UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

NINTENDO CO., LTD., and

NINTENDO OF AMERICA INC.,

Petitioners,

v.

ANCORA TECHNOLOGIES, INC.,

Patent Owner.

PTAB Case No. IPR2021-01338

Patent No. 6,411,941 B1

DECLARATION OF ANDREW WOLFE IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 6,411,941

TABLE OF CONTENTS

I. INTRODUCTION 1

A. Qualifications 2

1. Education 2

2. Work Experience 2

3. Curriculum Vitae 6

B. Materials Reviewed 7

C. Level of Ordinary Skill in the Art 7

D. Summary of Opinions 9

II. OVERVIEW OF THE TECHNOLOGY 9

A. Priority Date of the Claims 9

B. Overview of Relevant Technology When the '941 Patent Was Filed 10

1. Software Licenses 10

2. Computer BIOS 12

C. The '941 Patent 14

D. Claim Construction 21

III. OVERVIEW OF THE PRIOR ART 21

A. Hellman 21

B. Chou 31

C. Schneck 35

IV. UNPATENTABILITY OF THE '941 PATENT CLAIMS 42

A. Standards for Invalidity 42

B. Claim 1 43

1. Preamble: "A method of restricting software operation within a license for use with a computer including an erasable, non-volatile memory area of a BIOS of the computer, and a volatile memory area; the method comprising the steps of:" 43

Nintendo - Ancora Exh. 1003

2. Element 1.a: "selecting a program residing in the volatile memory" 55

3. Element 1.b: "using an agent to set up a verification structure in the erasable, non-volatile memory of the BIOS, the verification structure accommodating data that includes at least one license record" 60

4. Element 1.c: "verifying the program using at least the verification structure from the erasable non-volatile memory of the BIOS, and" 69

5. Element 1.d: "acting on the program according to the verification." 71

C. Claim 2: "A method according to claim 1, further comprising the steps of: establishing a license authentication bureau." 71

D. Claim 3 72

1. Preamble: "A method according to claim 2, wherein setting up a verification structure further comprising the steps of:" 72

2. Element 3.a: "establishing, between the computer and the bureau, a two-way data-communications linkage;" 73

3. Element 3.b: "transferring, from the computer to the bureau, a request-for-license including an identification of the computer and the license-record's contents from the selected program;" 73

4. Element 3.c: "forming an encrypted license-record at the bureau by encrypting parts of the request-for-license using part of the identification as an encryption key;" 77

5. Element 3.d: "transferring, from the bureau to the computer, the encrypted license-record; and" 79

6. Element 3.e: "storing the encrypted license record in the erasable non-volatile memory area of the BIOS." 79

E. Claim 6: "A method according to claim 1 wherein selecting a program includes the steps of: establishing a licensed-software-program in the volatile memory of the computer wherein said licensed-software-program includes contents used to form the license-record." 80

Nintendo - Ancora Exh. 1003

F. Claim 7 81

1. Preamble: "A method according to claim 6 wherein using an agent to set up the verification structure includes the steps of:".....81

2. Element 7.a: "establishing or certifying the existence of a pseudo-unique key in a first non-volatile memory area of the computer; and".....81

3. Element 7.b: "establishing at least one license-record location in the first nonvolatile memory area or in the erasable, non-volatile memory area of the BIOS.".....83

G. Claim 8 84

1. Preamble: "A method according to claim 6 wherein establishing a license-record includes the steps of:" 84

2. Element 8.a: "forming a license-record by encrypting of the contents used to form a license-record with other predetermined data contents, using the key; and" 84

3. Element 8.b: "establishing the encrypted license-record in one of the at least one established license-record locations." 85

H. Claim 9 86

1. Preamble: "A method according to claim 7 wherein verifying the program includes the steps of:".....86

2. Element 9.a: "encrypting the licensed-software-program's license-record contents from the volatile memory area or decrypting the license-record in the erasable, non-volatile memory area of the BIOS, using the pseudo-unique key; and".....86

3. Element 9.b: "comparing the encrypted licenses-software-program's license-record contents with the encrypted license-record in the erasable, non-volatile memory area of the BIOS, or comparing the license-software-program's license-record contents with the decrypted license-record in erasable non-volatile memory area of the BIOS.".....87

Nintendo - Ancora Exh. 1003

I. Claim 10: "A method according to claim 9 wherein acting on the program includes the step: restricting the program's operation with predetermined limitations if the comparing yields non-unity or insufficiency." 89

J. Claim 11: "A method according to claim 1 wherein the volatile memory is a RAM." 90

K. Claim 12: "The method of claim 1, wherein a pseudo-unique key is stored in the non-volatile memory of the BIOS." 90

L. Claim 13: "The method of claim 1, wherein a unique key is stored in a first non-volatile memory area of the computer." 93

M. Claim 14: "The method according claim 13, wherein the step of using the agent to set up the verification record, including the license record, includes encrypting a license record data in the program using at least the unique key." 94

N. Claim 16: "The method according to claim 13, wherein the step of verifying the program includes a decrypting the license record data accommodated in the erasable second non-volatile memory area of the BIOS using at least the unique key." 95

Nintendo - Ancora Exh. 1003

LIST OF APPENDICES

Appendix A *Curriculum Vitae* of Andrew Wolfe, Ph.D.

Appendix B Documents Cited

Nintendo - Ancora Exh. 1003

I. INTRODUCTION

1. I, Andrew Wolfe, have been retained by Petitioner Nintendo of America Inc. ("Petitioner") to investigate and opine on certain issues relating to United States Patent No. 6,411,941 ("the '941 patent") in their Petition for Inter Partes Review of that patent. The Petition requests that the Patent Trial and Appeal Board ("PTAB" or "Board") review and cancel claims 1-3, 6-14, and 16 of the '941 patent.

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

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

4. I understand that the '941 patent has been assigned to Ancora Technologies, Inc.

5. This declaration is based on the information currently available to me. To the extent that additional information becomes available, I reserve the right to

Nintendo - Ancora Exh. 1003

continue my investigation and study, which may include a review of documents and information that may be produced, as well as testimony from depositions that have not yet been taken.

A. Qualifications

1. Education

6. In 1985, I earned a B.S.E.E. degree in Electrical Engineering and Computer Science from The Johns Hopkins University. In 1987, I received an M.S. degree in Electrical and Computer Engineering from Carnegie Mellon University. In 1992, I received a Ph.D. in Computer Engineering from Carnegie Mellon University. My doctoral dissertation proposed a new approach for the architecture of a computer processor.

2. Work Experience

7. I have more than 35 years of experience as a computer architect, computer system designer, personal computer graphics designer, educator, and executive in the electronics industry.

8. In 1983, I began designing touch sensors, microprocessor-based computer systems, and I/O (input/output) cards for personal computers as a senior design engineer for Touch Technology, Inc. During the course of my design projects with Touch Technology, I designed I/O cards for PC-compatible computer systems, including the IBM PC-AT, to interface with interactive touch-based computer

Nintendo - Ancora Exh. 1003

terminals that I designed for use in public information systems. I continued designing and developing related technology as a consultant to the Carroll Touch division of AMP, Inc., where in 1986 I designed one of the first custom touch-screen integrated circuits. I designed the touch/pen input system for the Linus WriteTop, which many believe to be the first commercial tablet computer.

9. From 1986 through 1987, I designed and built a high-performance computer system as a student at Carnegie Mellon University. From 1986 through early 1988, I also developed the curriculum and supervised the teaching laboratory for processor design courses.

10. In the latter part of 1989, I worked as a senior design engineer for ESL-TRW Advanced Technology Division. While at ESL-TRW, I designed and built a bus interface and memory controller for a workstation-based computer system, and also worked on the design of a multiprocessor system.

11. At the end of 1989, I (along with some partners) reacquired the rights to the technology I had developed at Touch Technology and at AMP and founded The Graphics Technology Company. Over the next seven years, as an officer and a consultant for The Graphics Technology Company, I managed the company's engineering development activities and personally developed dozens of touch screen sensors, controllers, and interactive touch-based computer systems.

Nintendo - Ancora Exh. 1003

12. I have consulted, formally and informally, for a number of fabless semiconductor companies. In particular, I have served on the technical advisory boards for two processor design companies: BOPS, Inc., where I chaired the board; and Siroyan Ltd., where I served in a similar role for three networking chip companies—Intellon, Inc., Comsilica, Inc., and Entridia, Inc.—and one 3D game accelerator company, Ageia, Inc.

13. I have also served as a technology advisor to Motorola and to several venture capital funds in the U.S. and Europe. Currently, I am a director of Turtle Beach Corporation, providing guidance in its development of premium audio peripheral devices for a variety of commercial electronic products.

14. From 1991 through 1997, I served on the Faculty of Princeton University as an Assistant Professor of Electrical Engineering. At Princeton, I taught undergraduate and graduate-level courses in Computer Architecture, Advanced Computer Architecture, Display Technology, and Microprocessor Systems, and conducted sponsored research in the area of computer systems and related topics. I was also a principal investigator for DOD research in video technology and a principal investigator for the New Jersey Center for Multimedia Research. From 1999 through 2002, while a Consulting Professor, I taught a Computer Architecture course to both undergraduate and graduate students at Stanford University. At Princeton, I received several teaching awards, both from students and from the

Nintendo - Ancora Exh. 1003

School of Engineering. I have also taught advanced microprocessor architecture to industry professionals in seminars sponsored by the Institute of Electrical and Electronics Engineers ("IEEE") and the Association for Computing Machinery ("ACM"). I am currently a lecturer at Santa Clara University teaching courses on Microprocessor Systems, Real-Time Computing, and Mechatronics.

15. From 1997 through 2002, I held a variety of executive positions at a publicly-held fabless semiconductor company originally called S3, Inc. and later called SonicBlue Inc. I held the positions of Chief Technology Officer, Vice President of Systems Integration Products, Senior Vice President of Business Development, and Director of Technology, among others. At the time I joined S3, the company supplied graphics accelerators for more than 50% of the PCs sold in the United States. At S3 I supervised the design of several PC graphics accelerators. During my time at SonicBlue we launched more than 30 new consumer electronics products including devices to support copy-protected video and many of the first commercial products to support copy-protected internet audio content.

16. I have published more than fifty peer-reviewed papers in computer architecture and computer systems and IC design. I also have chaired IEEE and ACM conferences in microarchitecture and integrated circuit design and served as an associate editor for IEEE and ACM journals. I served on the IEEE Computer Society Awards committee. I am a Senior Member of IEEE and a Member of ACM.

Nintendo - Ancora Exh. 1003

I am a named inventor on at least fifty-six U.S. patents and thirty-seven foreign patents, which are listed in my curriculum vitae.

17. In 2002, I was the invited keynote speaker at the ACM/IEEE International Symposium on Microarchitecture and at the International Conference on Multimedia. From 1990 through 2005, I have also been an invited speaker on various aspects of technology and the PC industry at numerous industry events including the Intel Developer's Forum, Microsoft Windows Hardware Engineering Conference, Microprocessor Forum, Embedded Systems Conference, Comdex, and Consumer Electronics Show, as well as at the Harvard Business School and the University of Illinois Law School. I have been interviewed on subjects related to computer graphics and video technology and the electronics industry by publications such as the Wall Street Journal, New York Times, Los Angeles Times, Time, Newsweek, Forbes, and Fortune as well as on CNN, NPR, and the BBC. I have also spoken at dozens of universities including MIT, Stanford, University of Texas, Carnegie Mellon University, UCLA, University of Michigan, Rice University, and Duke University.

3. Curriculum Vitae

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

Nintendo - Ancora Exh. 1003

B. Materials Reviewed

19. My opinions expressed in this declaration are based on documents and materials identified in this declaration, including the '941 patent, the prior art references and background materials discussed in this declaration, and the other references specifically identified in this declaration. I have considered these materials in their entirety, even if only portions are discussed here.

20. I have also relied on my own experience and expertise in digital security, software licensing, and computer architecture.

C. Level of Ordinary Skill in the Art

21. I am not an attorney and offer no legal opinions. I have been informed about certain aspects of the law for purposes of my analyses and opinions.[1]

22. I understand that in analyzing questions of invalidity, the perspective of a person having ordinary skill in the art ("POSA") is often implicated, and the Board may need assistance in determining that level of skill.

23. I understand that the claims and written description of a patent must be understood from the perspective of a POSA. I have been informed that the following

---

[1] I understand that the patent laws were amended by the America Invents Act (AIA), but that the earlier statutory requirements still apply to pre-AIA patents. I have been informed that the '941 Patent is a pre-AIA patent, so the pre-AIA requirements control. Unless otherwise stated, my understanding of the law about patent invalidity as set forth in this declaration relates to the pre-AIA requirements.

Nintendo - Ancora Exh. 1003

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

24. Based on my experience in digital security, software licensing, and computer architecture, as well as my reading of the '941 Patent, it is my opinion that a person of ordinary skill with respect to the subject matter of the '941 Patent at the time of the alleged invention would have had at least a B.S. degree in computer science, computer engineering, or electrical engineering (or equivalent experience) and would have had at least two years of experience with computer science and computer engineering, including information encryption, computer architecture, and firmware programming. This definition is approximate, and additional educational experience in computer science and computer engineering could make up for less work experience and vice versa.

25. I am a person of at least ordinary skill in the art and was so on the date to which the '941 Patent claims priority (May 21, 1998). As shown by my qualifications and my curriculum vitae attached as Appendix A, I am aware of the knowledge and skill possessed by a person of ordinary skill in the art at the time of

Nintendo - Ancora Exh. 1003

the alleged invention claimed by the '941 Patent. In performing my analysis, I have applied the standard set forth above.

D. Summary of Opinions

26. I have reviewed and analyzed the '941 Patent (Ex. B-1, same as Ex. 1001 in the Petition) as well as prior art references Hellman (U.S. Patent 4,658,093) (Ex. B-3, same as Ex. 1004 in the Petition), Chou (U.S. Patent 5,892,906) (Ex. B-4, same as Ex. 1005 in Petition), and Schneck (U.S. Patent 5,933,498) (Ex. B-5, same as Ex. 1006 in Petition).

27. Based on my review and analysis, it is my opinion that claims 1-2, 11, and 13 of the '941 Patent are invalid as obvious based on Hellman in view of Chou. Based on my review and analysis, it is also my opinion that claims 1-3, 6-14, and 16 of the '941 Patent are invalid as obvious based on Hellman in view of Chou and Schneck.

II. OVERVIEW OF THE TECHNOLOGY

A. Priority Date of the Claims

28. I have been informed that a U.S. patent application may claim the benefit of the filing date of an earlier patent application if the earlier patent application disclosed each limitation of the invention claimed in the later-filed U.S. patent application. I have also been informed that priority is determined on a claim-by-claim basis so that certain claims of a patent may be entitled to the priority date

Nintendo - Ancora Exh. 1003

of an earlier-filed patent application even if other claims of the same patent are not entitled to that priority date.

29. I have also been informed that for patent applications filed before March 16, 2013, a patented claim is invalid if the claimed invention was patented or described in a printed publication in any country more than one year before the effective filing date of the claim, regardless of when the applicant conceived of the claimed invention.

30. I understand that the '941 Patent claims a priority date of May 21, 1998.

B. Overview of Relevant Technology When the '941 Patent Was Filed

1. Software Licenses

31. By the time of the '941 Patent's priority date in 1998, the field of software licensing was well-developed. Since at least the 1980s, practitioners in the field had widely recognized the new risks to software piracy introduced by the transformations to digital media.

32. Many entities recognized that one such risk was "copy protection" or "secondary distribution." Secondary distribution contrasted with, for example, preventing an unauthorized user from obtaining access to a software program in the first place. Secondary distribution dealt with the more challenging problem of allowing a user to have an authorized access to the software program but preventing

Nintendo - Ancora Exh. 1003

the user from then making unauthorized copies and distributing those copies. This problem was more challenging because it required some level of trust in the user but balanced against the possibility that the user may still have malicious motivations.

33. For secondary distribution, as with other forms of piracy prevention, encryption was considered a key tool to providing protection. Encryption was a leading solution for various reasons. Encryption was easy to implement but hard to break, making it an efficient solution. Encryption also allowed user-specific and device-specific solutions, given that different devices could be given different encryption/decryption keys.

34. European patent Application EP 0766165A2, Ex. B-6 ("'165 Application"), which published in 1997 from an application filed in 1996, disclosed a license notification system. The '164 Application disclosed sending encoded license information to a user terminal, with the license information encoded with a key specific to the user terminal. The user terminal checks the license information when the user operates a software program. If the license information is valid, then the licensee's name is displayed.

35. U.S. Patent 5,724,425, Ex. B-6 ("'425 Patent"), which issued in 1998 from an application filed in 1994, disclosed a "software passport." The software passport was formed by encrypting a message digest using an application writer's private key, a license, and the software program binary code. A user's computer

Nintendo - Ancora Exh. 1003

uses the encrypted message digest and the license to determine if the software program is secure to operate. The '425 Patent disclosed this technique to deal with the risk of users purchasing pirated software when they thought they were purchasing legitimate software.

2. Computer BIOS

36. By the time of the '941 Patent's priority date in 1998, the field of computer BIOS was well-developed. BIOS began to be used at least as far back as the 1970s, for example in 8-bit computers that ran the CP-M operating system. The usage of BIOS increased rapidly, and by 1998 BIOS was present in essentially all general-purpose computers, e.g., personal computers and servers. In these situations, BIOS provided the basic software routines that were run when the computer was first powered on. One of the primary responsibilities of BIOS was to load the operating system code and allow it to start executing, often called "booting" the computer.

