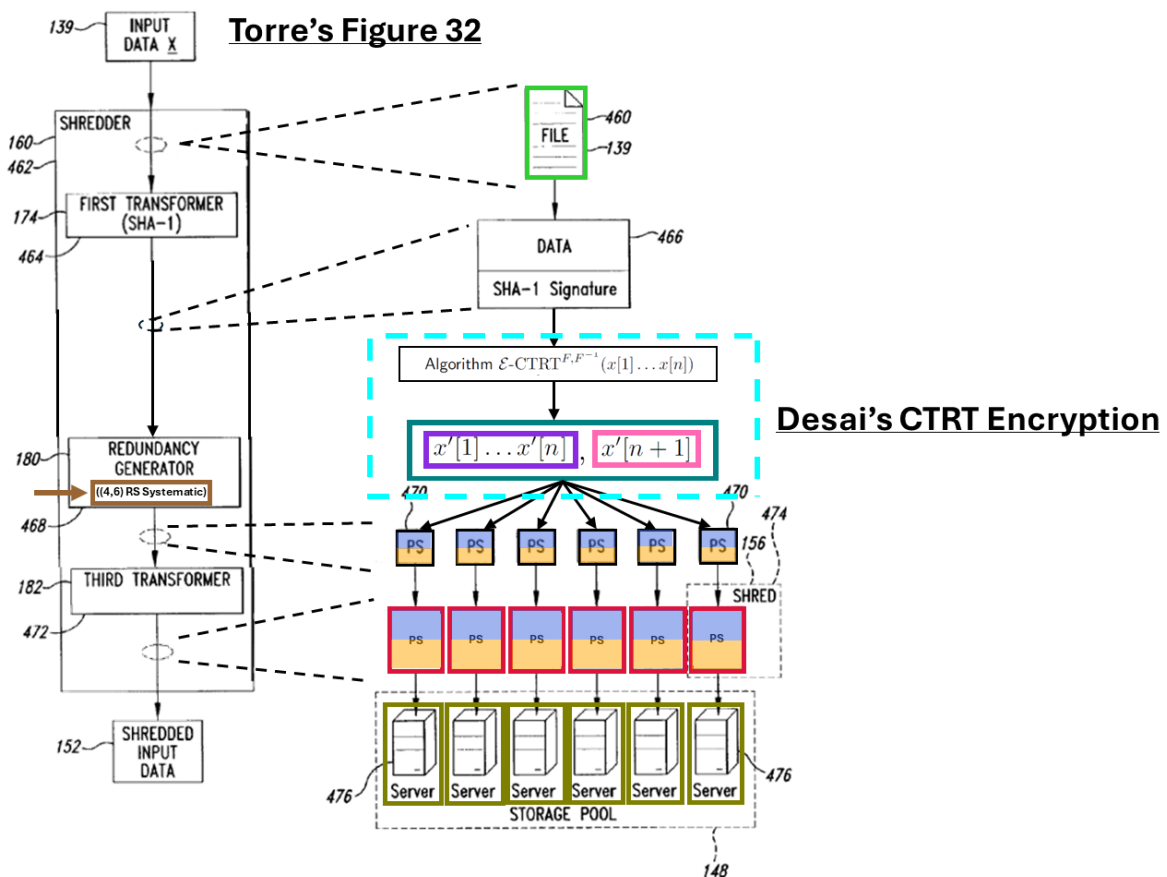
EX1003 (Torre), FIG. 22 (annotated).

305. **Second**, Torre/Desai System-2 also teaches the limitation that “*storing the plurality of n shares and the redundancy information comprises storing the redundancy information in the n shares.*” In Torre/Desai System-2, the shredder includes a redundancy generator that produces six pre-shreds 470 that each carry **redundancy information**. EX1003 (Torre), [0117], [0131]. See EX1022 (Primer: Reed-Solomon ECC), EX1019 (Costello), 8 (“In the case of a non-systematic encoder, every one of the n columns of the generator matrix generates a ‘parity’ sequence.”).

306. These pre-shreds 470—each including at least a portion of the **redundancy information**—are then used by the third transformer 472 to form **six**

shreds 474, thereby “storing the redundancy information in the n shares [six shreds].” EX1003 (Torre), [0132] (“The third transformer(s) ... form shreds 474, which includes the pre-shred data.”). These shreds are subsequently stored in servers 476, as illustrated in the annotated figure below. EX1003 (Torre), [0132]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

Torre/Desai System-2 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1004 (Desai), 16.

307. **In sum**, Torre/Desai System-2 teaches “*wherein each of the n shares [Torre’s six shreds 474] further comprises at least some of the redundancy information [Torre’s six shreds 474 each carry some of the redundancy information from the six pre-shreds 470 generated by the redundancy generator] and storing the plurality of n shares and the redundancy information comprises storing the redundancy information in the n shares [Torre’s third transformer 472 forms six shreds 474 that each include redundancy information from the preshreds 470, and the six shreds 474 are stored in servers 476].*”

- 7. Claim 3: The method of claim 2, wherein storing the redundancy information in the n shares comprises including in each of the n shares information redundant to information included in at least one other of the n shares.**

308. Torre/Desai System-2 teaches the additional limitation of Claim [3].

309. For example, the shredder in Torre/Desai System-2 employs a redundancy generator that applies non-systematic encoding to “generate[] six outputs of redundancy-generated data from four inputs of received data.” EX1003 (Torre), [0117], [0131]. This process is illustrated in Figure 22 of Torre, shown below, which depicts the application of non-systematic encoding to produce outputs that each contain embedded redundancy information. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

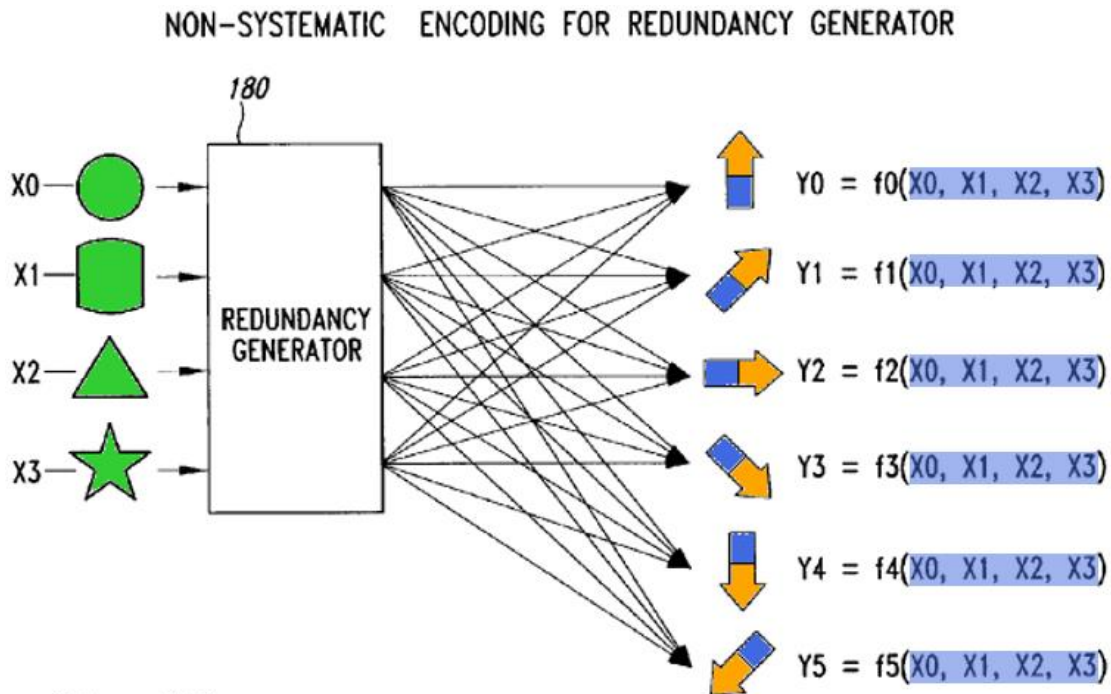


Fig. 22

EX1003 (Torre), FIG. 22 (annotated).

310. Figure 22 demonstrates that each of the “six outputs of redundancy-generated data—i.e., six pre-shreds (Y_0 – Y_5)—generated by the redundancy generator “from four inputs of received data” contains information that is redundant with respect to the others. EX1003 (Torre), [0117]. For example, pre-shred Y_0 is computed as a function of the input blocks X_0 , X_1 , X_2 , and X_3 . Similarly, pre-shred Y_1 is also derived from those same four input blocks. As a result, both Y_0 and Y_1 encode information pertaining to X_0 through X_3 , making each pre-shred at least partially redundant with the other. This behavior is characteristic of non-systematic encoding. EX1022 (Primer: Reed-Solomon ECC), 8 (“Nonsystematic codes add redundancy and transform the coded message such that no part of the original

message is recognizable from the un-decoded message.”). Figure 22 of Torre further illustrates this property, where the directional vector arrows associated with pre-shreds Y0 () and Y1 () both point upward—visually indicating that each contains overlapping, **redundant information**. EX1003 (Torre), FIG. 22.

311. As another example, pre-shred Y2 is also derived as a function of the same **input blocks X0, X1, X2, and X3**. Accordingly, both Y1 and Y2 encode overlapping information regarding these input blocks, meaning that each contains **data that is redundant** with respect to the other. This redundancy is characteristic of non-systematic encoding. EX1022 (Primer: Reed-Solomon ECC), 8 (“Nonsystematic codes add redundancy and transform the coded message such that no part of the original message is recognizable from the un-decoded message.”). This concept is further illustrated in Figure 22 of Torre, where the directional vector arrows associated with pre-shreds Y1 () and Y2 () both point sideways—indicating that each of these pre-shreds includes **information that is redundant to the other**. EX1003 (Torre), FIG. 22.