37. For many years, including through to 1998, it was typical to provide BIOS in a separate memory module, apart from the main memory. These came about for numerous reasons. As one reason, the BIOS programs needed to be secure and away from other program code. Namely, accidentally overwriting or destroying the BIOS program could permanently disable the computer. So storing it on a separate memory module was considered a good approach. As another reason, early

Nintendo - Ancora Exh. 1003

versions of BIOS were expected to remain static through the life of the device. As such, it was common to provide BIOS programs in a true read only memory (ROM). By "true" ROM, I mean a memory chip that could not have its contents changed, whether electronically or otherwise. Using true ROM also provided the benefit of not allowing BIOS to be accidentally modified, which was beneficial as described above. Additionally, it was advantageous to provide the BIOS in a non-volatile memory so that it was present when the computer was powered on.

38. By the 1990s, it became more common to store BIOS programs in alterable memory, i.e., memory that could be rewritten. This became more common at least in part because computer manufacturers came to realize that there was a benefit to being able to modify the BIOS programs "in the field," as opposed to have those programs completely static for the life of the devices.

39. Among these forms of rewritable memory, electrically-erasable programmable read-only memory (EEPROM) was a popular technology. EEPROM was considered beneficial for a number of reasons. For one reason, EEPROM could be rewritten using simple memory access routines that could be programmatically controlled. This provided the sort of flexibility that computer manufacturers were seeking. For another reason, EEPROM could be implemented as "flash memory," which was both reliable (not prone to unexpected loss of data) and cost effective (relatively less expensive than some other rewritable ROM technologies).

Nintendo - Ancora Exh. 1003

40. U.S. Patent 6,138,236, Ex. B-7 ("'236 Patent"), which issued in 2000 from an application filed in 1996, disclosed the use of both "boot ROM (read only memory)" and "boot PROM (programmable read only memory)." The '236 Patent explained that the boot PROM could be implemented as flash PROM, "often referred to as flash memory."

41. U.S. Patent 5,802,592, Ex. B-8 ("'592 Patent"), which issued in 1998 from an application filed in 1996, disclosed a technique for verifying the integrity BIOS programs stored in "alterable read only memory (such as FLASH ROM)."

42. U.S. Patent 5,835,594, Ex. B-9 ("'594 Patent"), which issued in 1998 from an application filed 1996, disclosed a system for protecting the content, such as BIOS updates, in "FLASH memory or erasable programmable read-only-memory (EPROM)."

C. The '941 Patent

43. The '941 Patent describes a "method of restricting software operation within a license limitation." '941 Patent, Abstract. The '941 Patent explains that there were many known techniques for restricting the operation of an unauthorized software program. '941 Patent, 1:12-17. The '941 Patent indicates that these techniques were "primarily motivated by the grand proliferation of illegally copied software, which is engulfing the marketplace," and commented on the large financial impact of this illegal copying. '941 Patent, 1:12-17.

Nintendo - Ancora Exh. 1003

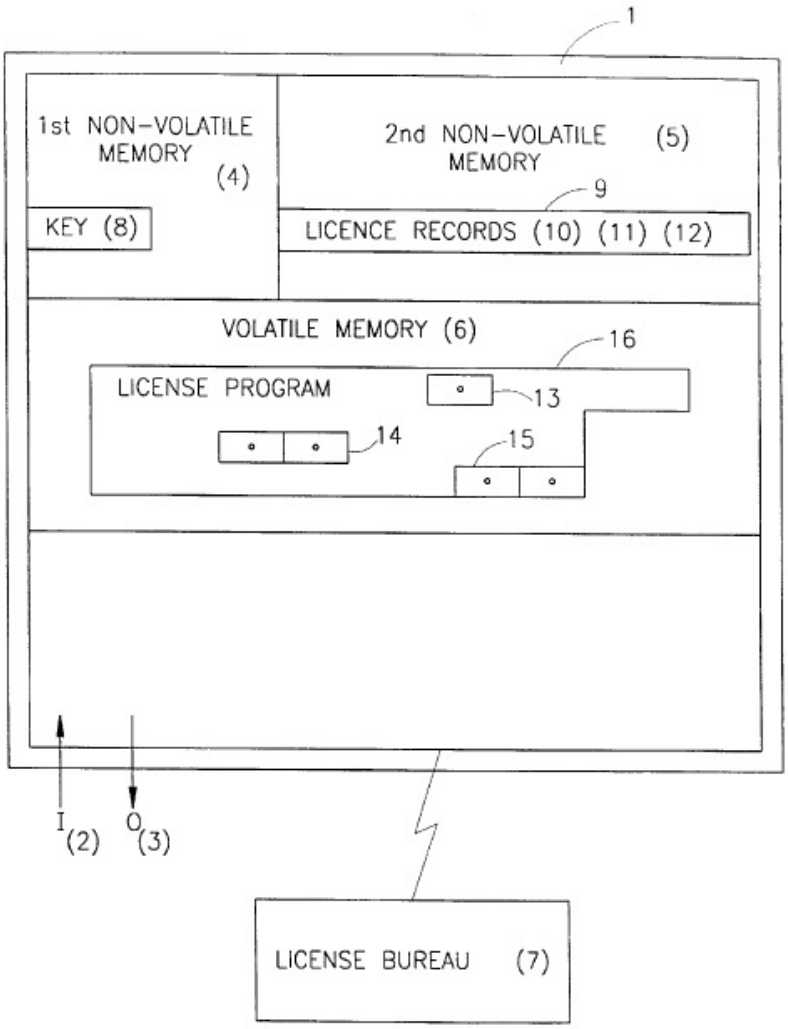
44. The ’941 Patent indicates that one prior art technique involved “writing a license signature onto the computer’s volatile memory (e.g., hard disk).” ’941 Patent, 1:19-26. The ’941 Patent explained that this technique was “very vulnerable to attack at the hands of skilled system’s programmers (e.g. ‘hackers’).” ’941 Patent, 1:19-26.

45. The ’941 Patent indicates that hardware-based techniques, such as the use of a “dongle” were “expensive, inconvenient, and not particularly suitable for software that may be sold by downloading.” ’941 Patent, 1:27-32.

46. Against that backdrop, the ’941 Patent discloses its technique with respect to a computer configuration shown in Figure 1 and a process shown in Figure 2.

47. The computer configuration of Figure 1 (shown below) contains numerous storage devices. The storage devices include the first non-volatile memory area 4, the second non-volatile memory area 5, and the volatile memory area 6. ’941 Patent, Abstract, Figure 1, 5:9-16. The first non-volatile memory area 4 stores a key 8. ’941 Patent, Figure 1, 5:19-24. The second non-volatile memory area 5 has a license record area 9 with license records 10, 11, 12. ’941 Patent, Figure 1, 5:25-33. The volatile memory area 6 include a license program 16, which has license record field 13, 14, and 15. ’941 Patent, Figure 1, 5:25-33. The computer can communicate with a license bureau 7. ’941 Patent, Figure 1, 5:17-18.

Nintendo - Ancora Exh. 1003



’941 Patent, Figure 1.

48. The ’941 Patent provides an example implementation of the invention in a “conventional computer having a conventional BIOS module.” ’941 Patent, 1:43-52. The computer can have a “ROM section” with a key embedded therein at the time of manufacture. ’941 Patent, 1:43-52. “The key constitutes, effectively, a unique identification code for the host computer.” ’941 Patent, 1:43-52. The key cannot be removed or modified. ’941 Patent, 1:43-52.

49. An "application program that is to be licensed to run on the specified computer, is associated with a license record." '941 Patent, 1:53-58. "The license record may be held in either encrypted to explicit form." '941 Patent, 1:53-58.

50. A license establishment procedure is then performed. '941 Patent, 1:59-2:9. "[A] verification structure is set in the BIOS so as to indicate that the specified program is licensed to run on the specified computer." '941 Patent, 1:59-2:9. "This is implemented by encrypting the license record … using said key … as an encryption key." '941 Patent, 1:59-2:9. "The resulting encrypted license record is stored in another (second) non-volatile section of the BIOS, e.g., E^2PROM[2] (or the ROM)." '941 Patent, 1:59-2:9. The '941 Patent notes that "unlike the first non-volatile section, the data in the second non-volatile memory may optionally be erased or modified (using E2PROM manipulation commands), so as to enable to add, modify or remove licenses." '941 Patent, 1:59-2:9.

51. Once the encrypted license record is stored "in the second non-volatile memory (e.g. E^2PROM)," a process for verifying the license can be performed. '941 Patent, 2:10-2:26. When a "program is loaded into the memory of the computer," the verification process is performed. '941 Patent, 2:10-2:26. A license record is retrieved from the program. '941 Patent, 2:10-2:26. The license record is then

[2] E^2PROM is another spelling of EEPROM.

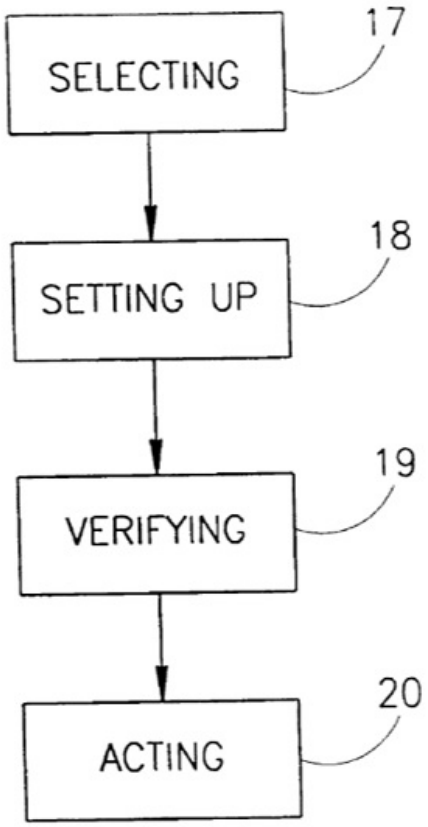
Nintendo - Ancora Exh. 1003

encrypted using the unique key from ROM. '941 Patent, 2:10-2:26. That encrypted license record is then compared "to the encrypted records that reside in the E^2PROM." '941 Patent, 2:10-2:26. If there is a match, then "the program is verified to run on the computer." '941 Patent, 2:10-2:26. If there is not a match, then "the program under question is not properly licensed" and appropriate action is taken. '941 Patent, 2:10-2:26.

52. The '941 Patent describes this technique again with respect to the process of Figure 2, shown below. At step 17, a program is selected. '941 Patent, 6:7-17. This can include "establishing a licensed-software-program in the volatile memory of the computer. '941 Patent, 6:7-17. At step 18, the verification structure is set up. '941 Patent, 6:17-28. This can include "certifying the existence of a pseudo-unique key in the first non-volatile memory area" and "establishing at least one license-record location in the first or the second nonvolatile memory area." '941 Patent, 6:17-28. Establishing the license record can include encrypting contents and storing the encrypted license-record in one of the "established license-record locations." '941 Patent, 6:17-28. At step 19, the program is verified. '941 Patent, 6:28-38. This can include encrypting license record contents from the program, and comparing the result with "the encrypted license-record in the first or the second non-volatile memory area." '941 Patent, 6:28-38. At step 20, the program is acted on. '941 Patent, 6:40-52. This can include "restricting the program's operation with

Nintendo - Ancora Exh. 1003

the predetermined limitations if the comparing yields non-unite or insufficiency." '941 Patent, 6:40-52.



'941 Patent, Figure 2.

53. The '941 Patent explains various purported benefits/improvements with the disclosed techniques.

54. The '941 Patent alleges that "[t]hose versed in the art will readily appreciate that any attempt to run a program at an unlicensed site will be immediately detected." '941 Patent, 2:28-35. If the program is not licensed on a specific computer, then that will be detected because there will not be an appropriate

Nintendo - Ancora Exh. 1003

license record encrypted with that computer's unique key and stored in the E^2PROM. '941 Patent, 2:28-35.

55. Further, a user cannot subvert this protection by copying a license record in the E^2PROM of a first computer to the E^2PROM of a second computer. '941 Patent, 2:37-59. If this were done, then when the verification of the program were performed on the second computer using the key of the second computer, the encryption result would not match the copied license record, which would have been encrypted with the key of the first computer. '941 Patent, 2:37-59. And the user cannot change the key because it is ROM. '941 Patent, 2:37-59.

56. The '941 Patent also says that storing the license record in BIOS improves the securing of that information. "An important advantage in utilizing non-volatile memory such as that residing in the BIOS is that the required level of system programming expertise that is necessary to intercept or modify commands, interacting with the BIOS, is substantially higher than those needed for tampering with data residing in volatile memory such as hard disk." '941 Patent, 3:4-17. "Furthermore, there is a much higher cost to the programmer, if his tampering is unsuccessful, i.e. if data residing in the BIOS (which is necessary for the computer's operability) is inadvertently changed by the hacker. This is too high of a risk for the ordinary software hacker to pay." '941 Patent, 3:4-17.

Nintendo - Ancora Exh. 1003

D. Claim Construction

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

58. For all terms of the challenged claims of the '941 patent, I have interpreted them as they would have been understood by a POSA at the time of the invention, *i.e.*, May 21, 1998.

III. OVERVIEW OF THE PRIOR ART

A. Hellman

59. Hellman's disclosure, which dates back to its 1983 filing date, describes a system for software distribution. In the system, software (including "programs") can be authorized for a given number of uses on a base unit (including a "computer"). Hellman, Abstract. The authorization for additional uses comes from the software's manufacturer. Hellman, Abstract. Hellman discloses a technique whereby the authorization message cannot be reused. Hellman, Abstract. The authorization can be specific to a base unit, "so that an authorization for one base unit cannot be transferred to another base unit." Hellman, Abstract. Hellman represents that its

Nintendo - Ancora Exh. 1003

technique "solves the 'software piracy problem'." Hellman, Abstract. As such, Hellman seeks to solve the same problem as the '941 Patent: using computer-specific authorizations to use a program that prevents unauthorized uses on other computers.

60. Hellman provides more explanation on the "software piracy problem" and the objectives of its disclosure. Hellman explains that "'software piracy' is a major problem in the computer and videogame industry." Hellman, 1:15-16. Hellman discusses a number of existing solutions to software piracy, such as storing a program in a non-standard format and providing a program on a physical disk with a physical defect. Hellman, 1:39-2:6. Hellman says that it would also be beneficial to allow a limit on the number of uses of a program, but that techniques for so limiting program use do not provide for protection against copying once the user has access to the full program. Hellman, 2:7-53.

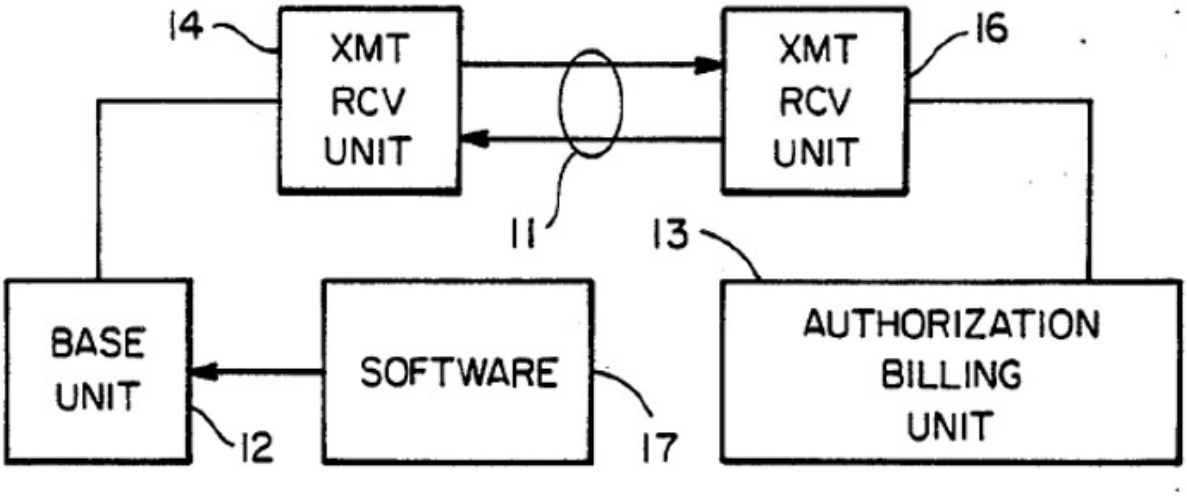
61. Hellman explains that cryptography is a possible tool for solving these problems, and Hellman discusses various relevant cryptography technologies. Hellman, 2:54-4:21.

62. Hellman sets out three objections to be achieved by its disclosure. First, a software manufacturer should be able to control the number of times a piece of software is used, and the authorization should not be recordable and reusable, and should not be transferable between base units. Hellman, 4:22-27. Second, software

Nintendo - Ancora Exh. 1003

should be able to be sold over telephone or other similar communications channels, without the authorization being reusable on any base unit other than the licensed one. Hellman, 4:28-33. Third, Hellman aims to prevent software piracy, i.e., "the illegal use of software on a base unit which has not paid a license fee." Hellman, 4:34-36.

63. Hellman depicts the system of its disclosure in Figure 1, shown below. The system includes a base unit 12 and an authorization and billing unit 13 that communicate over an insecure channel 11. Hellman, 5:39-50. The user of base unit 12 obtains "software package 17 by purchasing it at a store, over telephone line, or in some similar manner." Hellman, 5:51-56.



Hellman, Figure 1.