312. This same analysis supports the conclusion that additional pairs of pre-shreds contain **information that is redundant** with respect to one another: for example, Y2 () and Y3 () (both having directional vector arrows pointing sideways), Y3 () and Y4 () (both having directional vector arrows pointing

downward), and Y4 () and Y5 () (also pointing downward). Based on this, Torre's redundancy generator—when operating in non-systematic encoding mode—produces pre-shreds Y0 through Y4 that each include “information redundant to information included in” pre-shreds Y1 through Y5, respectively. Conversely, pre-shreds Y1 through Y5 each include “information redundant to information included in” pre-shreds Y0 through Y4, respectively.

313. **In sum**, Torre/Desai System-2 teaches “*wherein storing the redundancy information in the n shares [Torre's six shreds 474] comprises including in each of the n shares information redundant to information included in at least one other of the n shares [Torre redundancy generator, when operating in non-systematic encoding mode, produces pre-shreds that each include information redundant to information in the other pre-shreds, and the redundant information is included in the six shreds 474].*”

8. Claim 10: The method of claim 1, wherein at least some of the separate storage locations are geographically separate from one another.

314. Torre/Desai System teaches the additional limitation of Claim [10].

315. As an initial matter, I note that the '456 patent does not disclose a minimum distance for a location to be considered geographically separate. *See, e.g.*, EX1001 ('456 Patent), 3:28–32 (“According to another embodiment, the multiple depositories may be geographically remote such that a rogue employee or otherwise

compromised system at one depository will not provide access to a user's key or authentication data.”), 5:7–11 (“Another aspect of the invention includes a method of storing any type of data, including, but not limited to, authentication data in geographically remote secure data storage facilities thereby protecting the data against composition of any individual data storage facility.”), 5:51–53 (“The method also comprises storing the second pairing in a secure second data storage facility remote from the first secure data storage facility.”).

316. For example, Torre states that, in the embodiment illustrated in Figure 32, “[e]ach of the three data blocks 494 results in **six shreds 156** being stored on the **six servers 476**,” which collectively constitute the “storage pool 148.” EX1003 (Torre), [0132], [0135]. Torre further explains that the “storage pool 148 contain[s] multiple storage units 150” (for example, **servers 476**) and that “[f]or higher reliability, the storage units 150 could be *physically located separately from one another*.” EX1003 (Torre), [0068]–[0069]. See also EX1003 (Torre), FIG. 5, [0132]. For example, the storage devices may be connected by a “network” (EX1003 (Torre), [0052]), and hence “*geographically separate from one another*.” A POSITA would have recognized that “physically located separately” systems are geographically separated, as most distributed systems are; see §V.B.1 above.

317. **In sum**, Torre/Desai System teaches “*wherein at least some of the separate storage locations are geographically separate from one another* [Torre’s servers 476 are physically located separately from one another].”

9. Claim 11: The method of claim 1, wherein logically combining comprises logically combining using an exclusive-OR operation.

318. Torre/Desai System teaches the additional limitation of Claim 11.

319. As discussed in §IX.A.5.e above [1B-2], Desai’s $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm includes **step E4**, in which the algorithm performs an exclusive-OR operation (denoted by the symbol “ $\oplus$ ”) between the key K' and all of the encrypted data blocks $x'[1] \dots x'[n]$ to generate the resultant $x'[n+1]$. This XOR operation in **step E4** teaches the claimed “*logically combining*” limitation; it therefore discloses the limitation “*logically combining using an exclusive-OR operation.*” This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{E}\text{-CTR}^{F,F^{-1}}(x[1] \dots x[n])$
Step E1	$K' \leftarrow \{0, 1\}^l$
Step E2	$K = K' \bmod 2^k \quad // (K = k)$
Step E3	for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$
Step E4	$x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$
Step E5	return $x'[1] \dots x'[n+1]$

EX1004 (Desai), 16 (annotated).

320. **In sum**, Torre/Desai System teaches “*wherein logically combining comprises logically combining using an exclusive-OR operation* [Desai’s CTRT transformation algorithm performs an exclusive-OR operation].”

10. Claim 12: The method of claim 1, wherein the at least a portion of the encrypted data set comprises all of the encrypted data set.

321. Torre/Desai System teaches the additional limitation of Claim 12.

322. As explained in §IX.A.5.e above [1B-2], $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm includes **step E4**, in which the algorithm performs an XOR operation between all of the **encrypted data blocks $x'[1] \dots x'[n]$** (the “*encrypted data set*”) and the **encryption key K'** to compute **$x'[n+1]$** (the “*resultant*”). Accordingly, Desai teaches logically combining “*all of the encrypted data set*” **$x'[1] \dots x'[n]$** . This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.A.1 above.

	Algorithm $\mathcal{E}\text{-CTR}^{F,F^{-1}}(x[1] \dots x[n])$
Step E1	$K' \leftarrow \{0, 1\}^l$
Step E2	$K = K' \bmod 2^k \quad // (K = k)$
Step E3	for $i = 1, \dots, n$ do $x'[i] = x[i] \oplus F_K(i)$
Step E4	$x'[n+1] = K' \oplus x'[1] \oplus \dots \oplus x'[n]$
Step E5	return $x'[1] \dots x'[n+1]$

EX1004 (Desai), 16 (annotated).

323. **In sum**, Torre/Desai System teaches “*wherein the at least a portion of the encrypted data set comprises all of the encrypted data set* [Desai’s CTRT transformation algorithm performs an exclusive-OR operation between all of the encrypted data blocks $x'[1] \dots x'[n]$ and the encryption key K' to compute $x'[n+1]$].”

11. Claim 13

324. Claim 13 has substantially the same limitations as Claim 1 except for the preamble [13Pre], as seen in the comparison table below.

Claim 1	Claim 13
[1B] processing the first data set to produce a second data set, the processing comprising	[13A] produce a second data set from the first data set by
[1C] producing redundancy information based on information in the second data set	[13B] produce redundancy information based on information in the second data set
[1D] producing a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set	[13C] produce a plurality of n shares from the second data set, wherein each of the n shares comprises at least some of the second data set
[1E] storing the plurality of n shares and the redundancy information, wherein at least some of the n shares are stored in separate storage locations and the first data set is recoverable using at least a threshold number, less than n, of the plurality of n shares	[13D] store the plurality of n shares and the redundancy information in a plurality of storage locations, wherein at least some of the n shares are stored in separate storage locations and the first data set is recoverable using at least a threshold number, less than n, of the plurality of n shares

325. Therefore, for the same reasons discussed above in §§IX.A.5.c above—IX.A.5.i above regarding limitations [1B]–[1E], the Torre/Desai System also satisfies claim limitations [13A]–[13D], respectively.

- a. [13Pre] A computer system for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the system comprising one or more processors configured to:

326. To the extent the preamble is limiting, Torre/Desai System teaches preamble [13Pre].

327. For example, Torre discloses “securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations” for the same reasons I provided above in §IX.A.5.a above [1Pre].

328. Torre also discloses “[a] computer system ... comprising one or more processors” that is configured to perform the described method. For example, Torre states that “the embodiments disclosed herein, in whole or in part, can be equivalently implemented ... as one or more computer programs running on one or more computers (e.g., as one or more programs running on one or more data processing systems),” which corresponds to a “computer system.” EX1003 (Torre), [0141]. Torre further clarifies that such programs may “run[] on one or more processors e.g., microprocessors,” thereby satisfying the requirement of a system

“comprising one or more processors.” EX1003 (Torre), [0141], [0142], [0064]. See also §V.A above.

329. **In sum**, Torre/Desai System teaches “[a] computer system [Torre’s computer programs running on one or more computers] *for securing a first data set* [Torre’s method for shredding and deshredding **input data** for secure storage that runs on one or more computers] *while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations* [Torre’s method for deshredding **input data** to recover **the input data** using only a subset of the stored **shreds** if some of the **servers** are nonfunctional].”

12. Claim 25

330. Claim 25 is identical to claim 1 except for claim preamble [25Pre].

331. In the table below, the limitations in the same row are identical to one another.

Claim 1	Claim 25
[1A]	[25A]
[1B]	[25B]
[1B-1]	[25B-1]
[1B-2]	[25B-2]
[1B-3]	[25B-3]
[1C]	[25C]
[1D]	[25D]
[1E]	[25E]

332. Therefore, for the same reasons discussed above in §§IX.A.5.b above—IX.A.5.i above regarding limitations [1A]–[1E], the Torre/Desai System also satisfies claim limitations [25A]–[25E], respectively.

- a. **[25Pre] A non-transitory computer-readable medium comprising instructions that, when executed by one or more processors, cause a computer system to carry out a method for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations, the method comprising:**

333. To the extent the preamble is limiting, the Torre/Desai System teaches preamble [25Pre].

334. Torre teaches “*a method for securing a first data set while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations*” for the reasons I provided above in §IX.A.5.a above [1Pre].

335. Torre further discloses that the method may be implemented using “[*a non-transitory computer-readable medium comprising instructions that, when executed by one or more processors, cause a computer system to carry out*” the *method*. Specifically, Torre explains that its teachings “can be implemented, individually and/or collectively, by a wide range of hardware, **software**, firmware, or virtually any combination thereof.” EX1003 (Torre), [0141]. The software implementation is described as “**code**” (*i.e.*, “instructions”), which, “when executed by one or more processors, cause[s] a computer system” to perform the described

operations. *See* EX1003 (Torre), [0141] (noting that implementation may occur “as one or more **computer programs** running on **one or more computers** ..., as one or more **programs** running on **one or more processors e.g., microprocessors**, as firmware, or as virtually any combination thereof”); *see also* EX1003 (Torre), [0142], [0064].

336. In addition, Torre teaches that this software code may be stored in “signal bearing media,” including “floppy disks, hard disk drives, CD ROMs, digital tape, and computer memory”—each of which constitutes a “*non-transitory computer-readable medium*.” EX1003 (Torre), [0141], [0064]. *See also* §V.A above.

337. **In sum**, Torre/Desai System teaches “[a] *non-transitory computer-readable medium* [Torre’s computer programs running on computers, processors, or microprocessors] *comprising instructions* [Torre’s software code] *that, when executed by one or more processors* [e.g., Torre’s microprocessors], *cause a computer system to carry out a method for securing a first data set* [Torre’s method for shredding and deshredding **input data** for secure storage] *while providing recoverability of the first data set in the event of unavailability of data at one or more storage locations* [Torre’s method for deshredding **input data** to recover **the input data** using only a subset of the stored **shreds** if some of the **servers** are nonfunctional].”

13. Claims 14, 15, 22-24, 26, 27, and 30

338. The Torre/Desai System renders claims 14, 15, 22-24, 26, 27, and 30 obvious.

- Torre/Desai System-2 satisfies the additional limitations of claims 14 and 26 for the same reasons discussed in §IX.A.6 above [Claim 2];
- Torre/Desai System-2 satisfies the additional limitations of claims 15 and 27 for the same reasons discussed in §IX.A.7 above [Claim 3];
- Torre/Desai System-2 satisfies the additional limitations of claim 22 for the same reasons discussed in §IX.A.8 above [Claim 10];
- Torre/Desai System-2 satisfied the additional limitations of claim 23 for the same reasons discussed in §IX.A.9 above [Claim 11]; and
- Torre/Desai System-2 satisfied the additional limitations of claims 24 and 30 for the same reasons discussed in §320 above [Claim 12].

B. Ground II: Torre, Desai, and Watanabe Render Obvious Claims 1-3, 10-12, 25-27, And 30.

1. Torre/Desai/Watanabe Color-Coding Scheme

339. The illustrations in this ground have been color-coded in accordance with the color-coding scheme summarized in the table below.

Color	Disclosures in Torre, Desai, and/or Watanabe
red	Torre's shreds
brown	Torre's Reed-Solomon encoding
orange	Torre's redundancy information
olive	Torre's storage units
lime green	Torre's input data
royal blue	Torre's pass-through data blocks
violet	Desai's encrypted data blocks $x[1] \dots x[n]$

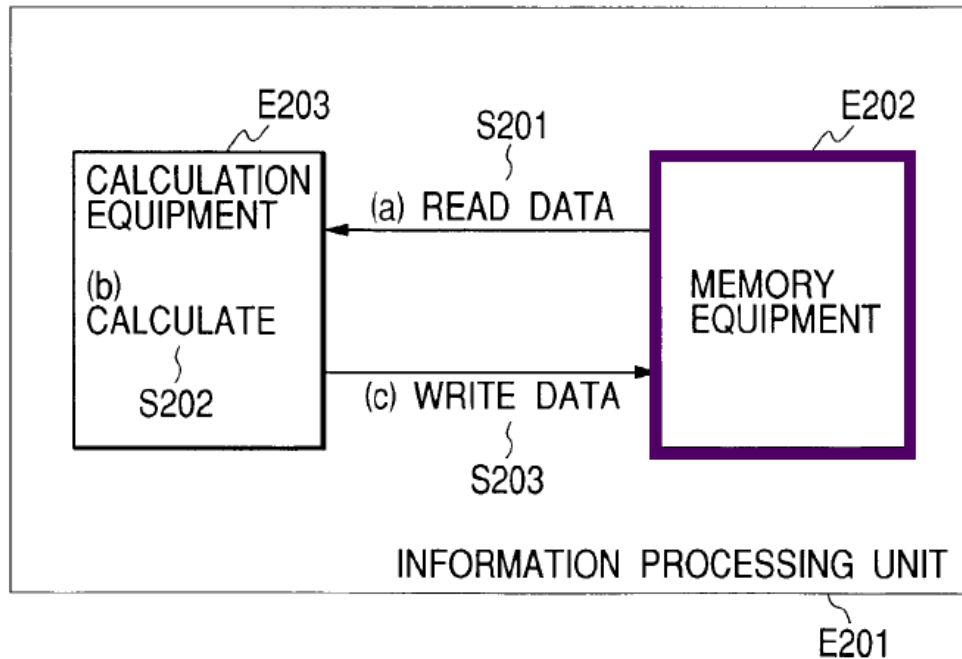
Color	Disclosures in Torre, Desai, and/or Watanabe
pink	Desai's encrypted data block $x[n+1]$
teal	Desai's encrypted data blocks $x[1] \dots x[n+1]$
dark blue	Desai's encryption key
light green	Desai's CTRT process
purple	Watanabe's memory

2. Overview of Watanabe (EX1005)

340. Watanabe pertains to “a method of encrypting and decrypting data used within an information processing unit.” EX1005 (Watanabe), [0003]. The reference discloses an “information processing unit” designed to encrypt the data exchanged between a memory device and computational hardware. This encryption mechanism serves to “break the correlation between data to be processed and current consumption.” EX1005 (Watanabe), [0025].

341. Watanabe's “First Exemplary Embodiment” specifically “relates to the encryption of data to be processed.” EX1005 (Watanabe), [0049]. “As shown in FIG. 2, data processing by an information processing unit E201 basically comprises: (a) *the reading of data from a memory device E202* (step S201); (b) calculation by calculation equipment E203 (step S202); and (c) the writing of data into *the memory device E202* (step S203).” EX1005 (Watanabe), [0049]. This is shown in Watanabe's Figure 2 below, annotated to use the same color-coding scheme as that described in the table of §IX.B.1 above.

FIG. 2



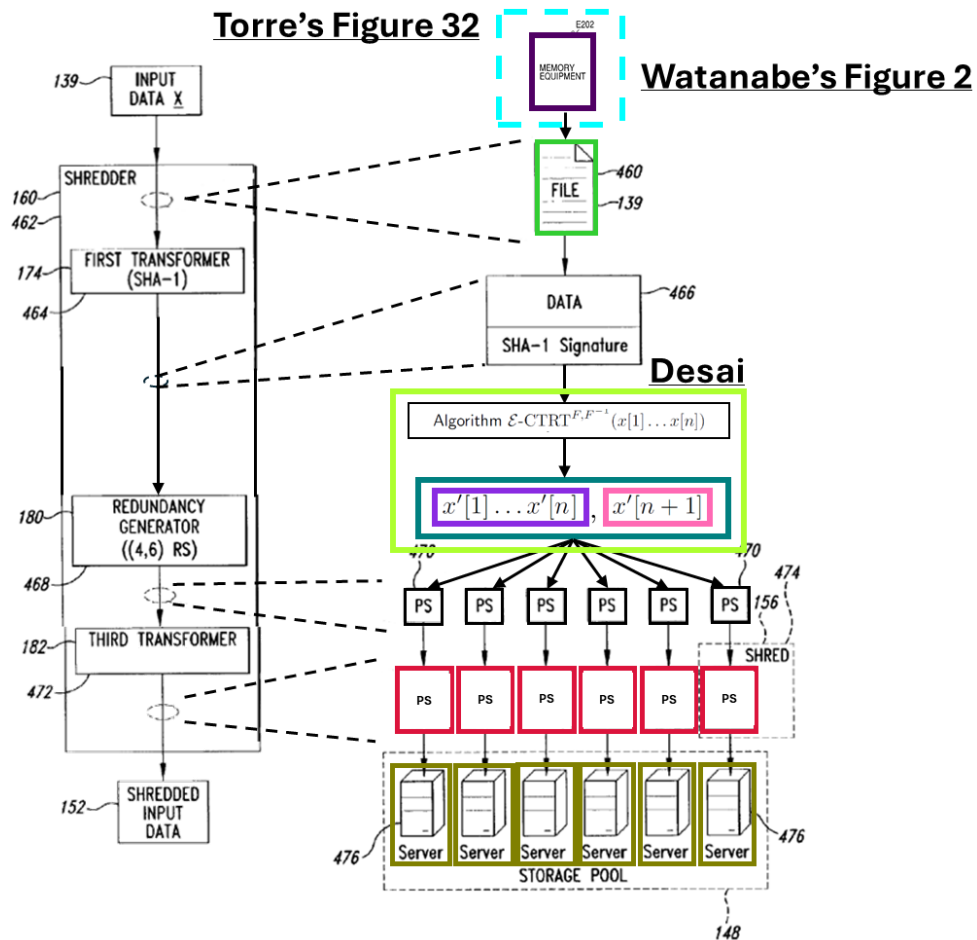
EX1005 (Watanabe), FIG. 2.

3. Combining Torre/Desai With Watanabe ("Torre/Desai/Watanabe System")

342. The Torre/Desai System is directed to encrypting **the input data**—using an all-or-nothing transform (AONT)—followed by splitting that **input data** into **shreds** and storing the **shreds** across distributed storage. It would have been obvious to a person of ordinary skill in the art to augment the Torre/Desai System with the teachings of Watanabe so that the system retrieves **the input data** to be processed from **memory**, as described in Watanabe's First Exemplary Embodiment. This integration would allow the Torre/Desai System to adopt Watanabe's approach of **memory**-based data retrieval prior to encryption and processing, as illustrated in the

annotated figure below. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.B.1 above.

Torre/Desai/Watanabe System (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1005 (Watanabe), FIG. 2 (annotated);
EX1004 (Desai), 16.

a. Reasons That A POSITA Would Have Been Motivated To Combine Torre/Desai With Watanabe To Arrive At The Torre/Desai/Watanabe System

343. A person of ordinary skill in the art would have been motivated to incorporate Watanabe's memory device into the Torre/Desai System so that the system receives **input data** from a **memory device**, as taught by Watanabe. There are at least two reasons supporting this motivation.