64. The base unit 12 generates a "user originated request for software use" for the software package 17. Hellman, 5:57-6:2. The request contains various elements of information. Hellman, 5:57-6:2. The request includes a software name,

Nintendo - Ancora Exh. 1003

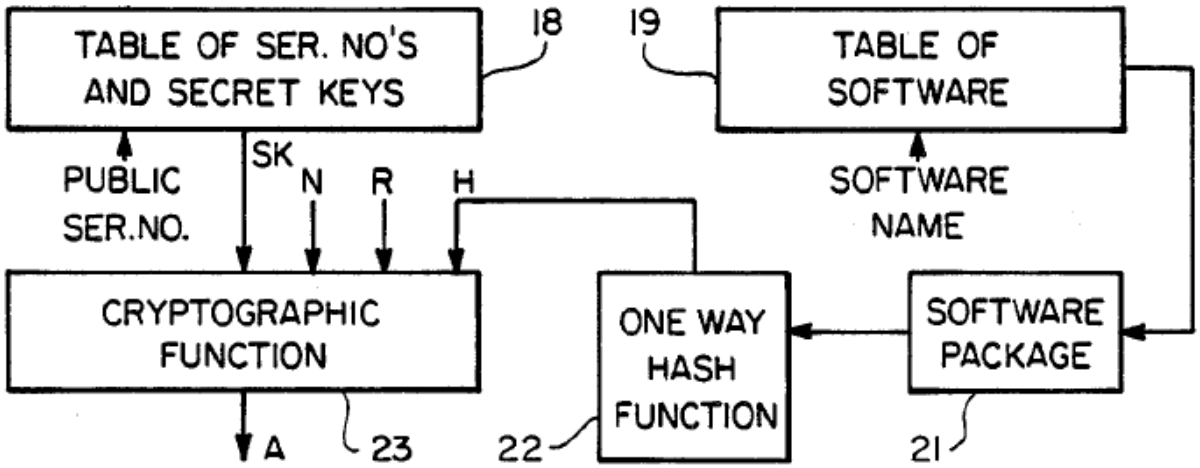
a serial number, a value N, a value R, and billing information. Hellman, 5:57-6:2. The software name is the name of the software package 17. Hellman, 5:57-6:2. The serial number is "a serial number, an identification number, user name or similar identifier unique to base unit 12." Hellman, 5:57-6:2. The value N is how many additional uses are requested. Hellman, 5:57-6:2. The value R is a "random number, counter value, or other non-repeating number generated by the base unit 12." Hellman, 5:57-6:2. The billing information is "a credit car[d] number or similar means for billing the user." Hellman, 5:57-6:2.

65. The base unit 12 transmits the request to the authorization and billing unit 13 over the insecure channel 11. Hellman, 5:57-6:2. The authorization and billing unit 13 receives the request, generates an authorization A "for that particular base unit 12 to use the software package 17 an additional N times" and then transmits the authorization A to the base unit 12. Hellman, 6:3-15. The base unit 12 checks whether the authorization A is correct, and if so updates its memory to allow N additional uses of software package 17. Hellman, 6:3-15.

66. Hellman depicts the operation of the authorization and billing unit 13 in Figure 2, shown below. The authorization and billing unit 13 stores a table of serial numbers and secrets keys in memory 18. Hellman, 6:16-30. The authorization and billing unit 13 uses the serial number received from the base unit 12 to determine the secret key, SK, for the base unit 12. Hellman, 6:16-30. The authorization and

Nintendo - Ancora Exh. 1003

billing unit 13 stores a table of software in memory 19 that allows it to determine a software package 21 from the software name provided in the request, and software package 21 is identical to software package 17. Hellman, 6:16-30.



Hellman, Figure 2.

67. A one-way hash function 22 take the software package 21 as input and generates a hash value H. Hellman, 6:31-61. "This output H is used as an 'abbreviation' or name for describing the software package 21." Hellman, 6:31-61. The value H is easily computable from the software package 21 using the one-way has function generator 22, but "given an H value it is difficult, taking perhaps millions of years, to computer any other software package which produces this same H value." Hellman, 6:31-61. The hash value H is much smaller than the software program 21, with the former containing "perhaps 100 bits," and the latter containing "typically 10,000 to 1,000,000 bits." Hellman, 6:31-61. Storage of the hash value

Nintendo - Ancora Exh. 1003

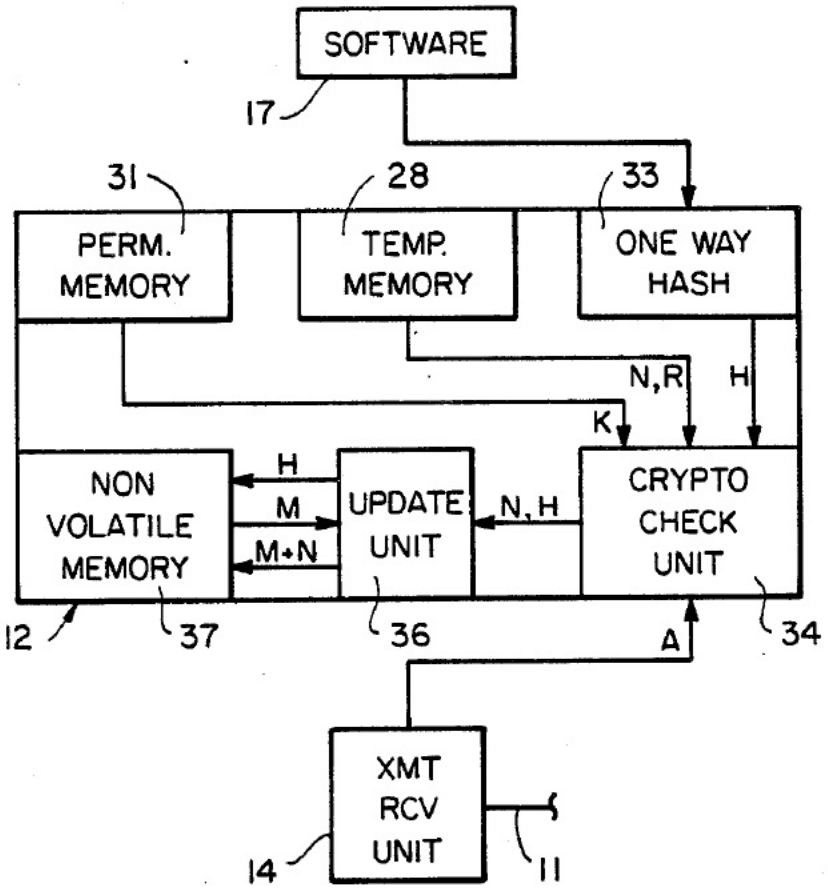
H is preferred to the software name, because, unlike the hash value H, the software name can be easily modified for purposes of circumventing the protections of Hellman. Hellman, 6:31-61.

68. A cryptographic function generator 23 takes as inputs the hash value H, the number of additional uses N, and the random number R, encrypts the inputs using the secret key SK, and thereby generates the authorization A. Hellman, 6:63-7:16. Because the secret key SK is not publicly known, the authorization A can be transmitted over the insecure channel 11 without the risk that it could be decrypted and modified. Hellman, 6:63-7:16. Because the authorization A is encrypted with the secret key SK that is unique to the base unit 12, the authorization A, if intercepted on the insecure channel 11, cannot be reused on another base unit 12 (which would have a different secret key). Hellman, 6:63-7:16. Because the authorization A contains the hash value H, the authorization A, if intercepted on the insecure channel 11, cannot be reused for any other software package (which would have a different hash value). Hellman, 6:63-7:16. Because the authorization A contains the number of uses, N, the authorization A, if intercepted on the insecure channel 11, cannot be reused for a different number of authorized uses. Hellman, 6:63-7:16. Because the authorization A contains the random number, R, the authorization A, if intercepted on the insecure channel 11, cannot be reused for another request that uses a different random value. Hellman, 6:63-7:16. In this way, even when an authorization to use

Nintendo - Ancora Exh. 1003

a software package is transmitted over an insecure channel, the software manufacturer can be sure that the authorization A cannot be reused to allow for other, unauthorized uses of the software package.

69. Hellman depicts the operation of the base unit 12 during verification of authorization A in Figure 6, shown below. The base unit 12 provides the software package 17 as an input to one-way hash generator 33 to generate hash value H, the same hash value generated previously by the authorization and billing unit 13. Hellman, 9:16-28.



Nintendo - Ancora Exh. 1003

Hellman, Figure 6.

70. The base unit 12 has a key K stored in permanent memory 31. Hellman, 9:29-40. The permanent memory 31 can be "for example a PROM which was burned in during manufacture of the base unit." Hellman, 9:7-10. The key K can be the same as the secret key SK used by the authorization and billing unit 13. Hellman, 9:29-40.

71. The base unit has the values N and R stored in temporary memory 28. Hellman, 9:50-63. The temporary memory 28 can be "for example a RAM." Hellman, 8:67-68.

72. The base unit 12 operates a cryptographic check unit 34 in order to verify the authorization A. Hellman, 9:50-63. The base unit 12 provides the key K, the value N, the value R, the hash value H, and the received authorization A as inputs to the cryptographic check unit 34. Hellman, 9:50-63. Cryptographic check unit 34 determines that the authorization A is valid if the input of K, N, R, and H results in generation of the same authorization A. Hellman, 9:50-63.

73. If the base unit determines that the authorization A is valid, then the update unit 36 accesses non-volatile memory 37. Hellman, 9:64-10:13. The base unit retrieves the value stored in the memory address represented by hash value H. Hellman, 9:64-10:13. This retrieves the value M, which is the current number of authorized uses remaining. Hellman, 9:64-10:13. The base unit 12 then adds M to

Nintendo - Ancora Exh. 1003

the new number of authorized uses, N., and stores the new total number of authorized uses in non-volatile memory 37 at the address indicated by hash value H. Hellman, 9:64-10:13. The non-volatile memory 37 can be "for example an EEPROM or a CMOS memory with battery backup." Hellman, 9:64-10:13.

74. Hellman depicts the operation of the base unit 12 upon operation of software package 17 in Figure 6, shown below. When operation of the software package 17 is attempted, the software package 17 is provided as an input to the one-way hash function generator 33 in order to generate the hash value H. Hellman, 10:33-43. The update unit 36 uses the hash value H "as an address to non-volatile memory 37. Hellman, 10:33-43. The non-volatile memory 37 responds by providing the "signal representing M, the number of uses of software package 17 which are still available." Hellman, 10:33-43.

Nintendo - Ancora Exh. 1003

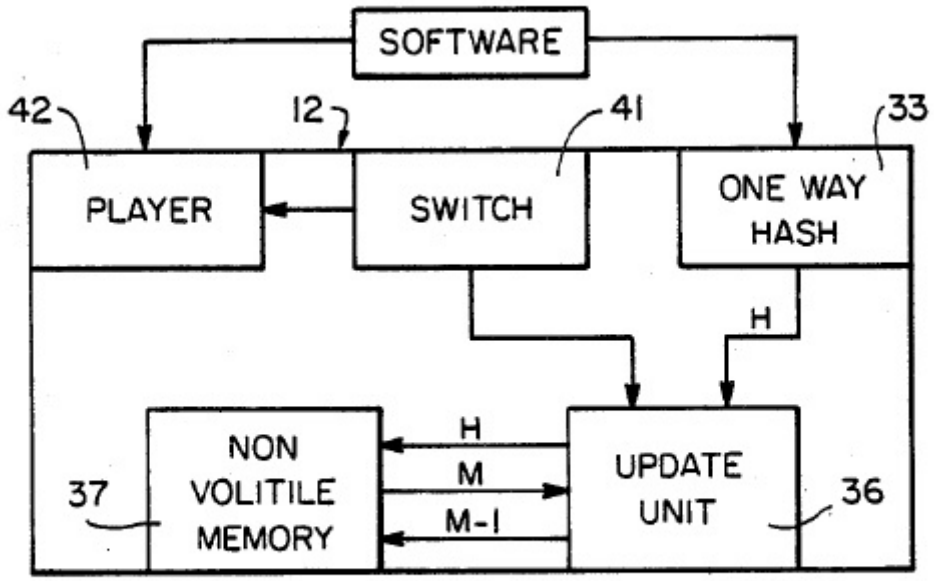


FIG_8

Hellman, Figure 8.

75. The update unit 36 then determines whether operation of the software package 17 will be permitted. Hellman, 10:44-54. If the value M received from the non-volatile memory 37 is greater than zero, then the update unit 36 uses the switch 41 which "activates software player 42, allowing it to use software package 17." Hellman, 10:44-54. The update unit 36 then decrements the value M, and "stores this as the new value in address H in non-volatile memory 37." Hellman, 10:44-54. If the value M retrieved from non-volatile memory 37 is zero, then operation of the software package 17 by the software player 42 is not permitted. Hellman, 10:44-54.

Nintendo - Ancora Exh. 1003

76. Hellman describes a variation to the above disclosure where the software package 17 is authorized for an "unlimited number of uses." Hellman, 10:55-65. This approach can be implemented "by reserving one value of M to represent infinity." Hellman, 10:55-65. For example, in an eight-bit value for M, the all 1's value, 255, can be reserved for "unlimited uses." Hellman, 10:55-65. "the update unit 36 would be designed to recognize the special pattern of all 1's and not change it when to software package was used." Hellman, 10:55-65.

77. The software player 42 "will vary from application to application." Hellman, 10:66-11:3. "[I]f the software is a computer program, then software player 42 would be a microprocessor or central processing unit (CPU)." Hellman, 10:66-11:3.

B. Chou

78. Chou's disclosure, which dates back to a 1996 filing date, describes an approach to discouraging computer theft. Chou, Abstract. A security routine is added to the BIOS memory. Chou, Abstract. The security routine looks for an externally connected memory device or checks for entry of a stored password. Chou, Abstract. Without passing this security check, the computer will not operate. Chou, Abstract. The computer can be changed between a mode that requires this security check and a mode that does not require the security check. Chou, Abstract.

Nintendo - Ancora Exh. 1003

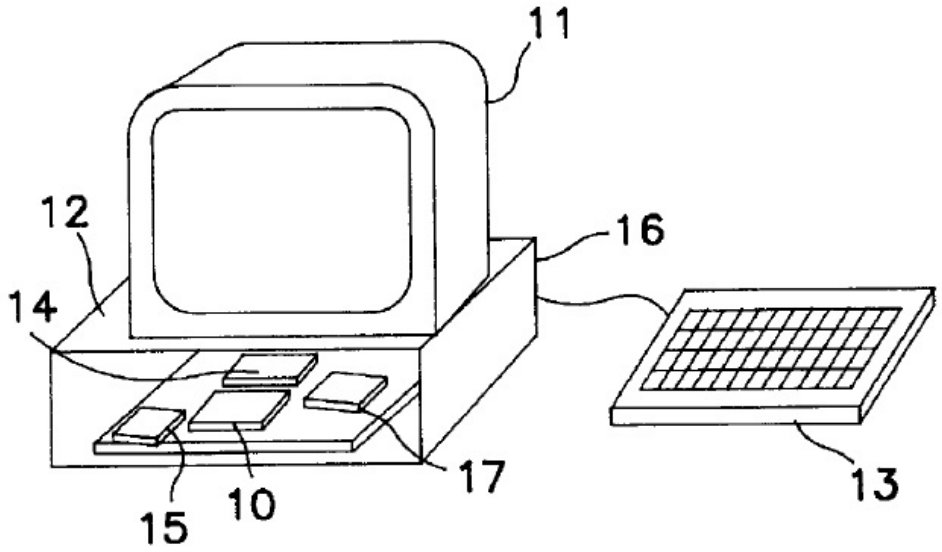
79. Chou explained that an important technological development that motivated the disclosed security techniques was a shift to using writable memory, such as EEPROM, as BIOS memory. Chou, 1:62-2:7. This created "the opportunity to provide password protection within the same memory which stores the BIOS routines." Chou, 1:62-2:7. Chou explained that storing this sensitive information like passwords in the BIOS memory provided a security benefit: "any attempt to delete the protection will result in the BIOS routine being disabled, disabling the boot up process." Chou, 1:62-2:7. In other words, Chou made the same observation as the '941 Patent, but several years earlier: by storing information with BIOS, the risk of rendering the entire computer inoperable discouraged tampering by all but the most advanced hacker. Chou, 1:62-2:7.

80. Chou disclosed that EEPROM memory was one advantageous way to implement this memory that comingled BIOS routines with other sensitive information. Chou, 2:2-7. In particular, EEPROM flash devices allowed the user to write data to the memory "without requiring the computer to be returned to the manufacture[r]." Chou, 2:2-7. Chou disclosed that its invention "makes use of these new BIOS memory devices effecting security measures which discourage theft." Chou, 2:2-7.

81. Chou disclosed a personal computer 10, as depicted in Figure 1, shown below. The computer 10 included a BIOS EEPROM 15 storing "the BIOS routines

Nintendo - Ancora Exh. 1003

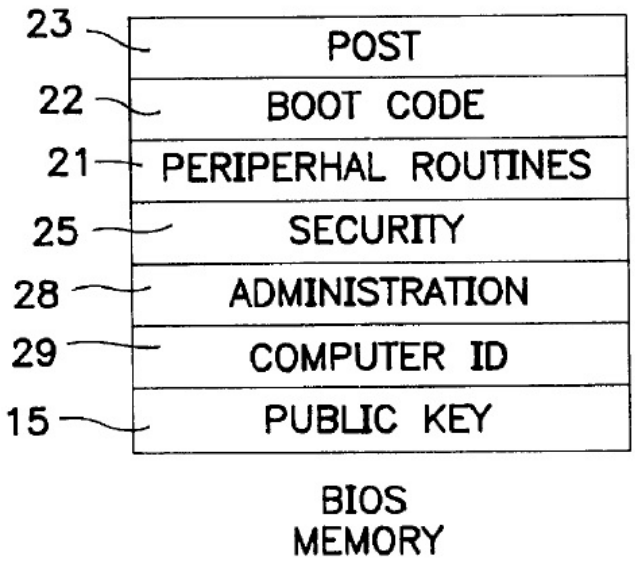
which provide for the basic input/output system." Chou, 3:21-28. The BIOS routines also "perform various functions, such as power-on self tests (POST), peripheral routines, boot codes, etc., for initially loading the computer operating system software." Chou, 3:44-48. The BIOS EEPROM 15 further stored "a security function stored as a programming routine." Chou, 3:21-28.



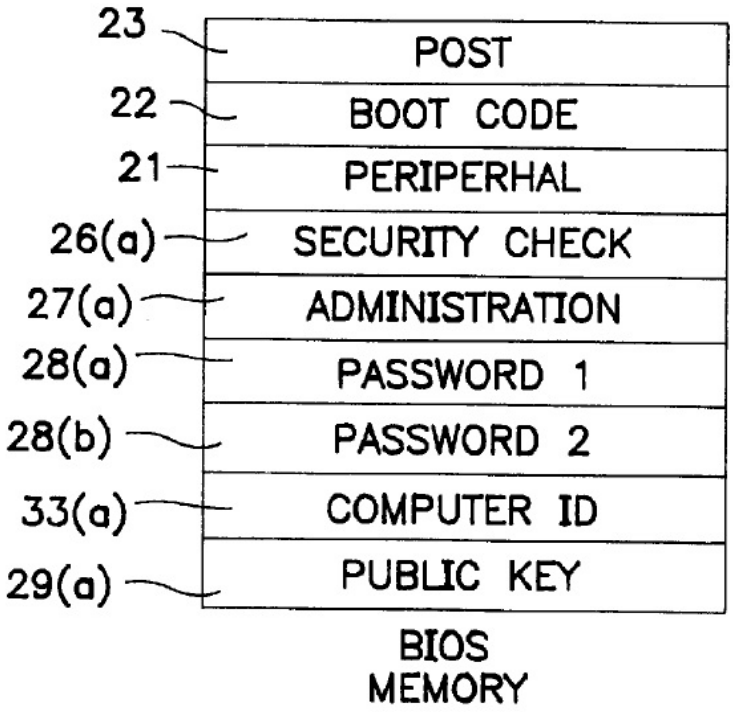
Chou, Figure 1.

82. Chou discloses various configurations for the BIOS memory in Figures 3 and 7, shown below. The BIOS memory "may be a flash EEPROM containing various executable BIOS routines as well as routines for implementing a security function." Chou, 3:51-67. The BIOS memory can include such BIOS routines as POST (power-on self test) routine 23, boot code 22, and "routine 21 for configuring peripheral devices connected to computer 10." Chou, 3:51-67.

Nintendo - Ancora Exh. 1003



Chou, Figure 3.



Chou, Figure 7.

Nintendo - Ancora Exh. 1003

83. The BIOS memory further includes the security routines 25 that implement the security techniques disclosed by Chou. Chou, 4:1-5. The security routine includes information to verify whether the user is an authorized user. Chou, Figure 3, 4:6-19. The BIOS memory stores further information for use in the security routines, such as a computer identifier (alternately referred to as 28, 29, and 33(a), a public key (alternately referred to as 15, 29, and 29(a)), and one or more passwords 28(a) / 28(b). Chou, Figure 3, Figure 7, 4:6-19, 7:14-35.

C. Schneck

84. Schneck's disclosure, which dates back to a 1997 filing date and a 1996 priority date, describes an approach to controlling access and distribution of digital data. Schneck, Abstract. Schneck explains that the transition from analog data to digital data greatly increased the ability to create unauthorized copies of data. Schneck, 1:40-2:67. Schneck referred to this issue as "secondary distribution" of the digital data. Schneck, 2:46-67.

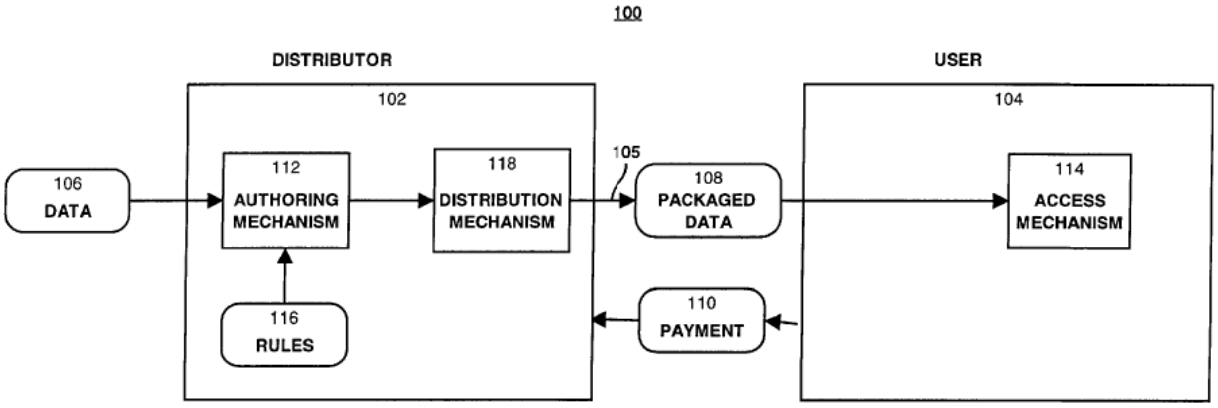
85. Schneck discloses that cryptography was the "principal technology" for protecting intellectual property. Schneck, 3:30-36. But Schneck observed a problem in the then-existing techniques: "Of those prior art systems which make some use of encryption, none protects the data after it has been decrypted. Thus, secondary distribution and multiple uses are possible." Schneck, 3:57-61. That is, Schneck observes that even if encryption is used to securely transfer digital data or to prevent

Nintendo - Ancora Exh. 1003

unauthorized access to digital data, once an authorized access is permitted, a user can make unauthorized copies. Schneck, 3:30-61.

86. Schneck describes that it would be best for control to the digital data to be controlled by the "originator," similar to the software package manufacturer in Hellman. Schneck, 3:66-67.

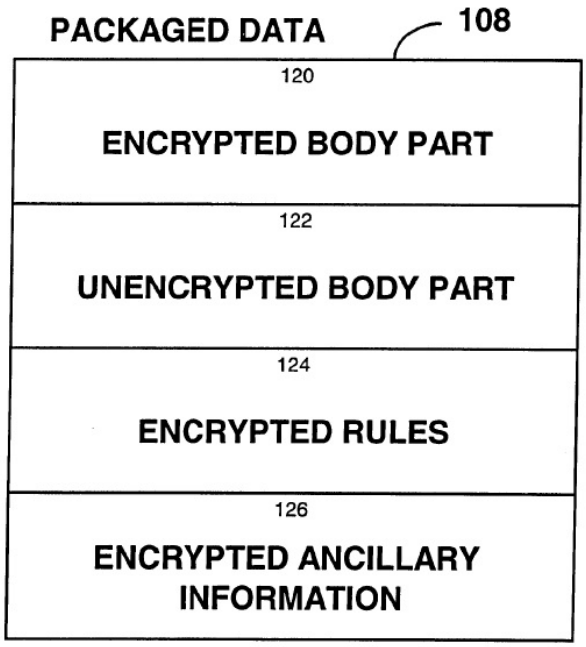
87. Schneck discloses the structure of a system 100 in Figure 1, shown below. The system includes a distributor 102 and a user 104. Schneck, 9:37-59. The distributor takes data 106 as input and uses the authoring mechanism 112 to create packaged data 108. Schneck, 9:37-59. "The packaged data 108 may include access rules 116 in encrypted form encoded therewith, or the access rules 116 may be provided to the user 104 separately." Schneck, 9:37-59. The distribution mechanism 118 distributes the package data 108 to the user. Schneck, 9:37-59.



Schneck, Figure 1.

Nintendo - Ancora Exh. 1003

88. Schneck discloses a structure of the packaged data 108 in Figure 2, shown below. The packaged data 108 can included both encrypted body part 120 and unencrypted body part 122. Schneck, 10:35-58. The two body parts are the digital content that the system is protecting (e.g., the data 106). Schneck, 10:35-58. The packaged data 108 also includes encrypted rules 124, which "are an encrypted version of access rules 116," and encrypted ancillary information 126. Schneck, 10:35-58. The access rules are encrypted using a "rule-encrypting key" that is "known only to (and protected within) each receiving computer of each user." Schneck, 12:1-16.



Schneck, Figure 2.

Nintendo - Ancora Exh. 1003

89. Schneck discloses a structure of the access rules 116 in Figure 3, shown below. The access rules "include various forms of validity checking and identification information." Schneck, 10:59-11:3. The access rules can include a license number 130. Schneck, 10:59-11:3. The access rules can include encrypted data key 138. Schneck, 10:59-11:3. The access rules can include "the actual rules 140, 142, 144-146 to be Is [sic] applied when access is made to the data by a user." Schneck, 10:59-11:3. The actual rules included various permissions and permission lists. Schneck, 10:59-11:3.

116
Version number **127**
Authentication (hash) **128**
License number of these rules **130**
Intellectual property identifier **132**
First valid generation of the product **134**
Last valid generation of the product **136**
Encrypted data key **138**
Standard permissions **140**
Extended permissions **142**
Custom permissions **144**
Co-requisite rules (permissions) for source data **145**
Token/biometrics **146**
System IDs/Public keys **147**

Schneck, Figure 3.

90. Scheck describes additional information for the content of access rules 116 in Table 1, shown below. The access rules 116 can include "System IDs/Public

Nintendo - Ancora Exh. 1003

keys" that serve the function of identifying "Other system to which these rules may be redistributed." Schenck, 11:32-35.

TABLE I

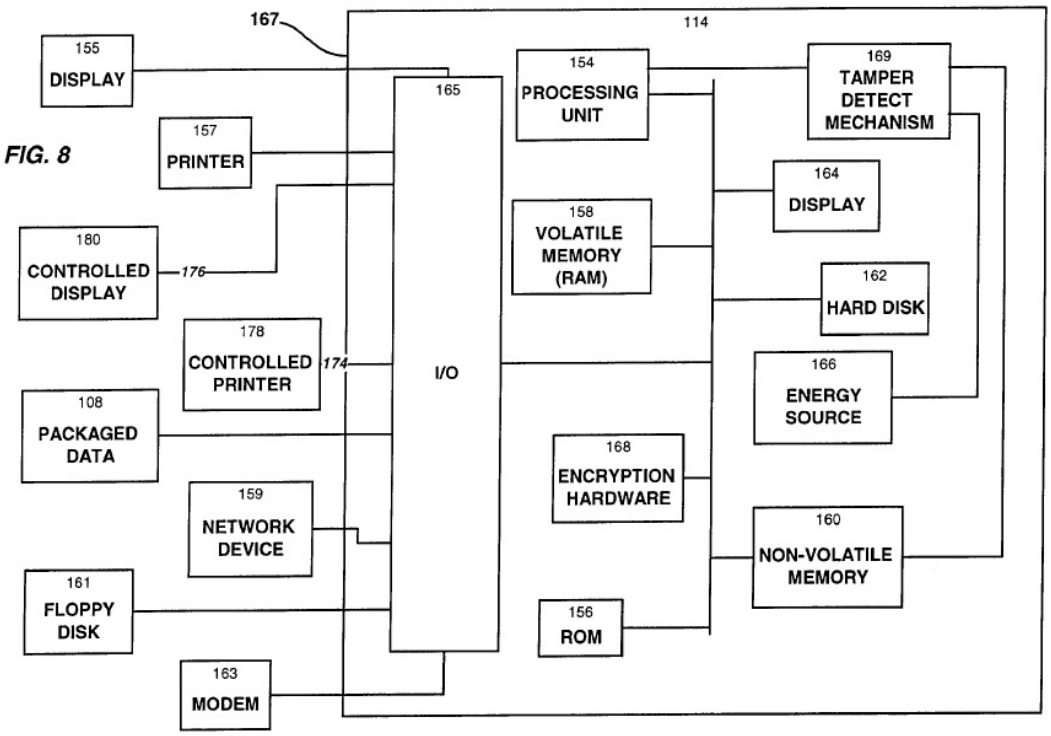
Field	Function
Version number 127	Defines internal configuration template
Authentication (hash) 128	Validates integrity of this data file.
License number of these rules 130.	Used by publisher to identify owner.
Intellectual property identifier 132.	Identifies the intellectual property product.
First valid generation of the product 134.	Defines extent of validity of the license.
Last valid generation of the product 136.	Defines extent of validity of the license.
Encrypted data key 138.	Key to access the data.
Standard permissions 140.	List of basic access permissions for data.
Extended permissions 142.	List of extended access permissions for data.
Custom permissions 144.	Executable code modules.
Co-requisite rules (permissions) for source data 145.	Indicates which source data rules are needed.
Token/biometrics 146	Indicates the physical tokens and/or biometric characteristics (if any) required for identification of each authorized user.
System IDs/Public keys 147	Other systems to which these rules may be redistributed.

91. As noted previously, Schneck allows for the encrypted rules 124 to be provided separate from the packaged data 108. Schneck, 9:51-54. Schneck describes such an embodiment with respect to Figure 5, where the packaged data 150 is transmitted to the user separately from the packaged rules 152. Schneck, 13:37-45. "The packaged data 150 without the rules has the form shown in Fig. 6,

Nintendo - Ancora Exh. 1003

which is essentially the same as the structure shown in Fig. 2, but without the encrypted rules 124." Schneck, 13:37-45.

92. Schneck discloses the structure of the user device with respect to Figure 8, shown below. "The access mechanism 114 allows a user 104 to access the data in packaged data 108 (or 150) according to the rules provided with (or separately from, as packaged rules 152) the packaged data and prevents the user or anyone else from accessing the data other than as described by the rules." Schneck, 15:20-29. Schneck notes that the access mechanism can alternatively be provided in a co-processor that operates in cooperation with an existing processing, as shown in Figure 9. Schenck 16:27-38.



Nintendo - Ancora Exh. 1003

Schneck, Figure 8.

93. The access mechanism 114 has a number of storage components, such as "volatile memory (RAM) 158," "electrically-alterable non-volatile memory 160," a hard disk 162, and read-only memory (ROM) 156. Schneck 15:30-38.

94. The access 114 mechanism includes a tamper detect mechanism 169. The purpose of the tamper detect mechanism 169 is to "allow[] the access mechanism 114 to ensure that all internal data (both the system's data and any user data) are destroyed before any tamperer can obtain them." Schneck, 16:16-19. When tampering with access mechanism 114 is detected, any cryptographic keys can be destroyed and memory devices can be cleared. Schneck 15:64-16:15.

95. Schneck disclosed that data stored in non-volatile memory should be stored in encrypted form in order to avoid unauthorized access. Schneck, 16:64-17:5, 17:6-12, 25:64-67. For example: "All communication between the components of the access mechanism 114 and the enclosed hard disk 162 is encrypted. Therefore, if the hard disk is removed the mechanism, any data stored thereon will be inaccessible without the appropriate keys." Schneck, 16:64-17:5. Further: "In general, within the system, the data are encrypted on any non-volatile storage devices so that they remain unavailable in the case of tampering." Schneck 17:6-12.

Nintendo - Ancora Exh. 1003

96. Schneck explains that this encrypting of the data on all non-volatile memory is important to prevent unauthorized secondary distribution: "Since all storage of data on internal non-volatile memory devices (for example, disks, flash memory, and the like) is encrypted, this ensures that a physical attack on the system will not result in compromise of plaintext." Schneck, 25:64-67.

IV. UNPATENTABILITY OF THE '941 PATENT CLAIMS

A. Standards for Invalidity

97. I am informed and understand that a patent cannot be properly granted for subject matter that would have been obvious to a person of ordinary skill in the art at the time of the alleged invention, and that a patent claim directed to such obvious subject matter is invalid under 35 U.S.C. § 103. It is also my understanding that in assessing the obviousness of claimed subject matter, one should evaluate obviousness in light of the prior art from the perspective of a person having ordinary skill in the art at the time the alleged invention was made (and not from the perspective of either a layman or a genius in that art). It is my further understanding that the question of obviousness is to be determined based on:

- The scope and content of the prior art;
- The difference or differences between the subject matter of the claim and the prior art (whereby in assessing the possibility of obviousness one should consider the manner in which a patentee and/or a Court has construed the scope of a claim);

Nintendo - Ancora Exh. 1003

- The level of ordinary skill in the art at the time of the alleged invention of the subject matter of the claim; and

- Any relevant objective factors (the "secondary indicia") indicating nonobviousness, including evidence of any of the following: commercial success of the products or methods covered by the patent claims; a long-felt need for the alleged invention; failed attempts by others to make the alleged invention; copying of the alleged invention by others in the field; unexpected results achieved by the alleged invention; praise of the alleged invention by the alleged infringer or others in the field; the taking of licenses under the patent by others and the nature of those licenses; expressions of surprise by experts and those skilled in the art at the subject matter of the claim; and whether the patentee proceeded contrary to accepted wisdom of the prior art.