344. *First*, Watanabe teaches that an encryption system—such as its calculation equipment E203—can retrieve data for processing directly from a **memory device E202**. EX1005 (Watanabe), [0049]. Watanabe specifically notes that “various types of data items are read from **the memory device E202** depending on the purpose or status of the unit.” EX1005 (Watanabe), [0049]. Based on this disclosure, a person of ordinary skill in the art would have understood that a **memory device** like Watanabe's is capable of maintaining and supplying “**input data**” for systems like Torre. *See* EX1003 (Torre), [0050] (“The **input data** can be a file or another form that has been generated directly or by other means. Generation of data includes, but is not limited to, generation by operating software programs and by receiving or recording data such as live audio or video, or audio or video broadcast streams.”).

345. Therefore, a POSITA would have been motivated to incorporate Watanabe's **memory device** into the Torre/Desai System so that the system can

flexibly receive **input data** from a variety of sources—including from **memory**. To the extent the Torre/Desai System does not already explicitly contemplate retrieval of data from **a memory device** (as discussed in §IX.A.5.b above [Ground I, 1A]), this integration would enhance its versatility and adaptability to real-world deployment scenarios.

346. **Second**, combining Watanabe with the Torre/Desai System would have merely involved applying a known technique—namely, using **a memory device** to store and supply data to a data processing system, as disclosed by Watanabe—to improve a similar system. Specifically, incorporating Watanabe’s **memory device** into the Torre/Desai System would enhance its ability to handle **input data** from a variety of sources, thereby increasing its versatility. This use aligns with the same purpose taught by Watanabe: to supply data for encryption or further processing. Thus, a POSITA would have recognized that using Watanabe’s approach to provide data as input to the Torre/Desai System is a straightforward and predictable application of known principles to achieve the same functional outcome in a closely related context.

347. A person of ordinary skill in the art would have had a reasonable expectation of success in integrating Watanabe’s memory device into the Torre/Desai System for at least the following two reasons.

348. **First**, a person of ordinary skill in the art would have further understood that data files—including for example **Torre’s input data file** (as discussed in §IX.A.2 above [Torre]) and Desai’s message/data blocks (as discussed in §IX.A.3 above [Desai])—can be stored on and read from data storage memory structures such as Watanabe’s **memory device**.

349. Torre discloses that “[a]fter receiving the input data (step 122),” its shredder “performs first transformations (step 132),” and that “[t]he first transformations could include one or more of compression, encryption, or signature generation functions.” EX1003 (Torre), [0061]. Watanabe similarly discloses that the “embodiment relate[d] to the encryption of data to be processed” involves “the reading of data from **a memory device**” and subsequently the “calculation by calculation equipment.” EX1005 (Watanabe), [0049]. Based on these teachings, a person of ordinary skill in the art would have understood that Watanabe’s **memory device** is readily compatible with the architecture and data flow of the Torre/Desai(/Watanabe) System, and could be integrated without disrupting its core shredding/deshredding functionality.

350. **Second**, the proposed combination would have merely involved applying a known technique—namely Watanabe’s **memory device**—to enhance the versatility of a similar system, the Torre/Desai System, in the same manner taught

by Watanabe. Specifically, the memory device improves versatility by allowing the system to store, receive, and process a variety of **input data** from various sources.

4. Claim 1

351. For the same reasons I discussed for Ground I, the Torre/Desai/Watanabe System teaches preamble [1Pre] and [1B]–[1E]. *See* §IX.A.5 above.

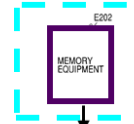
a. [1A] reading the first data set from **a memory that stores the first data set**;

352. To the extent Patent Owner contends that the combination of Torre and Desai fails to disclose limitation [1A], that limitation would nevertheless have been obvious in view of Watanabe.

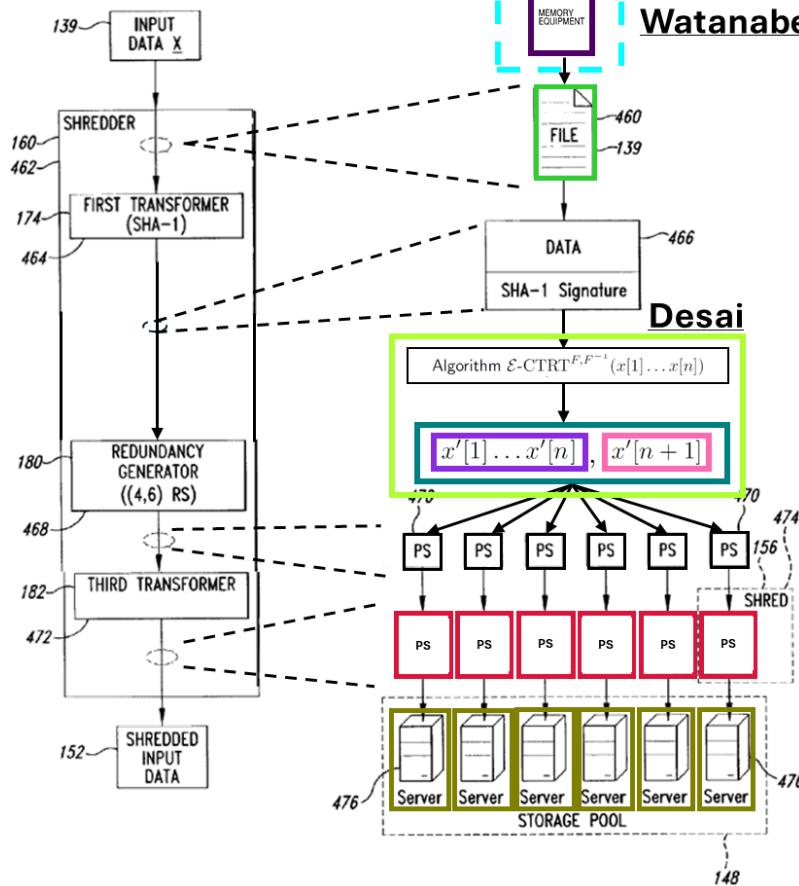
353. Specifically, Watanabe teaches that “data processing by an information processing unit E201 basically comprises: (a) ***the reading of data from a memory device E202*** (step S201)” EX1005 (Watanabe), [0049]. As discussed in §IX.B.3 above [Torre/Desai/Watanabe], it would have been obvious to a person of ordinary skill in the art to combine the Torre/Desai System with Watanabe so that the Torre/Desai System is configured to “read[] ... [input data] from a memory device,” as expressly taught by Watanabe. EX1005 (Watanabe), [0049]. This combination is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.B.1 above.

Torre/Desai/Watanabe System (Shredder)

Torre's Figure 32



Watanabe's Figure 2



EX1003 (Torre), FIG. 32 (annotated); EX1005 (Watanabe), FIG. 2 (annotated);
EX1004 (Desai), 16.

354. **In sum**, Torre/Desai/Watanabe System teaches “reading the first data set [Torre’s **input data 139**] from a memory that stores the first data set [Watanabe’s **memory device E202**].”

5. Claim 25

355. For the same reasons I discussed above in Ground I, the Torre/Desai/Watanabe System teaches preamble [25Pre] and [25B]–[25E]. *See* §IX.A.12 above [Ground I, Claim 25]. For the same reasons discussed in §IX.B.4.a above regarding claim limitation [1A], the Torre/Desai/Watanabe System also satisfies claim limitation [25A].

6. Claims 2, 3, 10–12, 26, 27, and 30

356. The Torre/Desai/Watanabe System renders obvious claims 2, 3, 10–12, 26, 27, and 30.

- The Torre/Desai/Watanabe System satisfies the additional limitations of claims 2 and 26 for the same reasons discussed in §IX.A.6 above [Ground I, Claim 2] and §IX.A.13 above [Ground I, Claim 26];
- The Torre/Desai/Watanabe System satisfies the additional limitations of claims 3 and 27 for the same reasons discussed in §IX.A.7 above [Ground I, Claim 3] and §IX.A.13 above [Ground I, Claim 27];
- The Torre/Desai/Watanabe System satisfies the additional limitations of claim 10 for the same reasons discussed in §IX.A.8 above [Ground I, Claim 10];
- The Torre/Desai/Watanabe System satisfies the additional limitations of claim 11 for the same reasons discussed in §IX.A.9 above [Ground I, Claim 11]; and

- The Torre/Desai/Watanabe System satisfies the additional limitations of claims 12 and 30 for the same reasons discussed in §320 above [Ground I, Claim 12] and §IX.A.13 above [Ground I, Claim 30].

C. Ground III: Ground I Or Ground II In Combination With Orsini Renders Obvious Claims 4-9, 16-21, and 28-29.

1. Torre/Desai/(Watanabe)/Orsini Color-Coding Scheme

357. The illustrations in this ground have been color-coded in accordance with the color-coding scheme summarized in the table below.

Color	Disclosures in Torre, Desai, Watanabe, and/or Orsini
red	Torre's shreds / Orsini's shares
brown	Torre's Reed-Solomon encoding
orange	Torre's redundancy information
olive	Torre's storage units
lime green	Torre's input data
royal blue	Torre's pass-through data blocks
violet	Desai's encrypted data blocks $x'[1] \dots x'[n]$
pink	Desai's encrypted data block $x'[n+1]$
teal	Desai's encrypted data blocks $x'[1] \dots x'[n+1]$
dark blue	Desai's encryption key
light green	Desai's CTRT process
purple	Watanabe's memory
azure blue	Orsini's cryptosplitting/data shuffling process

2. Overview of Orsini (EX1006)

358. Orsini contains disclosures that are substantially identical to those found in the '456 Patent. Specifically, Figures 1 through 25 of Orsini—and the corresponding written descriptions—are materially the same as Figures 1 through 25 and their associated disclosures in the '456 Patent. This equivalence is

demonstrated in EX1021, which provides a redline comparison showing the similarities between Orsini and the '456 Patent specification.