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

B. Claim 1

1. Preamble: "A method of restricting software operation within a license for use with a computer including an erasable, non-volatile memory area of a BIOS of the computer, and a volatile memory area; the method comprising the steps of:"

98. I take no position as to whether this preamble is limiting on the claim, however, it is my opinion that Hellman discloses this feature other than the "erasable, non-volatile memory area of a BIOS." It is my opinion that Chou discloses the "erasable, non-volatile memory area of a BIOS," and that a POSA would have found it obvious to include the "erasable, non-volatile memory area of a BIOS" in Hellman's system.

Nintendo - Ancora Exh. 1003

99. Hellman discloses a "computer" in the form of the base unit 12. Hellman, Figure 1, Figure 6. Hellman indicates that the base unit 12 can be a computer: "the base unit (computer, videogame base unit, record player, videorecorder or video disk player)." Hellman, Abstract. Hellman also refers to "acquaintances with similar base units (computer)." Hellman, 2:24-29. Hellman discloses that the base unit 12 can have a software player 42 that can be "a microprocessor or central processing unit (CPU)." Hellman, 10:66-11:3. Based at least on these disclosures, a POSA would have recognized that the base unit 12 is a "computer."

100. Hellman discloses restricting software operation within a license. In particular, Hellman teaches restricting software operation within a license through the disclosure of the number of authorized uses value, M, which is later stored as a license record. Hellman describes M as "the number of authorized uses of the software package … which still remain unused prior to this new authorization." Hellman, 9:64-10:13. Hellman discloses that the value M is used to determine if operation of the software is permitted consistent with previous authorizations. Hellman, 9:64-10:13, 10:33-49. The value of M is increased based on payment for uses made by the user. Hellman, 9:64-10:13 (incrementing M with new authorized uses N), 5:57-6:15 (receiving authorization for N uses in exchange for billing information). Based at least on these disclosures, a POSA would have recognized

Nintendo - Ancora Exh. 1003

that the right to use the software package embodied in the value M reflects "restricting software operation within a license."

101. Hellman discloses "software" and "software operation" in the form of the software package 17 and its use by the software player 42. Hellman discloses a software package 17 that can be a "computer program." Hellman, 10:66-11:3. Hellman discloses activating software player 42, "allowing it to use software package 17." Hellman, 10:44-49. Hellman discloses that the software package 17 can be operated by software play 42, as a computer program being operated by "a microprocessor or central processing unit (CPU)." Hellman, 10:66-11:3. Based at least on these disclosures, a POSA would have recognized that software package 17 is "software" and that use of software package 17 by the software player 42 is "software operation."

102. Hellman further discloses a "method of restricting software operation within a license" in the form of restricting use of software package 17 by the software player 42 within the limits of authorized uses value, M. Hellman discloses that the base unit 12 only allows use of the software package 17 by the software player 42 if the value M is greater than zero, and thus more licensed uses are remaining. Hellman, 10:33-65.

103. Hellman discloses that the base unit 12 has a "volatile memory area" in the form of temporary memory 28. Hellman discloses that the base unit 12 has a

Nintendo - Ancora Exh. 1003

temporary memory 28. Hellman 8:66-67, Figure 6. Hellman discloses that the temporary memory 28 can be provided as RAM. Hellman 8:66-67. A POSA would have recognized that RAM (an acronym for Random Access Memory) is a type of volatile memory.

104. Hellman discloses that the base unit 12 has an "erasable, non-volatile memory area" in the form of non-volatile memory 37. Hellman discloses that the base unit has a non-volatile memory 37. Hellman, Figure 6, 9:64-10:13. Hellman discloses that the non-volatile memory 37 can be provided as "an EEPROM." Hellman, 9:64-10:13. A POSA would have recognized that EEPROM was an acronym for electrically erasable programmable read-only memory. Based at least on these disclosures, a POSA would have recognized that the non-volatile memory 37 was an "erasable, non-volatile memory area."

105. Hellman does not disclose that the non-volatile memory 37 is a BIOS, or that the base unit 12 included BIOS.

106. Chou disclosed a computer with BIOS and an "erasable, non-volatile memory area of a BIOS." Chou discloses a "BIOS EEPROM 15." Chou, Figure 1, 3:21-28. Chou discloses that the BIOS memory 15 "may be a flash EEPROM containing the various executable BIOS routines." Chou, 3:52-55. Chou discloses the "BIOS Memory" containing various BIOS routines. Chou, Figure 3, Figure 7. Chou disclosed that there had been a transition from using other types of memory

Nintendo - Ancora Exh. 1003

for BIOS to the use of EEPROM. Chou, 1:63-2:7. Based at least on these disclosures, a POSA would have recognized that the BIOS EEPROM 15 was an "erasable, non-volatile memory area of the BIOS."

107. A POSA would have found it obvious to include BIOS in the base unit 12 of Hellman. Hellman's disclosure does not describe BIOS routines, or a memory storing those routines. In my opinion, that lack of disclosure made sense in context, and was not based on any incompatibility between Hellman's base unit 12 and BIOS.

108. First, while BIOS existed in 1983, BIOS was ubiquitous in all computers at that time. A POSA would have recognized that BIOS was ubiquitous in computers by the time of the May 1998 priority date of the '941 Patent.

109. Second, while the "BIOS" terminology was used with respect to some general-purpose computers (e.g., microcomputers) in 1983, the terminology was not as consistently used for other types of electronic devices. And Hellman disclosed its invention generically for multiple types of "base units." For example, Hellman discloses that the base unit 12 can be any of a "computer," a "videogame base unit," a "record player," a "videorecorder," and a "videodisk player." Hellman, Abstract, 10:66-11:3. Because Hellman's disclosure was generic to these various types of electronic devices, it is understandable why Hellman did not describe BIOS routines and a BIOS memory for every embodiment. Furthermore, although many non-

Nintendo - Ancora Exh. 1003

computer devices at the time stored code in non-volatile memory with the structure and function of a BIOS, they did not generally use that term.

110. Based at least on the foregoing reasons, a POSA would have recognized that Hellman did not disclose BIOS or a BIOS memory in the base unit 12 due to the time of Hellman's disclosure and the generic nature of the base unit 12. A POSA would have recognized that the lack of a BIOS disclosure in Hellman was not due to any incompatibility of BIOS or BIOS memory with Hellman. In fact, the opposite. A POSA would have understood that at least Hellman's computer embodiment included a BIOS.

111. By the time of the May 1998 priority date of the '941 Patent, a POSA would have been aware that BIOS was ubiquitous in general purpose computers (e.g., personal computers). The applicant for the '941 Patent took an even stronger position during prosecution of the '941 Patent, saying that "Since all computer must have a BIOS …" Office Action dated February 5, 2002, page 7. As a result, when implementing Hellman's approach as of the May 1998 priority date of '941 Patent, a POSA would have found it obvious to implement Hellman's technique in a computer having a BIOS and thus a memory storing the BIOS.

112. When adding BIOS to the base unit 12 of Hellman, a POSA would have found it obvious to use the non-volatile memory 37 of Hellman to store the BIOS

Nintendo - Ancora Exh. 1003

routines. There are several independent reasons why a POSA would have been motivated to store the BIOS routines in the non-volatile memory 37 of Hellman.

113. First, a POSA would have been motivated to use non-volatile memory 37 of Hellman to store BIOS because non-volatile memory 37 used a type of memory disclosed by Chou as being advantageous for storing BIOS. Hellman disclosed that non-volatile memory 37 could be provided as EEPROM. Hellman, 9:64-10:13. Hellman disclosed other memory modules, such as permanent memory 31 and temporary memory 28, but Hellman did not disclose any other EEPROM modules. Hellman, 8:61-9:15. Chou disclosed that a transition had occurred to using EEPROM for BIOS, and that the programmability of EEPROM made that an advantageous medium for storing BIOS. Chou, 1:63-2:7. Based at least on those disclosures, a POSA, when adding BIOS to the base unit 12 of Hellman, would have been motivated to store the BIOS in the non-volatile memory 37 (i.e., EEPROM) of Hellman.

114. Second, a POSA would have been motivated to use non-volatile memory 37 of Hellman to store BIOS because non-volatile memory 37 would have been one of a limited number of design choices. Hellman disclosed that non-volatile memory 37 could be provided as EEPROM. Hellman, 9:64-10:13. Hellman disclosed other memory modules, such as permanent memory 31 and temporary memory 28, but Hellman did not disclose any other EEPROM modules. Hellman,

Nintendo - Ancora Exh. 1003

8:61-9:15. A POSA would have recognized that in a typical computer as of the May 1998 priority date of the '941 Patent, there would have been a limited number of EEPROM modules available. EEPROM modules were a specialized type of memory not used for general purpose storage. As such, they were generally only added to a computer for a special purpose use. As of May 1998, a typical computer would have had less than five such EEPROM modules, and perhaps only one. As such, the EEPROM module embodied in non-volatile memory 37 of Hellman would have been one of a limited number of choices for where to store the BIOS routines. It would have been easy and obvious to choose non-volatile memory 37 from among this limited set of options. In many computers, the BIOS EEPROM would be the only EEPROM module in the computer.

115. Third, a POSA would have been motivated to use non-volatile memory 37 of Hellman to store BIOS because Chou disclosed that storing sensitive information with the BIOS routines provided extra protection to that sensitive information. Chou disclosed that it was beneficial to store sensitive information like passwords in the BIOS memory because "any attempt to delete the protection will result in the BIOS routine being disabled, disabling the boot up process." Chou, 1:63-2:7. A POSA would have understood this disclosure to mean that information should be stored within the BIOS memory because tampering with that information would risk disabling the entire computer. A POSA would have recognized that this

Nintendo - Ancora Exh. 1003

would have discouraged many users from attempting to tamper with that information in the first place. Hellman disclosed storing sensitive information, the number of authorized uses M, in non-volatile memory 37. Hellman, 9:63-10:13. Thus, a POSA would have been motivated to store BIOS together with the values M in the non-volatile memory 37, in order to discourage users from tampering with the values M. Chou, 1:63-2:7.

116. Fourth, a POSA would have been motivated to use non-volatile memory 37 of Hellman to store BIOS because that would have been the most efficient cost and space implementation. As of the May 1998 priority date of the '941 Patent, and still to this day, computer manufacturers generally tried to implement functionality with little cost as possible and in as small of a form factor as possible. A POSA would have recognized that storing BIOS routines in the already existing non-volatile memory 37 of the base unit 12, as opposed to adding another memory module, would have reduced cost and minimized the physical space occupied by the memory modules.

117. A POSA would have had a strong expectation of success in storing BIOS in the non-volatile memory 37 for numerous reasons.

118. First, Chou already describes storing BIOS in EEPROM, which was one of the formats disclosed for use for non-volatile memory 37. Hellman, 9:64-10:13; Chou, 1:63-2:7, 3:21-28, 3:62-67.

Nintendo - Ancora Exh. 1003

119. Second, Chou already disclosed that add-on information, such as passwords, encryption keys, and security modules could be added to the memory space used by the BIOS. Chou, 3:62-4:5, Figure 3, Figure 7. A POSA would have recognized that license information, such as the value M or any other license information, could easily be stored within that BIOS memory space instead.

120. Third, a POSA would have recognized that EEPROM modules of the time had sufficient space to store both the authorization values M from Hellman (which is disclosed in some embodiments as being only 8 bits) and the BIOS routines from Chou. Chou implicitly disclosed this in that it disclosed storing add-on information and security modules in a single EEPROM with BIOS routines. Chou, 3:62-4:5, Figure 3, Figure 7. A POSA would have recognized that there were numerous sizes of EEPROM modules available in May 1998 which would have had sufficient space to store the BIOS routines and the authorization values from Hellman.

120A. A POSA would not have been discouraged from the above-described motivations to combine and strong expectation of success based on any alleged incompatibility between the teachings of Hellman and Chou.

120B. As an initial matter, a POSA would have been encouraged by Chou itself to store sensitive information, like Hellman's licensing information, in the BIOS memory. Chou, 1:63-2:7. While there would have been some risk introduced

Nintendo - Ancora Exh. 1003

by storing non-BIOS information in BIOS memory, that increased risk is what Chou observed as the benefit of doing so in the first place: preventing tampering with the sensitive information without also impacting the BIOS data and thus disabling the entire device. Chou, 1:63-2:7.

120C. Hellman's use of a hash value as a memory address would not have negated this motivation provided by Chou. With Hellman's approach of using a hash value as a memory address, there already would have been some risk of duplicate uses of memory addresses. Namely, with Hellman's approach, one value (i.e., a numerical representation of the software package content itself) was mapped into a second data space (i.e., the maximum range of memory addresses allocated for the purposes of storing the license information in non-volatile memory 37) using one-way hash function generator 33. A POSA would have recognized that the former value (the software package contents) would have been a larger data space than the latter value (the range of memory addresses). Hellman, 6:48-53. Hence, while perhaps unlikely, it would have been possible for two different software packages to map to hash to the same hash value H and thus the same memory address. This is a fundamental property of hash tables, but in practice is made statistically unlikely enough to be tolerable or insignificant.

120D. Based at least on the foregoing observation, a POSA would have recognized that an implementation of Hellman's approach would have had to

Nintendo - Ancora Exh. 1003

account for potential duplicate uses of memory locations. Namely, if two software packages hashed to the same hash value H, Hellman's approach would have need to account for the fact that both would by default have stored license information in the same memory location, one overwriting the other. With that understanding in mind, a POSA would have likewise recognized that an implementation of Hellman's approach where non-volatile memory 37 was the BIOS memory (in light of Chou's disclosure, as discussed above) would also have had to account for potential duplicate uses of memory locations by both BIOS data and a hash value H. A POSA would not have been discouraged by these observations from making the modification of Hellman based on Chou. Rather a POSA would have understood these to be the sort of ordinary and minute implementation details required by any computer implementation, including, as just discussed, an implementation of Hellman's unmodified disclosure.

120E. As for how a POSA would have avoided duplicate use of a memory location by both BIOS data and a hash value H, there are likely endless such possibilities. As one example, a POSA would have recognized that the one-way hash function generator 33 of Hellman could have been modified to map to a specific range of memory values. Hellman already disclosed this ability, because the one-way hash function generator 33 would already have to be mapping to the range of acceptable memory addresses in non-volatile memory 37. With the combination of

Nintendo - Ancora Exh. 1003

Hellman and Chou, a POSA would have found it trivial to modify one-way hash function generator 33 to a different memory address range where the BIOS data was not being stored. For instance, in Chou shows in Figure 3 that a memory space 25 labelled "Security" is allocated in the BIOS Memory 15 in the BIOS space for storing the new security function software disclosed by Chou. A POSA would have recognized that this memory space--potentially enlarged if needed--could be reused for mapping the has values H from the one-way hash function generator 33. In that way, the license information from Hellman would be stored amidst the BIOS data in non-volatile memory 37, but without risk of the license data overwriting the BIOS data.

120F. There are likely countless other ways that a POSA would have found it reasonable to implement Hellman as modified by Chou to avoid duplicate use of a memory location by both BIOS data and Hellman's license data. This is precisely the sort of routine and ordinary design and implementation process that a POSA would be accustomed to performing.

2. Element 1.a: "selecting a program residing in the volatile memory"

121. It is my opinion that Hellman discloses this feature and alternatively renders it obvious based on knowledge of skill in the art. It is my opinion that Schneck further demonstrates that a POSA would have found this feature obvious.

Nintendo - Ancora Exh. 1003

122. Hellman discloses a program in the form of software package 17. Hellman, 5:51-56, 10:50-54, 10:66-11:3. Hellman discloses a volatile memory in the form of temporary memory 28. Hellman, 8:61-9:15.

123. Hellman discloses numerous ways in which the software package 17 is selected. Hellman either discloses explicitly, discloses implicitly, or renders obvious that this selection would be of the software package 17 in the temporary memory 28.

124. First, Hellman discloses that the software package 17 is selected in the form of purchasing the software package 17 "at a store, over telephone line, or in some similar manner." Hellman, 5:51-56. A POSA would have recognized that as part of loading the purchased software package 17, the software package 17 would have been loaded into temporary memory 28 (e.g., RAM), such as part of transferring the software package 17 from the medium on which it was purchased into some other storage device used by the base unit 12 for storage.

125. Second, Hellman discloses that the software package 17 is selected when the base unit 12 is generating a request for software use. Hellman, 5:57-6:2. As part of generating the request for software use, the base unit 12 determines a software name for the software package 17. Hellman, 5:57-63. A POSA would have recognized that one way in which to determine the software name would have been to load the software package 17 into RAM (temporary memory 28), and extract the software name from the contents of the software package 17.

Nintendo - Ancora Exh. 1003

126. Third, Hellman discloses that the software package 17 is selected when the software package 17 is input into the one-way hash function generator 33. Hellman, 9:16-28, 9:50-63, 10:33-49. The software package 17 is provided as an input to the one-way hash function generator 33 in order to generate the hash value H. Hellman, 9:16-28, 9:50-63, 10:33-49. A POSA would have recognized that the software package 17 would have been present in RAM (temporary memory 28) when it was input to the one-way hash function generator 33. A POSA would have recognized that the one-way hash function generator 33 could have been implemented as either a hardware or a software module. At least in the case where one-way hash function generator 33 was a software module, a POSA would have recognized that a standard way to provide software package 17 as input to one-way hash function generator 33 would have been to first load software package 17 into RAM, and then provide it as input to the one-way hash function generator 33.

127. Fourth, Hellman discloses that the software package 17 is selected when the software package 17 is selected for use by the software player 42. Hellman, 10:33-11:3. Hellman discloses that software player 42 can "use software package 17." Hellman, 10:44-49. Hellman discloses that when software package 17 is a "computer program," the software player 42 is "a microprocessor or central processing unit (CPU)." Hellman, 10:66-11:3. A POSA would have recognized that in this context, the typical way for a software program to be used by a

Nintendo - Ancora Exh. 1003

microprocessor or CPU would be to first load it into RAM (temporary memory 28), and then to select it for execution by the microprocessor or CPU.

128. Fifth, a POSA would have found it obvious to modify Hellman so that the software package 17 was maintained in RAM (temporary memory 28) between the time that it was selected for extraction of the software name and the time when it was selected for verifying the authorization A. More specifically, as part of generating the request for software use, the base unit 12 determines a software name for the software package 17. Hellman, 5:57-63. A POSA would have recognized that one way in which to determine the software name would have been to load the software package 17 into RAM (temporary memory 28), and to extract the software name from the contents of the software package 17. Thereafter, the same software package 17 is provided as an input to the one-way hash function generator 33 in order to generate the hash value H. Hellman, 9:16-28, 9:50-63.

129. A POSA would have recognized that the time between sending the request for software use and the validation of the returned authorization could be a short period of time, e.g., a number of seconds or minutes. A POSA would have recognized that it would have been efficient to maintain the software package 17 in RAM (temporary memory 28) between these two events in order to avoid having to re-load the software package 17 into RAM (temporary memory 28) twice. With such a modification, the software package 17 would still have been in the temporary

Nintendo - Ancora Exh. 1003

memory 28 when it is selected for providing as input to one-way hash function generator 33.

130. While this feature was disclosed by or alternatively rendered obvious by Hellman, the feature would further have been obvious based on the teachings of Schneck.

131. Schneck discloses that a first step to accessing protected information would be to "request the operating system to read such data into memory." Schneck at 18:7–10. The protected information, such as software package 17 from Hellman, would then be selected for any further operations while it was residing in volatile memory (temporary memory 28). Schneck at 18:7–10. Thus, even if Hellman did not explicitly disclose that the software package 17 was first loaded into temporary memory 28 prior to any of the occasions on which it was selected.

132. A POSA would have been motivated to modify the base unit 12 to first load the software package 17 into temporary memory 28 for several reasons. First, Schneck teaches that this was a standard approach for operating a protected software program, and thus a POSA would have been motivated to use a known and reliable method for operating on software package 17. Second, a POSA would have been motivated to first load the software package 17 into temporary memory 28 because a POSA would have recognized temporary memory 28 as one of a limited number of design choices. In fact, other than temporary memory 28, Hellman does not

Nintendo - Ancora Exh. 1003

explicitly describe any other memory devices in the base unit 12 where the software package 17 could be stored during the selecting activities described above. Third, a POSA would have been motivated to first load the software package 17 into temporary memory 28 because a POSA would have recognized that it would have been faster to perform the various selecting activities described above if the software package 17 were present in temporary memory 28 (e.g., RAM), as opposed to some other, slower storage medium.

3. Element 1.b: "using an agent to set up a verification structure in the erasable, non-volatile memory of the BIOS, the verification structure accommodating data that includes at least one license record"

133. It is my opinion that, based on the modification of Hellman to use non-volatile memory 37 as the "erasable, non-volatile memory of the BIOS" as taught by Chou (discussed above), Hellman discloses this feature. Furthermore, a POSA would have found it obvious to modify Hellman based on the teachings of Schneck to store the number of authorized uses in encrypted form, which is relevant to certain dependent claims.

134. Hellman discloses a "license record" in the form of the number of authorized uses value M. Hellman describes M as "the number of authorized uses of the software package … which still remain unused prior to this new authorization." Hellman, 9:64-10:13. Hellman discloses that the value M is used to determine if

Nintendo - Ancora Exh. 1003

operation of the software is permitted consistent with previous authorizations. Hellman, 9:64-10:13, 10:33-49. The value of M is increased based on payment for uses made by the user. Hellman, 9:64-10:13 (incrementing M with new authorized uses N), 5:57-6:15 (receiving authorization for N uses in exchange for billing information). In at least some embodiments, M represents unlimited authorized use. Based at least on these disclosures, a POSA would have recognized that the right to use the software package embodied in the value M is a "license record."

135. Hellman discloses a "verification structure" in the form of the memory structure of non-volatile memory 37 storing at least one value M at memory addresses defined by at least one hash value H. Hellman discloses that hash value H is "an 'abbreviation' or name for describing the software package 21," which is an "exact replica" of software package 17. Hellman, 6:16-61. Hellman discloses that hash value H has the characteristic that "it is easily com[]puted from its input signal, software package 21, but given an H value it is difficult, taking perhaps millions of years, to compute any other software package w[h]ich produces this same H value." Hellman, 6:16-61. Hellman discloses that H is used as an "interrogatory signal" to the non-volatile memory 37, and that update unit 36 uses H "as an address to non-volatile memory 37." Hellman, 9:64-10:13, 10:33-43.

136. Based at least on these disclosures, a POSA would have recognized that update unit 36 sets up a structure of memory addresses defined by hash value H for

Nintendo - Ancora Exh. 1003

storing authorized use values M in the non-volatile memory 37. And because the stored authorized use value M is used to verify if operation of software package 17 is permitted, a POSA would have recognized that this memory structure is a verification structure.

137. Hellman discloses an "agent" in the form of update unit 36. As described above, update unit 36 stores the authorized use value M in the non-volatile memory 37. Hellman, 9:64-10:13. Update unit 36 retrieves the authorized use value M from the non-volatile memory 37. Hellman, 10:33-43. Update unit 36 updates the value M in the non-volatile memory 37. Hellman, 10:44-49. While Hellman does not specifically disclose how update unit 36 is implemented, a POSA would have recognized that the update unit 36 would have been implemented by a software routine, potentially along with a hardware module. Based at least on these disclosures, a POSA would have recognized that the update unit 36 is an agent, and that it sets up the verification structure in the non-volatile memory 37.

137A. A POSA would have recognized that the update unit 36 would have been implemented by software, hardware, or some combination of the two. Hellman does not explicitly say whether the update unit 36 should be implemented in software, hardware, or a combination of the two. A POSA would have recognized from this lack of discussion that it was not necessary that one type of implementation be used over another. In other words, a POSA would have understood that it was up to the

Nintendo - Ancora Exh. 1003

discretion of the implementer whether to use software, hardware, or a combination of the two.

137B. This understanding would have been confirmed by the fact that the activities performed by the update unit 36 were of a type that could be performed in software, hardware, or both. The update unit 36 retrieves a value stored at a location in EEPROM, performs integer addition and/or subtraction, and transmits a value to be stored at a location in EEPROM. Hellman, 9:64-10:13. These are all tasks that a POSA would have understood could be implemented in software, hardware, or both. A POSA would have been motivated to implement the update unit 36 in software in particular because that would have allowed the provider of the base unit to change the implementation logic of the update unit 36 over time, without having to physically disassemble, modify, and reassemble the base unit.

138. A POSA would additionally have recognized that the authorization and billing unit 13 may cooperate with the update unit 36 to act as the "agent." Hellman discloses that the authorization and billing unit 13 stores a table of serial numbers and secrets keys in memory 18. Hellman, 6:16-30. The authorization and billing unit 13 uses the serial number received from the base unit 12 to determine the secret key, SK, for the base unit 12. Hellman, 6:16-30. The authorization and billing unit 13 stores a table of software in memory 19 that allows it to determine a software package 21 from the software name provided in the request, and software package

Nintendo - Ancora Exh. 1003

21 is identical to software package 17. Hellman, 6:16-30. Because authorization and billing unit 13 generates the authorization A that leads to the updating of the authorized use value M in the non-volatile memory 37, a POSA would have recognized that the authorization and billing unit 13 may be considered an agent.

138A. A POSA would have recognized that the authorization and billing unit 13 would have been implemented by software, hardware, or some combination of the two. Hellman does not explicitly say whether the authorization and billing unit 13 should be implemented in software, hardware, or a combination of the two. A POSA would have recognized from this lack of discussion that it was not necessary that one type of implementation be used over another. In other words, a POSA would have understood that it was up to the discretion of the implementer whether to use software, hardware, or a combination of the two.

138B. This understanding would have been confirmed by the fact that the activities performed by the authorization and billing unit 13 were of a type that could be performed in software, hardware, or both. The authorization and billing unit 13 stores a table of serial numbers and secret keys, stores a table of information about software packages, performs a hash function, and performs a cryptographic function. Hellman, 6:16-7:16. These are all tasks that a POSA would have understood could be implemented in software, hardware, or both. A POSA would have recognized that the maintaining of the two lookup tables in particular would have been the type

Nintendo - Ancora Exh. 1003

of functionality typically involving a software implementation. A POSA would have been motivated to implement the authorization and billing unit 13 in software in particular because that would have allowed the provider of the authorization and billing unit 13 to change the implementation logic of the authorization and billing unit 13 over time, without having to physically disassemble, modify, and reassemble the authorization and billing unit 13.

139. Furthermore, a POSA would have found it obvious to modify Hellman based on the teachings of Schneck to store the number of authorized uses in encrypted form, which is relevant to certain dependent claims.

140. As an initial observation, Hellman and Schneck attempt to solve a similar problem: preventing an authorized user from distributing unauthorized copies of licensed software. Hellman at 1:39–2:53; Schneck at 2:40–67. Hellman calls this "copy protection," while Schneck calls this "secondary distribution." Hellman at 1:39–2:53; Schneck at 2:40–67.

141. While both Hellman and Schenck attempt to address this same problem, Hellman only effectively deals with one aspect of it. Hellman's approach prevents the interception and reuse of the authorization signal A. Hellman at 6:62–7:16. This thereby protects against the risk of a malicious tampering with respect to the insecure channel 11 between the base unit 12 and the authorization and billing unit 13. Hellman at 6:62–7:16. Hellman achieves this result because the authorization A is

Nintendo - Ancora Exh. 1003

encrypted with a key, SK, specific to the base unit 12 and a cryptographic hash specific to the software package 17. Hellman at 6:62–7:16. As a result, the authorization A cannot be reused for a different software package 17 or on a different base unit 12. Hellman at 6:62–7:16.

142. However, a POSA would have recognized that Hellman's approach does not protect against tampering once the authorized use value M is stored in the non-volatile memory 37 of the base unit 12. This is the case because M is stored simply as an integer value, i.e., in plaintext, in the non-volatile memory 37. Hellman at 9:64–10:13. And, the value M is stored at an address determined by hash value H. Hellman at 9:64–10:13. A POSA would have recognized that, as a result, for a software package 17 present on the base unit 12, a malicious user could generate the hash value H, interrogate the non-volatile memory 37, and thereby retrieve the authorized use value M. A POSA could then write a new authorized use value M to the memory address for hash value H, thereby granting new, unauthorized uses to the software package 17. A POSA would have recognized that this tampering was possible because M was stored as a plaintext numerical value, and thus could be overwritten without any need for cryptographic verification.

143. A POSA would have recognized that this risk was heightened for the situation where the number of authorized uses was the special value indicating an "unlimited number of uses of a software package." Hellman, 10:55-65. Hellman

Nintendo - Ancora Exh. 1003

disclosed that a special value of M could be used to indicate infinite uses of the software package 17. Hellman, 10:55-65. Hellman suggests an "all 1's pattern" for this special value. Hellman, 10:55-65. But, a POSA would have recognized that, to the extent that a malicious user does not know in advance what the special value is, the malicious user could interrogate the non-volatile memory 37 as described above and thereby discover this special infinite use value. A POSA would have recognized that this special value of M could then be stored at the memory address for any software package 17, and thereby grant unlimited uses that were not paid for.

144. Given that Hellman had a shortcoming in its technique for preventing unauthorized secondary distribution, a POSA would have found it obvious to modify Hellman to correct that shortcoming. Schneck discloses one technique that a POSA would have found obvious for correcting this shortcoming of Hellman.

145. Like Hellman, Schneck disclosed that the authorization information, authorization A in Hellman and access rules 116 in Schneck, were transmitted to the user device in encrypted form. Schneck at 9:46–59. But Schneck also disclosed that information that arrives at the user device should be stored in encrypted form, which is the cause of Hellman's shortcoming. Schneck, 16:64-17:5, 17:6-12, 25:64-67. Schneck discloses that, because "all storage of data on internal non-volatile memory devices (for example, disks, flash memory, and the like) is encrypted, this ensures

Nintendo - Ancora Exh. 1003

that a physical attack on the system will not result in compromise of plaintext." Schneck, 25:64-67.

146. In light of this disclosure of Schneck, a POSA would have found it obvious to modify Hellman to store the number of authorized uses value M in encrypted form in non-volatile memory 37. A POSA would have recognized that storing M in encrypted form would prevent the sort of tampering described above and warned against by Schneck.

147. In considering how to store M in encrypted form, one technique that a POSA would have found obvious would have been to store the authorization A in non-volatile memory 37 at memory address H. A POSA would have considered this as a desirable solution because base unit 12 already receives authorization A in encrypted form, and thus base unit 12 could simply store the authorization A without the need for additional decrypting/encrypting beyond that already disclosed in Hellman.

148. A POSA would have recognized that storing authorization A in non-volatile memory 37 could have been especially beneficial where the special value M for an unlimited number of uses was used. Hellman, 10:55-65. With the unlimited uses value, the value M never needs to be incremented or decremented. Hellman, 10:55-65. As such, a POSA would have recognized that each time the software package 17 is used, the update unit 36 could simply perform the verification of the

Nintendo - Ancora Exh. 1003

value M instead of having to modify M, re-encrypt the value, and then update the storage at address H in non-volatile memory 37.

149. A POSA would have recognized that storing the authorization A in the non-volatile memory 37 would have corrected the vulnerability in Hellman described above. First, because authorization A is encrypted with a key, K, unique to the base unit 12, a malicious user could not copy the authorization A into the non-volatile memory 37 of some other base unit 12 and still have it enable use of software package 17 on that other base unit 12. Second, because authorization A is encrypted with hash value H unique to the software package 17, a malicious user could not copy the authorization A into some other memory address H on the same base unit 12 and have it enable use of the other software package 17.

150. A POSA would have recognized that Hellman's system, when modified based on the teachings of Schneck as described above would achieve the objective of both references of preventing unauthorized secondary distribution of software.

4. Element 1.c: "verifying the program using at least the verification structure from the erasable non-volatile memory of the BIOS, and"

151. It is my opinion that Hellman discloses this feature. As discussed above, Hellman disclosed: the "program" in the form of software package 17; the "verification structure" in the form of the structure of non-volatile memory 37 defined by hash values H storing authorized use values M; and the "erasable non-

Nintendo - Ancora Exh. 1003

volatile memory of the BIOS" in the form of the non-volatile memory 37, as modified by the teachings of Chou to store BIOS routines.

152. Hellman discloses verifying the software package 17 using the verification structure stored in non-volatile memory 37. Hellman discloses that, when use of the software package 17 is attempted, the base unit generates the hash value H. Hellman, 10:33-54. The base unit then uses hash value H as a memory address in non-volatile memory 37 in order to retrieve the number of authorized uses M for that software package 17. Hellman, 10:33-54. The base unit 12 then checks whether the authorized use value M is greater than zero in order to verify whether operation of the software package 17 is permitted. Hellman, 10:33-54. Based at least on these disclosures, a POSA would have recognized that Hellman discloses this feature.

153. When Hellman was modified by the teachings of Schneck to store authorization A in the non-volatile memory 37, a POSA would have recognized that verification would have been performed using the authorization A. A POSA would have recognized that one way in which the verification could be performed would be to decrypt the authorization A using the unique key K of the base unit in order to retrieve the value M included therein. The value M could then be used as already described in Hellman. Hellman, 10:33-54. A POSA would have recognized that,

Nintendo - Ancora Exh. 1003

because the authorization A was encrypted, the authorization A would have to be decrypted in order to retrieve the number of authorized uses value M.

5. Element 1.d: "acting on the program according to the verification."

154. It is my opinion that Hellman discloses this feature. As discussed above, Hellman disclosed: the "program" in the form of software package 17. Hellman disclosed "the verification" in the form of the verification process described for claim element 1.c.

155. Hellman discloses acting on the program according to the verification in the form of allowing use of the software package 17 by the software player 42 only if the number of authorized uses is greater than zero. Hellman, 10:44-65. If the value of M is greater than zero, then the update unit 36 sends a control signal to the switch 41 to allow software player 42 to use software package 17. Hellman, 10:44-49. If the value of M is zero, then the update unit 36 does not activate the switch to allow the software player 42 to use software package 17. Hellman, 10:50-54. "The user is thus prevented from using software for which he does not have current authorized use." Hellman, 10:50-54.

C. Claim 2: "A method according to claim 1, further comprising the steps of: establishing a license authentication bureau."

156. It is my opinion that Hellman discloses this feature.

Nintendo - Ancora Exh. 1003

157. Hellman discloses establishing a "license authentication bureau" in the form of establishing authorization and billing unit 13. Hellman discloses that authentication and billing unit generates an authorization A. Hellman, 6:3-8. Hellman refers to the authorization A as an "authenticator": "The software manufacturer generates an authenticator which is a cryptographic function of the base unit's key, the software, the number of times use of the software is authorized, and the random number generated by the base unit." Hellman, 4:46-63. As discussed previously, a POSA would have recognized that the number of authorized uses is a license because it defines the scope of use of software package 17 that is permitted. Hellman, 6:3-15, 9:64-10:13.