359. Orsini discloses that “[i]n one embodiment of the present invention, [a] data splitting module comprises a parser program or software suite which comprises **data splitting**.” EX1006 (Orsini), [0278]. According to Orsini, this data splitting module carries out a process referred to as a “**cryptographic split**.” EX1006 (Orsini), [0281]. A “**cryptographic split (cryptosplit)**” is described as a method that “partitions [] data into N number of shares.” EX1006 (Orsini), [0281]. As part of this partitioning, Orsini explains that “[t]he units are [] distributed (either randomly or by a predetermined set of values) into the N shares,” and notes that this process “could also involve a shuffling of the order of the units in the shares.”

360. Orsini further explains that the parsing process may involve “splitting the data according to the contents of a generated random number, or key.” EX1006 (Orsini), [0280]. In one illustrative example, Orsini describes a **cryptosplit and data shuffling operation** where the data is “23 bytes in size,” with each “data unit” measuring “one byte.” EX1006 (Orsini), [0282]. In this scenario, “a key would be obtained to create a sequence of 23 random numbers (r1, r2, r3 through r23),” and “each of the units of data (in this example 23 individual bytes of data) is associated with one of the 23 random numbers corresponding to one of the four shares.” Each

unit of data is then distributed into the respective shares based on its associated random number. EX1006 (Orsini), [0284].

361. Orsini further discloses that “the data units in **each share** could be shuffled utilizing a different algorithm.” EX1006 (Orsini), [0286]. This shuffling of data units may occur at various stages in the process: “as the original data is split into the data units, or after the data units are placed into **the shares**, or after **the share** is full, for example.” EX1006 (Orsini), [0286].

3. Combining Grounds I or II With Orsini

362. The Torre/Desai System discussed in Ground I and the Torre/Desai/Watanabe System discussed in Ground II are both directed to securely storing input data by parsing and splitting that data into **shreds**, which are then stored in separate storage locations. *See* §§IX.A.4 above [Torre/Desai], IX.B.3 above [Torre/Desai/Watanabe].

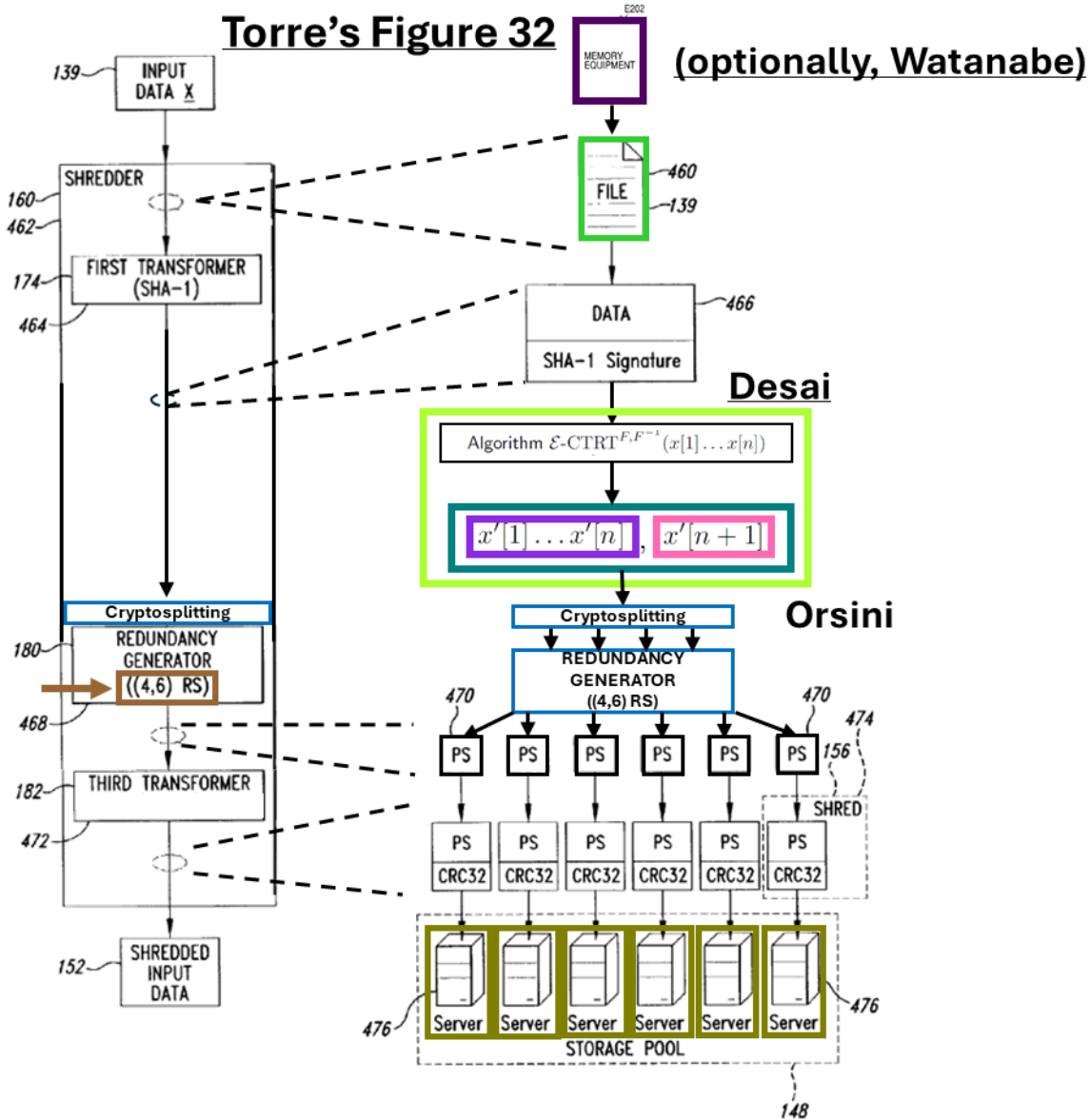
363. A person of ordinary skill in the art would have found it obvious to enhance either system—whether the Torre/Desai System in Ground I or the Torre/Desai/Watanabe System in Ground II—with Orsini’s **cryptosplit or data shuffling technique**. Incorporating Orsini’s approach would increase the security of the stored data by adding randomized distribution or reordering of data units during or after the shredding process. The resulting combination—collectively referred to as the Torre/Desai(/Watanabe) System—is described in further detail below.

Watanabe has a parenthetical around it to denote that Watanabe is relevant only to Ground II.

a. Combining Torre/Desai(/Watanabe) System With Orsini's Cryptosplit Process ("Torre/Desai (/Watanabe)/Orsini System-1")

364. It would have been obvious to a person of ordinary skill in the art to augment the Torre/Desai(/Watanabe) System with Orsini so that the output of Desai's CTRT process is further processed using Orsini's [cryptosplit technique](#). Specifically, the output of Desai's $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm— $x' = x'[1] \dots x'[n+1]$ —would be subject to Orsini's [cryptosplitting process](#), which partitions and optionally shuffles the data units to enhance security. EX1006 (Orsini), [0281] ("A cryptographic split (cryptosplit) partitions the data into N number of shares."), [0286] ("It is readily apparent to those of ordinary skill in the art that there are many ways of performing redundant cryptosplitting and non-redundant cryptosplitting For example, the data units in each share could be shuffled utilizing a different algorithm."). *See also* §IX.C.2 above. This would result in four pre-shreds, which could then be supplied as inputs to the "4 input–6 output [Reed Solomon redundancy generation function](#)," as disclosed in Torre. EX1003 (Torre), [0131]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.C.1 above.

Torre/Desai/(Watanabe)/Orsini System-1 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1005 (Watanabe), FIG. 2 (annotated);
EX1004 (Desai), 16; EX1006 (Orsini), [0281], [0286].

i. **Reasons Why A POSITA Would Have Combined Torre, Desai, Orsini (and, optionally, Watanabe) As Provided In Torre/Desai (/Watanabe)/Orsini System-1**

365. A person of ordinary skill in the art would have been motivated to combine the Torre/Desai(/Watanabe) System with Orsini's cryptosplit functionality, resulting in what is referred to herein as the Torre/Desai(/Watanabe)/Orsini System-1. There are at least two reasons supporting this motivation.

366. *First*, Torre's "4 input-6 output **Reed Solomon redundancy** generation function" is designed to accept four input segments and produce six pre-shreds as output. *See* EX1003 (Torre), FIGS. 21–22, [0131]. In view of this, a person of ordinary skill in the art would have been motivated to consult prior art references to identify suitable techniques for formatting the output of Desai's $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm into four distinct segments. Orsini's **cryptosplit process** offers such functionality. Orsini teaches that its data splitting module can serve as "an add-on to any application which requires *the ultimate security* of its data elements," and that its "parsing process operates *on any type of data set*, and on any and all file types, or in a database on any row, column or cell of data in that database." EX1006 (Orsini), [0278]; *see also* EX1006 (Orsini), [0115].

367. Based on these teachings, a POSITA would have understood that Orsini's **cryptosplit process** provides a secure and flexible method for segmenting

the output of Desai's encryption process into four segments suitable for Torre's redundancy engine. Doing so would yield the predictable and desirable result of enhancing the security of the shredding process.

368. *Second*, a person of ordinary skill in the art would have been motivated to apply a known technique—namely, Orsini's cryptosplit process—to enhance the security of a known system, such as the Torre/Desai(/Watanabe) System. This enhancement would be achieved in the same manner that Orsini describes: by randomly shuffling and splitting the output of Desai's $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm prior to providing it as input to the “4 input–6 output **Reed Solomon**” **redundancy** generator 180. Orsini's cryptosplit technique adds an additional layer of security by increasing the complexity and obfuscation of the resulting shreds. As Orsini explains, the cryptosplit module may be used “as an add-on to any application which requires the ultimate security of its data elements.” EX1006 (Orsini), [0278].

369. A POSITA would have recognized that integrating this process into the Torre/Desai(/Watanabe) framework would make it more difficult for a third-party attacker to reconstruct the original input data from the shreds. Even if the attacker obtained the resulting shreds from the Torre/Desai(/Watanabe)/Orsini System-1, they would still need to reverse the randomized splitting and shuffling introduced by Orsini's cryptosplit process.

370. A person of ordinary skill in the art would have had a reasonable expectation of success in combining the Torre/Desai(/Watanabe) System with Orsini's cryptosplit functionality, as implemented in the Torre/Desai(/Watanabe)/Orsini System-1. This expectation is supported by at least the following four reasons.

371. **First**, Orsini expressly teaches that its cryptosplit process may be used “as an add-on to *any* application” and that it “operates on *any type of data set*, and on any and all file types, or in a database on any row, column or cell of data in that database.” EX1006 (Orsini), [0278]. Based on this disclosure, a person of ordinary skill in the art would have readily understood that Orsini's cryptosplit process is designed for broad compatibility and modular integration. Accordingly, a POSITA would have recognized that Orsini's technique could be effectively applied to the Torre/Desai(/Watanabe) System—particularly to the output of Desai's $E\text{-CTR}^{F,F^{-1}}$ transformation algorithm—without requiring significant adaptation.

372. **Second**, in the Torre/Desai(/Watanabe)/Orsini System, the output of Orsini's cryptosplit process—comprising four data segments—would be provided directly to the “4 input–6 output Reed Solomon” redundancy generator 180, which is configured to receive four segments of data as input. EX1003 (Torre), [0131]. A person of ordinary skill in the art would have understood that each of the four segments produced by the cryptosplit process could be supplied to the redundancy

generator without requiring additional data transformation. As illustrated in FIG. 21 (systematic encoding) and FIG. 22 (non-systematic encoding), the redundancy generator is capable of directly processing those segments. Accordingly, the integration of Orsini's cryptosplit functionality into the Torre/Desai(/Watanabe) System would not have required any undue experimentation. The connection between the cryptosplit output and the redundancy generator input would have been straightforward and would have predictably resulted in the enhanced security of the overall shredding process in the Torre/Desai(/Watanabe) System.

373. **Third**, the proposed combination would have merely involved applying a known technique—Orsini's cryptosplit process—to enhance the security of a similar system, namely the Torre/Desai(/Watanabe) System, in the same manner taught by Orsini. The cryptosplit process increases the randomness and obfuscation of the input fed into the redundancy generator. As a result, the shreds produced by the redundancy generator—and subsequently by the third transformer—also undergo increased randomness.

374. **Fourth**, Torre states that “the embodiments disclosed herein ... can be equivalently implemented ... as one or more *computer programs* running on one or more computers.” EX1003 (Torre), [0141]. Accordingly, implementing Orsini's cryptosplit process within the Torre/Desai(/Watanabe) System would have simply involved writing or integrating software code. As Torre explains, “designing the

circuitry and/or writing the code for the software and or firmware would be well within the skill of one of ordinary skill in the art in light of this disclosure.” EX1003 (Torre), [0141]. Accordingly, this level of software implementation was well within the ordinary skill of a POSITA at the time of the invention.

b. Combining Torre/Desai(/Watanabe) System With Orsini’s Data Shuffling Process (“Torre/Desai (/Watanabe)/Orsini System-2”)

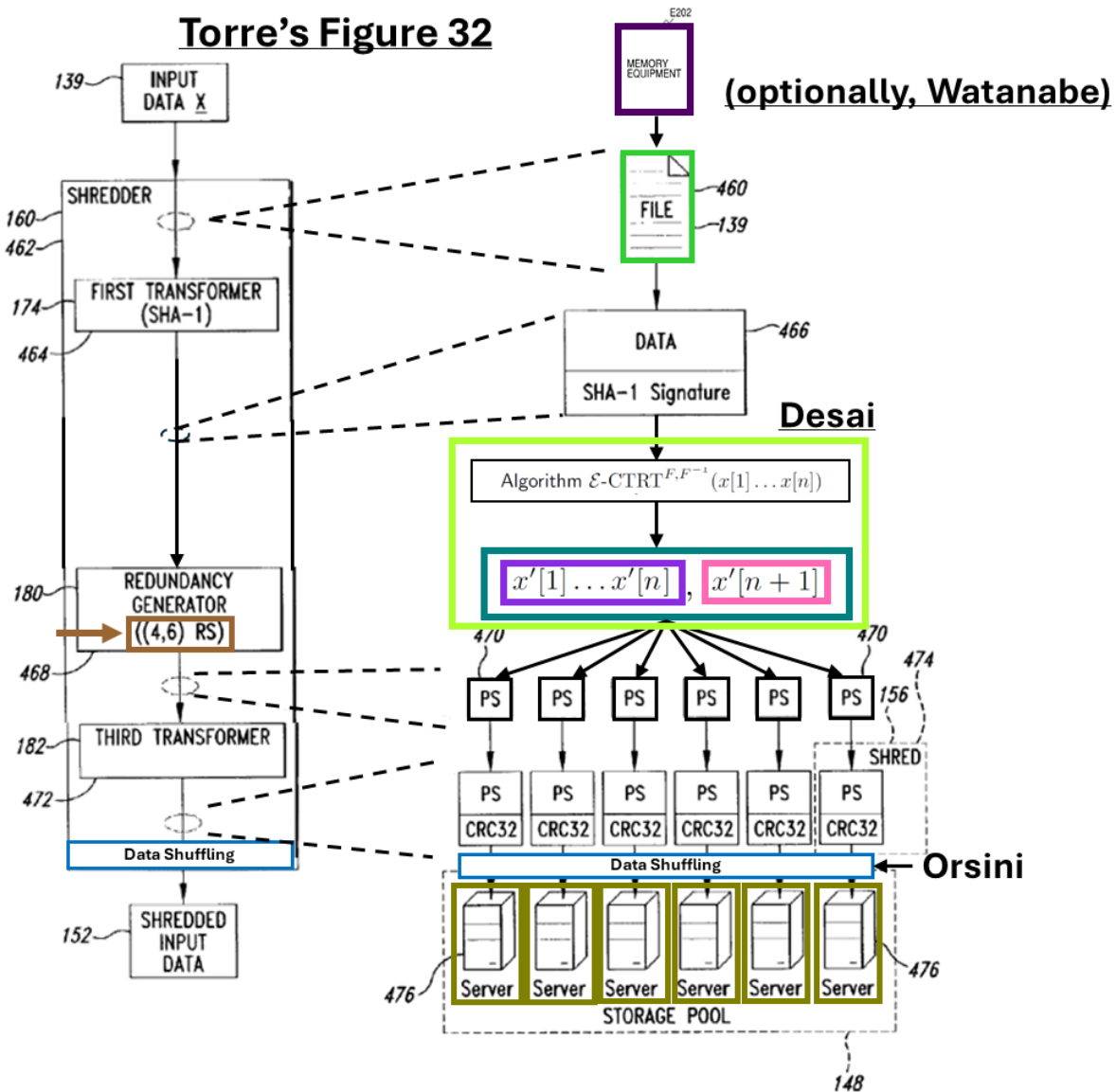
375. It would also have been obvious to a person of ordinary skill in the art to combine the Torre/Desai(/Watanabe) System with Orsini’s teachings by incorporating Orsini’s data shuffling process *after* the third transformer generates the shreds. Orsini explicitly states that “the data units in each share could be shuffled utilizing a different algorithm,” and further explains that this “data unit shuffling may be performed ... *after* the data units are placed into the shares.” EX1006 (Orsini), [0286].

376. Accordingly, a POSITA would have recognized that applying Orsini’s shuffling algorithm to the shreds output, by the third transformer in the Torre/Desai(/Watanabe) System, would enhance the randomness and obfuscation of the shreds after the they are generated by Torre/Desai(/Watanabe) System’s third transformer. This combination is referred to herein as “Torre/Desai(/Watanabe)/Orsini System-2” and is illustrated in the figure below,

annotated to use the same color-coding scheme as that described in the table of §IX.C.1 above.

Torre/Desai(/Watanabe)/Orsini System-2 (Shredder)

Torre's Figure 32



EX1003 (Torre), FIG. 32 (annotated); EX1005 (Watanabe), FIG. 2 (annotated);
EX1004 (Desai), 16; EX1006 (Orsini), [0281], [0286].

i. Reasons that a POSITA Would Have Combined Torre, Desai, Orsini (and, optionally, Watanabe) As Provided In Torre/Desai (/Watanabe)/Orsini System-2

377. A person of ordinary skill in the art would have been motivated to combine the Torre/Desai(/Watanabe) System with Orsini as implemented in Torre/Desai(/Watanabe)/Orsini System-2. This motivation is supported by at least the following two reasons.

378. **First**, Orsini teaches that its data splitting module—which includes both the cryptosplit and data shuffling processes—can be used “as an add-on to *any* application which requires *the ultimate security* of its data elements.” EX1006 (Orsini), [0278]; *see also* EX1006 (Orsini), [0115]. Based on this disclosure, a person of ordinary skill in the art would have been motivated to incorporate Orsini’s data shuffling functionality into the Torre/Desai(/Watanabe) System in order to further enhance the security of the resulting shreds.