158. As such, the authorization and billing unit 13 generates an authenticator for a license of a software package by encrypting license content for the software package using an encryption key unique to the base unit 12. Based at least on these disclosures, a POSA would have recognized that Hellman discloses "establishing a license authentication bureau."

D. Claim 3

1. Preamble: "A method according to claim 2, wherein setting up a verification structure further comprising the steps of:"

159. It is my opinion that Hellman discloses this feature, as described for claim 2 above.

Nintendo - Ancora Exh. 1003

2. Element 3.a: "establishing, between the computer and the bureau, a two-way data-communications linkage;"

160. It is my opinion that Hellman discloses this feature.

161. As discussed above, Hellman disclosed: the "computer" in the form of base unit 12; and the "bureau" in the form of the authorization and billing unit 13.

162. Hellman disclosed "establishing … a two-way data communications linkage" between the base unit 12 and the authorization and billing unit 13 in the form of insecure communication channel 11. Hellman, 5:39-50. Hellman discloses communication over insecure communication channel 11 in one direction when the base unit 12 transmits the request for software use to the authorization and billing unit 13. Hellman, 5:57-6:2. Hellman discloses communication insecure communication channel 11 in the other direction when the authorization and billing unit 13 transmits the authorization A to the base unit 12. Hellman, 6:3-15.

3. Element 3.b: "transferring, from the computer to the bureau, a request-for-license including an identification of the computer and the license-record's contents from the selected program;"

163. It is my opinion that Hellman discloses this feature. Furthermore, a POSA would have found it obvious to modify Hellman based on the teachings of Schneck to include additional information in the request-for-license beyond what Hellman discloses.

Nintendo - Ancora Exh. 1003

164. Hellman discloses a "request-for-license" in the form of the "user originated request for software use." Hellman, 5:57-6:2.

165. Hellman discloses "transferring, from the computer to the bureau" the request for software use, in the form of the base unit 12 transmitting the request for software use to the authorization and billing unit 13. Hellman, 5:57-6:2.

166. Hellman discloses that the request for software use includes "an identification of the computer" in the form of the serial number for the base unit 12. Hellman, 5:57-6:2. Hellman discloses that the request for software use includes a "serial number." Hellman, 5:57-6:2. Hellman discloses that the serial number can be "a serial number, identification number, user name or similar identifier unique to base unit 12." Hellman, 5:57-6:2.

167. A POSA would have found it obvious to modify Hellman to use a public key for the base unit 12 in the request for software as the "serial number." Hellman disclosed that the serial number could be any "serial number, identification number, user name or similar identifier unique to base unit 12." Hellman, 5:57-6:2. Schneck disclosed that "System IDs/Public keys" can be used to identify a device. Schneck 11:32-34. A POSA would have been motivated to use a public key in Hellman's system because Hellman already disclosed that a public key cryptographic system was compatible with Hellman's system. Hellman, 11:20-68.

Nintendo - Ancora Exh. 1003

168. A POSA would have recognized that the public key could be transmitted from the base unit 12 to the authorization and billing unit 13 as part of the request for software use in order to identify the base unit 12, as suggested by Schneck. Schneck 11:32-34. A POSA would have recognized the additional benefit of including the public key in the request for software use in order to allow the authorization and billing unit 13 to generate the authorization A without needing to maintain secret keys for the base units 12 in memory of the authorization and billing unit 13. Hellman, 6:16-30. A POSA would have recognized that this would have improved the efficiency and security of the authorization and billing unit 13.

169. Hellman discloses that the request for software use includes "the license-record's contents from the selected program" in the form of the software name and the number of request uses value N. Hellman, 5:57-6:2. Hellman discloses that the request for software use includes the "software name," which is "the name of the software package to be used." Hellman, 5:57-6:2. Hellman discloses that the request for software use includes the value N, which is "the number of additional uses of software requested." Hellman, 5:57-6:2.

170. At least where the request for software use is the first request for software use for the software package 17, the value N is the value that is ultimately stored in the non-volatile memory 37 as the license record. A POSA would have recognized that when the first request for software use is transmitted, the existing

Nintendo - Ancora Exh. 1003

value M for the number of authorized uses is zero. When the base unit 12 receives the authorization A, which contains the transmitted value N, it is the value N that is stored as the value M in the non-volatile memory 37. Hellman, 9:64-10:13. This is so because "M+N" is equal to the value N when the preexisting value of M is zero. Hellman, 9:64-10:13.

171. A POSA would have found it obvious to modify Hellman to use the hash value H in the request for software use instead of the "software name." Hellman discloses that the software name "allows authorization and billing unit 13 to determine the complete contents of software package 17 from knowledge of the much smaller information software name." Hellman, 6:16-30. A POSA would have recognized that in some circumstances, a software name may not be sufficiently unique or well-structured to unique identify the software package 17. For example, if multiple software manufacturers share an authorization and billing unit 13, then there would be a risk that two software manufacturers could inadvertently use the same software name for two different software packages.

172. Based on the risk that the software name would not uniquely identify a software package, a POSA would have been motivated to use the hash value H instead. Hellman discloses that hash value H is an "'abbreviation' or name for describing the software package." Hellman, 6:31-61. As such, Hellman already indicates that hash value H could be used as a shortened identifier for the software

Nintendo - Ancora Exh. 1003

package 17. As such, a POSA would have recognized that the hash value H could serve as an identifier of the software package 17 and would be more likely to uniquely identify the software package 17 than would the software name. With such a modification, the billing and authorization unit could maintain a table correlating hash value H to software package 17, instead of software name to software package 17. Hellman, 6:16-30.

4. Element 3.c: "forming an encrypted license-record at the bureau by encrypting parts of the request-for-license using part of the identification as an encryption key;"

173. It is my opinion that Hellman discloses this feature other than use of "part of the identification as an encryption key." It is my opinion that a POSA would have found it obvious to include this feature in Hellman's system based Schneck's teachings.

174. Hellman discloses forming an "encrypted license-record" in the form of authorization A. As discussed previously for element 1.b, authorization A is a license record. Authorization A is encrypted. Hellman, 6:62-7:2.

175. As discussed previously for claim 2, the authorization and billing unit 13 is a "bureau." Hellman discloses forming the authorization A at the "bureau" in the form of forming authorization A at the authorization and billing unit 13. Hellman, 6:62-7:2.

Nintendo - Ancora Exh. 1003

176. Hellman discloses encrypting "parts of the request-for-license" to form authorization A in the form of using the value N to form the authorization A. Hellman, 6:62-7:2. The value N is included in the request for software use. Hellman, 5:57-6:2. The value N is encrypted as part of the authorization A. Hellman, 6:62-7:2.

177. As discussed above for element 3.b, a POSA would have found it obvious to include the hash value H in the request for software use. Hellman discloses that the hash value H is included in the authorization A. Hellman 6:62-7:2. In this additional way, a POSA would have found it obvious to encrypt "parts of the request-for-license" to form authorization A.

178. As discussed above for element 3.b, a POSA would have found it obvious to use a public key for the base unit 12 as the "identification of the computer" in the request for software use. As discussed above for element 3.b, a POSA would have found it obvious to make this modification because it would allow the authorization and billing unit 13 to use the public key as the encryption key for forming authorization A instead of a locally-maintained secret key. For at least those reasons, a POSA would have found it obvious to use "part of the identification as an encryption key."

Nintendo - Ancora Exh. 1003

5. Element 3.d: "transferring, from the bureau to the computer, the encrypted license-record; and"

179. It is my opinion that Hellman discloses this feature. As discussed above, Hellman disclosed: the "bureau" in the form of authorization and billing unit 13; the "computer" in the form of base unit 12; and the "encrypted license-record" in the form of authorization A.

180. Hellman discloses transferring the encrypted license-record from the bureau to the computer in the form of transmitting the authorization A from the authorization and billing unit 13 to the base unit 12. Hellman, 6:3-15.

6. Element 3.e: "storing the encrypted license record in the erasable non-volatile memory area of the BIOS."

181. It is my opinion that, based on the modification of Hellman to use non-volatile memory 37 as the "erasable, non-volatile memory of the BIOS" as taught by Chou (discussed for claim 1 preamble), and based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), Hellman discloses this feature.

182. Hellman disclosed storing the value M in the non-volatile memory 37. Hellman, 9:64-10:13. As discussed for claim 1 preamble, a POSA would have found it obvious to use non-volatile memory 37 as the erasable, non-volatile memory of the BIOS.

Nintendo - Ancora Exh. 1003

183. Hellman disclosed storing the value M in the non-volatile memory 37. As discussed for element 1.b, a POSA would have found it obvious to store the authorization A in the non-volatile memory 37 instead of the plaintext value M.

184. Based on those modification to Hellman discussed previously, a POSA would have found it obvious to store "the encrypted license record in the erasable non-volatile memory area of the BIOS."

E. Claim 6: "A method according to claim 1 wherein selecting a program includes the steps of: establishing a licensed-software-program in the volatile memory of the computer wherein said licensed-software-program includes contents used to form the license-record."

185. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), Hellman discloses this feature.

186. As discussed above, Hellman disclosed: the "computer" in the form of base unit 12; the "volatile memory" in the form of temporary memory 28; a "licensed-software-program" in the form of software package 17; and the "license-record" in the form of authorized uses value M.

187. As discussed for element 1.a, Hellman disclosed loading the software package 17 in the temporary memory 28. Based at least on these disclosures, a POSA would have recognized that Hellman discloses "establishing a licensed-software-program in the volatile memory of the computer."

Nintendo - Ancora Exh. 1003

188. As discussed for element 1.b, a POSA would have found it obvious to store the authorization A in the non-volatile memory 37 instead of the plaintext value M. Hellman discloses that the software package 17 includes contents used to form the authorization A. Hellman discloses that the entirety of the software package 17 is used to generate the hash value H. Hellman, 6:31-61, 9:16-28. The hash value H is part of the authorization A. Hellman, 6:62-7:2. Based at least on these disclosures, a POSA would have recognized that Hellman discloses that the "licensed-software-program includes contents used to form the license-record."

F. Claim 7

1. Preamble: "A method according to claim 6 wherein using an agent to set up the verification structure includes the steps of:"

189. It is my opinion that Hellman discloses this feature, as described for claim 6 above.

2. Element 7.a: "establishing or certifying the existence of a pseudo-unique key in a first non-volatile memory area of the computer; and"

190. It is my opinion that Hellman discloses this feature

191. As discussed above, Hellman disclosed: the "computer" in the form of base unit 12.

192. Hellman discloses "a first non-volatile memory area" in the form of permanent memory 31. Hellman discloses that base unit 12 includes a "permanent

Nintendo - Ancora Exh. 1003

memory 31, for example a PROM." Hellman 8:61-9:15. A POSA would have recognized that PROM is an acronym for programmable read-only memory, and that PROM was a non-volatile memory.

193. Hellman discloses a "pseudo-unique key" in the permanent memory 31 in the form of key K stored in permanent memory 31. Hellman, 9:29-40. Hellman discloses that the key K can be the same as the secret key SK. Hellman, 9:29-40. Hellman discloses that "no two users share the same secret key." Hellman, 9:41-45.

194. Hellman discloses "certifying the existence" of the key K in the permanent memory 31 in the form of retrieving the key K for input to the cryptographic check unit 34 in order to verify the validity of authorization A. Hellman, 9:50-63, 9:64-10:13, Figure 6.

195. Based at least on these disclosures, a POSA would have recognized that Hellman discloses "certifying the existence of a pseudo-unique key in a first non-volatile memory area of the computer."

196. As an additional note, the '941 Patent says that in "the context of the present invention, a 'pseudo-unique' key may relate to a bit string which uniquely identifies each first non-volatile memory. Alternately the 'pseudo-unique' key may relate to a random bit string (or to an assigned bit string) of sufficient length such that: there is an acceptably low probability of a successful unauthorized transfer of licensed software between two computers, where the first volatile

Nintendo - Ancora Exh. 1003

memories of these two computers have the same key." '941 Patent, 4:10-18. Because Hellman discloses that the key K is unique amongst base units 12, a POSA would recognize that the key K is a pseudo-unique key as recited in this claim.

3. Element 7.b: "establishing at least one license-record location in the first nonvolatile memory area or in the erasable, non-volatile memory area of the BIOS."

197. It is my opinion that Hellman discloses this feature

198. As discussed above, Hellman disclosed: the "non-volatile memory area of the BIOS" in the form of non-volatile memory 37, as modified by Chou to also store BIOS routines.

199. Hellman discloses "establishing at least one license-record location" in the non-volatile memory 37 in the form of establishing a memory address defined by hash value H where a license record (number of authorized uses M, or authorization A) for the correspond software package 17 will be stored. Hellman, 9:64-10:13. Hellman discloses that a license record for the software package 17 will be stored at a single, specific memory address: the memory address that corresponds to hash value H. Hellman, 9:64-10:13. And hash value H is generated based on the content of the software package itself. Hellman, 31-61, 9:16-28. As such, a POSA would have recognized that Hellman discloses establishing a license record location in the non-volatile memory 37.

Nintendo - Ancora Exh. 1003

200. Based at least on these disclosures, a POSA would have recognized that Hellman discloses "establishing at least one license-record location in the first nonvolatile memory area or in the erasable, non-volatile memory area of the BIOS."

G. Claim 8

1. Preamble: "A method according to claim 6 wherein establishing a license-record includes the steps of:"

201. It is my opinion that Hellman discloses this feature, as described for claim 6 above.

2. Element 8.a: "forming a license-record by encrypting of the contents used to form a license-record with other predetermined data contents, using the key; and"

202. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), Hellman discloses this feature.

203. As discussed above, Hellman disclosed: a "license-record" in the form of authorization A, as modified to Schneck to store authorization A in non-volatile memory 37; the "contents used to form a license-record" in the form of the software package 17 and its corresponding hash value H; and a "key" in the form of key SK.

204. Hellman discloses forming the authorization A by encrypting the "contents used to form a license-record … using the key" in the form of encrypting

Nintendo - Ancora Exh. 1003

the hash value H with other contents using the key SK (which can be the same as key K). Hellman, 6:62-7:2, 9:29-40.

205. Hellman discloses encrypting "other predetermined data contents" in the license record in the form of number of requested uses N and random number R. Hellman, 6:62-7:2, 9:50-63.

206. Based at least on these disclosures, a POSA would have recognized that Hellman as modified by Schneck discloses "forming a license-record by encrypting of the contents used to form a license-record with other predetermined data contents, using the key."

3. Element 8.b: "establishing the encrypted license-record in one of the at least one established license-record locations."

207. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), Hellman discloses this feature.

208. As discussed above, Hellman disclosed: an "encrypted license-record" in the form of authorization A, as modified to Schneck to store authorization A in non-volatile memory 37; and the "established license-record locations" in the form the memory addresses defined by a hash value H.

209. Hellman discloses establishing the number of authorized uses value M in the memory location defined by hash value H. Hellman, 9:64-10:13. As discussed

Nintendo - Ancora Exh. 1003

above for element 1.b, a POSA would have found it obvious to store the authorization A in the memory location defined by hash value H, instead of the value M. Based at least on these disclosures, a POSA would have recognized that Hellman as modified by Schneck discloses "establishing the encrypted license-record in one of the at least one established license-record locations."

H. Claim 9

1. Preamble: "A method according to claim 7 wherein verifying the program includes the steps of:"

It is my opinion that Hellman discloses this feature, as described for claim 7 above.

2. Element 9.a: "encrypting the licensed-software-program's license-record contents from the volatile memory area or decrypting the license-record in the erasable, non-volatile memory area of the BIOS, using the pseudo-unique key; and"

210. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), this feature would have been an obvious modification of Hellman.

211. As discussed above, Hellman disclosed: the "license-record" in the form of authorization A, as modified to Schneck to store authorization A in non-volatile memory 37; the "non-volatile memory area of the BIOS" in the form of the

Nintendo - Ancora Exh. 1003

non-volatile memory 37, as modified by Chou to store the BIOS routines; and the "pseudo-unique key" in the form of the key K.

212. As discussed above for element 1.c, when Hellman was modified by the teachings of Schneck to store authorization A in the non-volatile memory 37, a POSA would have recognized that verification would have been performed using the authorization A. A POSA would have recognized that one way in which the verification could be performed would be to decrypt the authorization A using the unique key K for the base unit 12 in order to retrieve the value M included therein. The value M could then be used as already described in Hellman. Hellman, 10:33-54. A POSA would have recognized that, because the authorization A was encrypted, the authorization A would have to be decrypted in order to retrieve the number of authorized uses value M.

3. Element 9.b: "comparing the encrypted licenses-software-program's license-record contents with the encrypted license-record in the erasable, non-volatile memory area of the BIOS, or comparing the license-software-program's license-record contents with the decrypted license-record in erasable non-volatile memory area of the BIOS."

213. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), a POSA would have found this feature to be an obvious modification of Hellman.

Nintendo - Ancora Exh. 1003

214. As discussed above, Hellman disclosed: the "license-software-program's license-record contents" in the form of the software package 17 and its corresponding hash value H; the "license-record" in the form of authorization A, as modified to Schneck to store authorization A in non-volatile memory 37; and the "non-volatile memory area of the BIOS" in the form of the non-volatile memory 37, as modified by Chou to store the BIOS routines.

215. As discussed above for element 9.a, when Hellman was modified by the teachings of Schneck to store authorization A in the non-volatile memory 37, a POSA would have found it obvious to decrypt the authorization A from the non-volatile memory 37 as part of verification of the software package 17.