379. **Second**, a person of ordinary skill in the art would have been motivated to apply a known technique—Orsini’s data shuffling process—to improve the security of a known system, such as the Torre/Desai(/Watanabe) System, in the same manner taught by Orsini. Specifically, Orsini’s process involves randomly shuffling data units within each share, and applying that step to the shreds produced by the third transformer would have introduced additional obfuscation into the final output.

As Orsini itself explains, its data shuffling mechanism is intended for “any application which requires the ultimate security of its data elements.” EX1006 (Orsini), [0278]. Accordingly, a POSITA would have recognized that adding Orsini’s shuffling process after the generation of shreds would make it more difficult for a third-party attacker to reconstruct the original input data, as they would need to reverse the effects of the added data shuffling.

380. A person of ordinary skill in the art would have had a reasonable expectation of success in combining the Torre/Desai(/Watanabe) System with Orsini as implemented in Torre/Desai(/Watanabe)/Orsini System-2. This expectation would have been supported by at least the following three reasons.

381. *First*, Orsini teaches that its data splitting module—which includes the data shuffling process—can be applied “as an add-on to *any* application” and that it “operates on *any type of data set*, and on any and all file types, or in a database on any row, column or cell of data in that database.” EX1006 (Orsini), [0278].

382. Orsini further specifies that “data unit shuffling may be performed ... *after* the data units are placed into the shares,” which directly corresponds to the timing of shred generation in the Torre/Desai(/Watanabe) System. EX1006 (Orsini), [0286]. Based on these teachings, a person of ordinary skill in the art would have understood that Orsini’s data shuffling process is readily compatible with the

architecture and data flow of the Torre/Desai(/Watanabe) System, and could be integrated without disrupting its core functionality.

383. **Second**, the proposed combination would have merely involved applying a known technique—namely, Orsini’s data shuffling process—to enhance the security of a similar system, the Torre/Desai(/Watanabe) System, in the same manner taught by Orsini. Specifically, the shuffling process increases the randomness and obfuscation of the data units within each shred, which are generated by the Torre/Desai(/Watanabe) System’s third transformer.

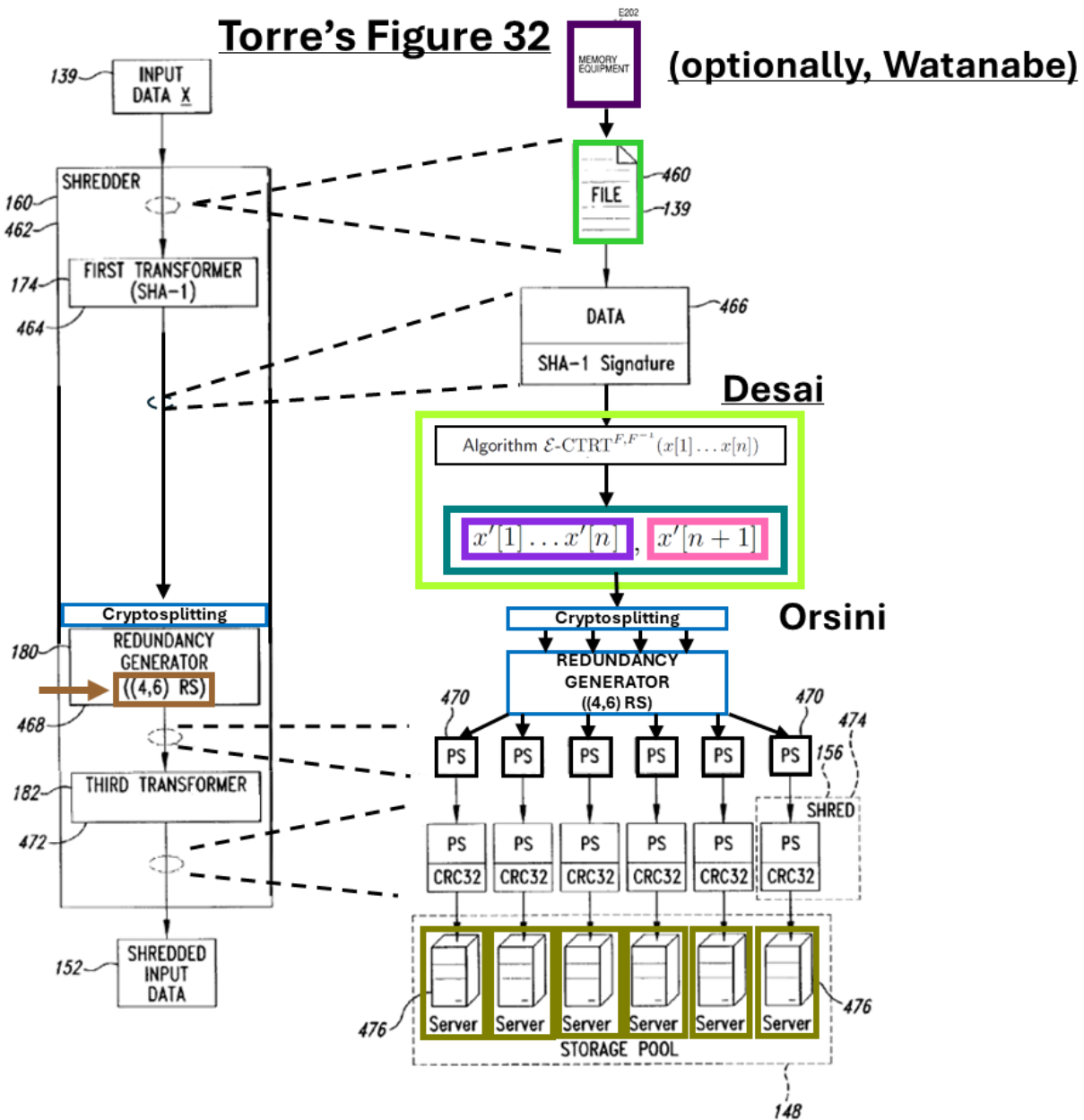
384. **Third**, Torre states that “the embodiments disclosed herein ... can be equivalently implemented ... as one or more *computer programs* running on one or more computers.” EX1003 (Torre), [0141]. Therefore, implementing Orsini’s data shuffling functionality within the Torre/Desai(/Watanabe) System would have simply required the development or integration of program code—an implementation step that falls squarely within the routine skill set of a person of ordinary skill in the art. *See* EX1003 (Torre), [0141].

4. **Claim 4:** The method of claim 1, wherein producing **the plurality of n shares** comprises causing **the n shares** to comprise a substantially random distribution of units of data of **the second data set**.
5. **Claim 5:** The method of claim 4, wherein causing **the n shares** to comprise a substantially random distribution comprises using a key.
6. **Claim 6:** The method of claim 5, wherein substantially randomly distributing units of data comprises using a key.
7. **Claim 7:** The method of claim 1, wherein producing **the plurality of n shares** comprises substantially randomly distributing units of data of **the second data set** into **the plurality of n shares**.

385. The Torre/Desai(/Watanabe)/Orsini System-1 teaches the additional limitations recited in Claims 4–7.

386. As detailed in §IX.C.3.a above [Torre/Desai(/Watanabe)/Orsini System-1], the Torre/Desai(/Watanabe)/Orsini System-1 applies Orsini’s cryptosplit process to the output of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm—that is, $x' = x'[1] \dots x'[n+1]$ —in order to generate four data segments. These segments are then supplied directly to the redundancy generator and the third transformer, which process them to produce the shreds (*i.e.*, the “plurality of n shares”), as illustrated in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.C.1 above.

Torre/Desai(/Watanabe)/Orsini System-1 (Shredder)



EX1003 (Torre), FIG. 32 (annotated); EX1005 (Watanabe), FIG. 2 (annotated);

EX1004 (Desai), 16; EX1006 (Orsini), [0281], [0286].

387. Orsini teaches “splitting the data according to the contents of a generated *random* number, or *key*” (EX1006 (Orsini), [0280]) such that “[t]he units are [] distributed ... *randomly*” into output data segments. EX1006 (Orsini), [0281]. For instance, Orsini describes an example of its “*cryptographic split process*” in which the data is “23 bytes in size, with the data unit size chosen to be one byte” and “[e]ach byte would be distributed into one of the 4 *shares*.” EX1006 (Orsini), [0282]. In this example, “[a]ssuming a random distribution, a key would be obtained to create a sequence of 23 random numbers (r_1, r_2, r_3 through r_{23}), each with a value between 1 and 4 corresponding to the four shares,” and “each of the units of data ... is associated with one of the 23 random numbers corresponding to one of the four *shares*.” EX1006 (Orsini), [0282]. The data units are then distributed into the *shares* based on those associated random values. *See also* EX1006 (Orsini), [0282], [0284] (describing splitting of data using a *key* composed of random numbers).

388. Because Orsini’s *cryptosplit* generates the four data segments by randomly distributing the data units of the output of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm (*i.e.*, $x' = x'[1] \dots x'[n+1]$), the shreds subsequently produced by the redundancy generator and the third transformer in the Torre/Desai/(Watanabe)/Orsini System-1 include randomly distributed data units of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm’s output.

389. **In sum**, with respect to Claim 4, Torre/Desai(/Watanabe)/Orsini System-1 “*caus[es] the n shares [Torre’s shreds] to comprise a substantially random distribution of units of data [Orsini’s cryptosplit randomly distributes data units] of the second data set [Desai’s output data blocks $x[1] \dots x[n+1]$].*”

390. **In sum**, with respect to Claim 5, Torre/Desai(/Watanabe)/Orsini System-1 teaches “*wherein causing the n shares [Torre’s shreds] to comprise a substantially random distribution [Orsini’s cryptosplit randomly distributes data units of the output of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm] comprises using a key [Orsini’s randomly generated key]*”

391. **In sum**, with respect to Claim 6, Torre/Desai(/Watanabe)/Orsini System-1 teaches “*wherein substantially randomly distributing units of data [Orsini’s cryptosplit randomly distributes data units of the output of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm] comprises using a key [Orsini’s randomly generated key].*”

392. **In sum**, with respect to Claim 7, the Torre/Desai(/Watanabe)/Orsini System-1 teaches “*wherein producing the plurality of n shares [Torre’s shreds] comprises substantially randomly distributing units of data of the second data set [Orsini’s cryptosplit randomly distributes data units of the output of Desai’s $E\text{-CTRT}^{F,F^{-1}}$ transformation algorithm] into the plurality of n shares [Torre’s shreds]*”

through a combination of Orsini's cryptosplit process, Torre's redundancy generator, and Torre's third transformer.

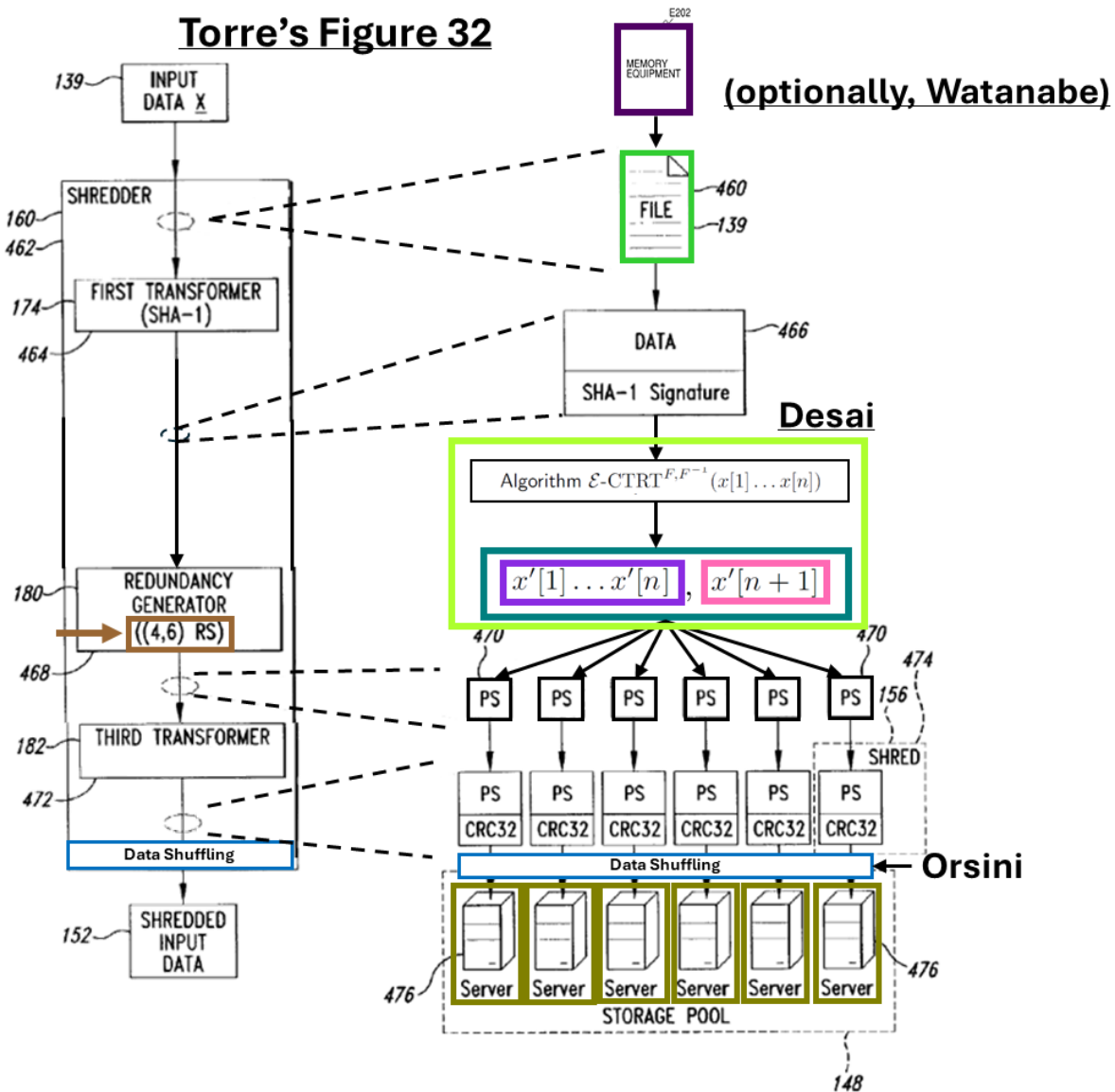
8. **Claim 8: The method of claim 1, wherein producing the plurality of n shares comprises distributing units of data into the plurality of n shares and substantially randomly shuffling the units of data in the shares.**

393. Ground I or Ground II in view of Orsini renders Claim 8 obvious because the Torre/Desai(/Watanabe)/Orsini System-2 teaches the additional limitation recited in Claim 8.

394. As explained in §§IX.A.4 above [Torre/Desai] and IX.C.3.b above [Torre/Desai(/Watanabe)/Orsini System-2], the Torre/Desai(/Watanabe)/Orsini System-2 includes a redundancy generator that generates six pre-shreds 470, which are then processed by the third transformer to form shreds. EX1003 (Torre), [0131]. Following this, Orsini's data shuffling process is applied to the shreds to further randomize the distribution of data units within the shreds. Orsini teaches that "the units are then distributed (either randomly or by a predetermined set of values) into the N shares," and that this process "could also involve a *shuffling of the order of the units in the shares.*" EX1006 (Orsini), [0281], [0284], [0286]. This is shown in the figure below, annotated to use the same color-coding scheme as that described in the table of §IX.C.1 above.

Torre/Desai(/Watanabe)/Orsini System-2 (Shredder)

Torre's Figure 32



EX1003 (Torre), FIG. 32 (annotated); EX1005 (Watanabe), FIG. 2 (annotated); EX1004 (Desai), 16; EX1006 (Orsini), [0281], [0286].

395. **In sum**, Torre/Desai(/Watanabe)/Orsini System-2 teaches “*wherein producing the plurality of n shares [Torre’s shreds] comprises distributing units of*

data into the plurality of n shares [Torre's redundancy generator and third transformer distribute data units into the shreds] and substantially randomly shuffling the units of data in the shares [Orsini's data shuffling process shuffles the data units in the shreds]."

9. Claim 9: The method of claim 8, wherein substantially randomly shuffling the units of data comprises using a key.

396. Ground I or Ground II in view of Orsini renders Claim 9 obvious because the Torre/Desai(/Watanabe)/Orsini System-2 teaches the additional limitation recited in Claim 9.

397. For example, Orsini explains that its cryptosplit and data shuffling process employs a "key" to achieve random distribution of data units. EX1006 (Orsini), [0282], [0284]. Specifically, Orsini teaches generating a "key" composed of a sequence of random numbers and using that key to govern the assignment of individual data units to various shares. Orsini explains that "[a]ssuming a random distribution, a key would be obtained to create a sequence of 23 random numbers (r1, r2, r3 through r23), ... The distribution of the bytes of data into the four shares would occur by placing the first byte of the data into share number r1, byte two into share r2, byte three into share r3, through the 23rd byte of data into share r23." EX1006 (Orsini), [0282]. *See also* EX1006 (Orsini), [0281].

398. **In sum**, Torre/Desai(/Watanabe)/Orsini System-2 teaches “*wherein substantially randomly shuffling the units of data [Orsini’s data shuffling process shuffles the data units in the shreds] comprises using a key [the key disclosed in Orsini].*”

10. Claims 16-21 and 28-29

399. The Torre/Desai(/Watanabe)/Orsini System renders obvious claims 16-21 and 28-29.

- The Torre/Desai(/Watanabe)/Orsini System satisfies the additional limitations of claims 16 and 28 for the same reasons discussed in §IX.C.4 above [Claim 4];
- The Torre/Desai(/Watanabe)/Orsini System satisfies the additional limitations of claims 17 and 29 for the same reasons discussed in §IX.C.4 above [Claim 5];
- The Torre/Desai(/Watanabe)/Orsini System satisfies the additional limitations of claim 19 for the same reasons discussed in §IX.C.4 above [Claim 6];
- The Torre/Desai(/Watanabe)/Orsini System satisfies the additional limitations of claim 18 for the same reasons discussed in §IX.C.4 above [Claim 7];
- The Torre/Desai(/Watanabe)/Orsini System satisfies the additional limitations of claim 20 for the same reasons discussed in §IX.C.8 above [Claim 8]; and

- The Torre/Desai(/Watanabe)/Orsini System satisfies the additional limitations of claim 21 for the same reasons discussed in §IX.C.9 above [Claim 9].

X. Conclusion

400. In my opinion, the challenged claims 1–30 of the '456 Patent are unpatentable.

XI. Availability for Cross-Examination

401. In signing this declaration, I recognize that the declaration will be filed as evidence in a case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross examination in the case and that cross examination will take place within the United States. If cross examination is required of me, I will appear for cross examination within the United States during the time allotted for cross examination.

XII. Right to Supplement

402. I reserve the right to supplement my opinions in the future to respond to any arguments that the Patent Owner may raise and to take into account new information as it becomes available to me.

XIII. Jurat

403. I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false

Declaration of Dr. Erez Zadok
U.S. Patent No. 9,135,456

statements and the like so made are punishable by fine or imprisonment, or both,
under Section 1101 of Title 18 of the United States Code.

A handwritten signature in black ink, appearing to read "Erez Zadok", written in a cursive style.

July 9, 2025

Dr. Erez Zadok