216. A POSA would have found it obvious to compare the decrypted authorization A with the software package 17 and the corresponding hash value H. As discussed above for element 1.b, one of the reasons that a POSA would have been motivated to modify Hellman to store the authorization A instead of the number of authorized uses value M in the non-volatile memory 37 was the fact that storing plaintext value M would allow a malicious user to change the value of M for any software package 17. Further, as discussed above for element 1.b, a POSA would have recognized that storing authorization A would have prevented this tampering, because the hash value H was part of the authorization A, and as such the

Nintendo - Ancora Exh. 1003

authorization A could not be reused at any other memory location in the non-volatile memory 37.

217. A POSA would have found it obvious that in order to achieve this benefit in the modification of Hellman, it would be necessary to compare the hash value H included as part of the authorization A to the software package 17 that the user was attempting to use. If the base unit 12 did not perform this comparison of the hash value H in the authorization A to the hash value H for the software package 17 in use, then the tampering described above would not be prevented. As such, a POSA would have found it obvious to implement this comparison in the modified system of Hellman, and thereby include "comparing the license-software-program's license-record contents with the decrypted license-record in erasable non-volatile memory area of the BIOS."

I. Claim 10: "A method according to claim 9 wherein acting on the program includes the step: restricting the program's operation with predetermined limitations if the comparing yields non-unity or insufficiency."

218. It is my opinion that Hellman discloses this feature. As discussed above, Hellman disclosed: the "program" in the form of software package 17.

219. Hellman discloses restricting the operation of the software package 17 in the form of preventing the user from using the software package 17 when the number of authorized uses equals zero. Hellman, 10:50-54.

Nintendo - Ancora Exh. 1003

220. As discussed above for element 9.b, a POSA would have found it obvious to compare the hash value H from the authorization A stored in the non-volatile memory 37 to the hash value H for the software package 17 that the user is attempting to use. A POSA would have recognized that when those hash values H do not match, the authorization A does not correspond to the software package 17 that the user is attempting to use. In such a case, a POSA would have found it obvious to prevent the user from using the software package 17. Hellman, 10:50-54. A POSA would have been motivated to prevent this operation in order to prevent the user from making unauthorized use of the software package 17.

J. Claim 11: "A method according to claim 1 wherein the volatile memory is a RAM."

221. It is my opinion that Hellman discloses this feature. As discussed above for claim 1 preamble, Hellman disclosed the "volatile memory" in the form of temporary memory 28. Hellman disclosed that temporary memory could be "for example a RAM." Hellman, 8:67-68.

K. Claim 12: "The method of claim 1, wherein a pseudo-unique key is stored in the non-volatile memory of the BIOS."

222. It is my opinion that this feature would have been an obvious modification of Hellman based on the teachings of Schneck. As discussed above, Hellman disclosed: the "non-volatile memory of the BIOS" in the form non-volatile memory 37, as modified by Chou to store BIOS routines.

Nintendo - Ancora Exh. 1003

223. A POSA would have found it obvious to modify Hellman to use a public key for the base unit 12 in the request for software as the "serial number." Hellman disclosed that the serial number could be any "serial number, identification number, user name or similar identifier unique to base unit 12." Hellman, 5:57-6:2. Schneck disclosed that "System IDs/Public keys" can be used to identify a device. Schneck 11:32-34. A POSA would have been motivated to use a public key in Hellman's system because Hellman already disclosed that a public key cryptographic system was compatible with Hellman's system. Hellman, 11:20-68.

224. A POSA would have recognized that the public key could be transmitted from the base unit 12 to the authorization and billing unit 13 as part of the request for software use in order to identify the base unit 12, as suggested by Schneck. Schneck 11:32-34. A POSA would have recognized the additional benefit of including the public key in the request for software use in order to allow the authorization and billing unit 13 to generate the authorization A without needing to maintain secret keys for the base units 12 in memory of the authorization and billing unit 13. Hellman, 6:16-30. A POSA would have recognized that this would have improved the efficiency and security of the authorization and billing unit 13.

225. A POSA would have recognized that the public key would be a "pseudo-unique key." Schneck disclosed that "System IDs/Public keys" can be used to identify a device. Schneck 11:32-34. Based on this disclosure of Schneck and

Nintendo - Ancora Exh. 1003

the knowledge of public key cryptography in the art as of May 1998, a POSA would have recognized that the public key would be able to identify a particular base unit 12.

226. When Hellman was modified to use a public key, a POSA would have recognized that the public key could be stored in the non-volatile memory 37. A POSA would have recognized that the public key, like any other key disclosed in Hellman, would need to be stored in non-volatile memory in order to be maintained when power was lost. Hellman, 9:16-49. Hellman only disclosed two such non-volatile memories: permanent memory 31 and non-volatile memory 37. Hellman, 8:61-9:15, Figure 6. A POSA would have found it obvious to store the public key in the non-volatile memory 37 as one of a limited number of design choices, i.e., one of two options.

227. Additionally, a POSA ould have found it obvious to store the public key in the non-volatile memory 37 in order to allow that value to change. Hellman disclosed that the key K was stored in permanent memory 31 because it would not change. Hellman, 9:16-49. But a POSA would have recognized that it may be beneficial to change a key periodically, including a public key, such as to prevent a brute force attack attempting to determine the private key being used by the base unit 12. A POSA would recognize that, in order for the public key to be able to change, it would be beneficial to public key in the non-volatile memory 37.

Nintendo - Ancora Exh. 1003

228. A POSA would have further found it obvious to store the public key in the non-volatile memory 37 because Chou discloses that a public key can be stored in the BIOS memory. Chou, 4:6-19, Figure 3, Figure 7. And, as discussed above for claim 1 preamble, a POSA would have found it obvious to use the non-volatile memory 37 as the BIOS memory.

L. Claim 13: "The method of claim 1, wherein a unique key is stored in a first non-volatile memory area of the computer."

229. It is my opinion that Hellman discloses this feature. As discussed above, Hellman disclosed: the "computer" in the form of base unit 12.

230. Hellman discloses "a first non-volatile memory area" in the form of permanent memory 31. Hellman discloses that base unit 12 includes a "permanent memory 31, for example a PROM." Hellman 8:61-9:15. A POSA would have recognized that PROM is an acronym for programmable read-only memory, and that PROM was a non-volatile memory.

231. Hellman discloses a "unique key" in the permanent memory 31 in the form of key K stored in permanent memory 31. Hellman, 9:29-40. Hellman discloses that the key K can be the same as the secret key SK. Hellman, 9:29-40. Hellman discloses that "not two users share the same secret key." Hellman, 9:41-45.

Nintendo - Ancora Exh. 1003

M. Claim 14: "The method according claim 13, wherein the step of using the agent to set up the verification record, including the license record, includes encrypting a license record data in the program using at least the unique key."

232. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), Hellman discloses this feature.

233. As discussed above, Hellman disclosed: the "agent" in the form of the update unit 36 in cooperation with authorization and billing unit 13; a "verification record, including the license record" in the form of the authorization A, as modified to Schneck to store authorization A stored at the specific memory location defined by hash value H in the non-volatile memory 37; the "program" in the form of software package 17; and the "unique key" in the form of the key K, which can be the same as the secret key SK.

234. Hellman discloses that the software package 17 includes data used to form the authorization A. Hellman discloses that the entirety of the software package 17 is used to generate the hash value H. Hellman, 6:31-61, 9:16-28. The hash value H is part of the authorization A. Hellman, 6:62-7:2. Hellman discloses that the authorization A is encrypted using the secret key SK, which can be the same as the key K. Hellman, 6:62-7:2, 9:29-40. Based at least on these disclosures, a

Nintendo - Ancora Exh. 1003

POSA would have recognized that Hellman discloses that the "encrypting a license record data in the program using at least the unique key."

N. Claim 16: "The method according to claim 13, wherein the step of verifying the program includes a decrypting the license record data accommodated in the erasable second non-volatile memory area of the BIOS using at least the unique key."

235. It is my opinion that, based on the modification of Hellman to store the authorization A instead of the plaintext value M (discussed for element 1.b), this feature would have been an obvious modification of Hellman.

236. As discussed above, Hellman disclosed: the "license-record" in the form of authorization A, as modified to Schneck to store authorization A in non-volatile memory 37; an "erasable, non-volatile memory area of the BIOS" in the form of the non-volatile memory 37, as modified by Chou to store the BIOS routines; and the "unique key" in the form of the key K. A POSA would have recognized that the non-volatile memory 37, an "erasable, non-volatile memory area of the BIOS" could also be an erasable second non-volatile memory area of the BIOS.

237. As discussed above for element 1.c, when Hellman was modified by the teachings of Schneck to store authorization A in the non-volatile memory 37, a POSA would have recognized that verification would have been performed using the authorization A. A POSA would have recognized that one way in which the verification could be performed would be to decrypt the authorization A using the

Nintendo - Ancora Exh. 1003

unique key K for the base unit 12 in order to retrieve the value M included therein. The value M could then be used as already described in Hellman. Hellman, 10:33-54. A POSA would have recognized that, because the authorization A was encrypted, the authorization A would have to be decrypted in order to retrieve the number of authorized uses value M.

238. I have considered the impact of secondary indicia in forming my opinions. I am unaware of any secondary indicia at this time that would impact my conclusions that these claims are obvious in view of the recited art.

239. I declare under the penalty of perjury that all statements made in this Declaration are true and correct.

Executed on 8-9-2021 in Los Gatos, CA.

Andrew Wolfe

Nintendo - Ancora Exh. 1003

Appendix B

B-1 (Ex. 1001 in the Petition) U.S. Patent No. 6,411,941 to Mullor et al. ("'941 Patent")

B-2 (Ex. 1002 in the Petition) Image File Wrapper of U.S. Patent No. 6,411,941 ("File History")

B-3 (Ex. 1004 in the Petition) U.S. Patent No. 4,658,093 ("Hellman")

B-4 (Ex. 1005 in the Petition) U.S. Patent No. 5,892,906 ("Chou")

B-5 (Ex. 1006 in the Petition) U.S. Patent No. 5,933,498 ("Schneck")

B-6 European patent Application EP 0766165A2 ("'165 Application")

B-7 U.S. Patent 5,724,425 ("'425 Patent")

B-8 U.S. Patent 6,138,236 ("'236 Patent")

B-9 U.S. Patent 5,802,592 ("'592 Patent")

B-10 U.S. Patent 5,835,594 ("'594 Patent")

Nintendo - Ancora Exh. 1003

APPENDIX A

Andrew Wolfe Ph.D.

20 S. Santa Cruz Ave. Suite 101
Los Gatos, CA 95030
(408) 402-5872 (office) (408) 394-1096 (mobile)
Email: awolfe@awolfe.org

Education:

Ph.D. in Computer Engineering, Carnegie Mellon University, 1992
Visiting Graduate Student, Center for Reliable Computing, Stanford University, 1988-1989
M.S. in Electrical and Computer Engineering, Carnegie Mellon University, 1987
B.S.E.E. in Electrical Engineering and Computer Science, The Johns Hopkins University, 1985

Recent Employment:

Consultant, [October 2002-present]
Wolfe Consulting

Consultant on processor technology, computer systems, consumer electronics, software, design tools, and intellectual property issues. Testifying and consulting expert for IP and other technology-related litigation matters.

Sample clients include:

AMD	Nvidia	Samsung
IBM	Motorola	HTC
SMIC	AMKOR	Huawei
Dell	Honeywell	Western Digital
Nintendo	Kingston	Sonos
Moneygram	Arraycomm	Insilica
Synaptics	Activision	Sawstop
Mysticom	P.A.R.C.	Quester Ventures

Lecturer, [September 2013-present]
Santa Clara University

Teaching graduate and undergraduate courses on embedded computing, mechatronics and computer architecture.

Chief Technical Officer, [1999-2002]; Sr. VP of Business Development, [2001-2002]; VP, Systems Integration, S3 Fellow , [1998 – 1999]; Director of Technology, S3 Fellow , [1997 - 1998]
SONIC|blue, Inc, Santa Clara, CA (formerly S3 Inc.)

Strategic Business Development:
Developed and implemented strategy to reposition S3 from PC graphics into the leading networked consumer electronics company.

- Acquired Diamond Multimedia and coordinated integration of communications, Rio digital music, and workstation graphics divisions into S3.
- Identified and negotiated acquisitions to grow digital media businesses including Empeg, ReplayTV, and Sensory Science.
- Identified and negotiated strategic investments including Comsilica, Intellon, KBGear Interactive, Entridia, DataPlay and others.
- Developed strategy for integrated graphics/core-logic products and established a joint venture with Via Technologies to design and market these products.
- Negotiated divestiture of graphics chip business to Via and the workstation graphics division to ATI.

Product Planning and Development:

- Drove roadmap development within SONICblue product divisions.
- Managed Business Development for all product lines.
- Led New Product Development and Corporate Vision processes.
- Acting co-General Manager of Rio digital music business in 2nd half of 2001. Responsible for all areas of product development, business development, and cost management.
- Managed development of the Savage/MX and Savage/IX mobile 3D graphics accelerators and Savage/NB system logic products.

Public Relations, Public Policy and Investor Relations:

- Present company products and strategy at industry events such as CES, Comdex, and Microprocessor Forum.
- Discuss new products and initiatives with the press.
- Promote issues of interest to SONICblue to industry groups and in Washington.
- Brief analysts, and investors on company progress. Participate in quarterly conference calls.

IP Management and Licensing:

- Negotiated and managed partnership agreements including a critical cross-licensing agreement with Intel.
- Renegotiated technology-licensing agreements with IBM for workstation graphics products.
- Evaluated outside technology opportunities, managed video research and development, and managed corporate IP strategy with legal staff including patent filings, cross licensing, and litigation.

Consulting Professor , [1999-2002]
Stanford University, Stanford, CA

Teaching computer architecture and microprocessor design.

Assistant Professor [1991 - 1997]
Princeton University, Princeton, NJ

Teaching and research in the Electrical Engineering department. Research in embedded computing systems, multimedia, video signal processors, compiler optimization, and high performance computer architecture. Principal investigator or project manager for ~$6M in funded research.

Visiting Assistant Professor , [1992]
Carnegie Mellon University, Pittsburgh, PA

Research and preparation of teaching materials on advanced microprocessor designs including new superscalar and superpipelined processor architectures.

Founder and Vice President and Consultant, [1989 - 1995]
The Graphics Technology Company, Inc., Austin, TX

Founded company to develop touch-sensitive components and systems for the first generation of PDA devices and interactive public systems. Obtained financing from Gunze Corp., Osaka, Japan. Company is now part of 3M.

Senior Electrical Engineer, [1989]
ESL - TRW, Advanced Technology Division, Sunnyvale, CA

Designed the architecture for an Intel i860-based multiple-processor digital signal processing system for advanced military applications. Designed several FPGA interface chips for VME-bus systems.

Design Consultant, [1986 -1987]
Carroll Touch Division, AMP Inc., Round Rock, TX

Developed several new technologies for touch-screen systems. Designed the first ASIC produced for AMP, a mixed-signal interface chip for controlling touch-screen sensors. Developed the system electronics, system firmware, and customer utility software for numerous products including those based on the new ASIC.

Senior Design Engineer, [1983 -1985]
Touch Technology Inc., Annapolis, MD

Nintendo - Ancora Exh. 1003

Advisory Boards:

Director, Turtle Beach Corporation (NASDAQ:HEAR) (formerly Parametric Sound Corporation), KBGear Interactive, Inc., Comsilica, Inc., Rioport.com, various S3 subsidiaries.

Technical Advisory Boards, Ageia, Inc., Intellon, Inc., Comsilica, Inc., Entridia, Inc., Siroyan, Ltd., BOPS, Inc, Quester Venture Funds

Carnegie Mellon University Silicon Valley Advisory Board; Johns Hopkins University Tech Transfer Advisory Board

Awards:

Micro Test-of-Time Award (in recognition of one of the ten most influential papers of the first 25 years of the symposium), 2014
Business 2.0 "20 Young Executives You Need to Know", 2002
Walter C. Johnson Prize for Teaching Excellence, 1997.
Princeton University Engineering Council Excellence in Teaching Award, Spring 1996
AT&T/Lucent Foundation Research Award, 1996.
Walter C. Johnson Prize for Teaching Excellence, 1995
IEEE Certificate of Appreciation, 1995, 2001.
AT&T Foundation Research Award, 1993.
Semiconductor Research Corporation Fellow, 1986 - 1991.
Burroughs Corporation Fellowship in Engineering, 1985 - 1986.

Professional Activities:

Program Chair: Micro-24, 1991, Hot Chips 13, 2001.

General Chair: Micro-26, 1993, Micro-33, 2000.

Associate Editor: IEEE Computer Architecture Letters; ACM Transactions in Embedded Computing Systems

Speaker at CES, WinHec, Comdex, Intel Dev. Forum, Digital Media Summit, Microprocessor Forum, etc.

Keynote speaker at Micro-34, ICME 2002

IEEE B. Ramakrishna Rau Award committee – 2012-2016

IEEE Computer Society Awards Committee – 2015

CES Awards Judge – 2016

Entrepreneurship Mentor – Draper University

Over 50 refereed publications.

Publications since January 2006:

Wolfe, A., "Retrospective on Code Compression and a Fresh Approach to Embedded Systems", IEEE MICRO, July/Aug. 2016, Invited paper.

Nintendo - Ancora Exh. 1003

Patents:

U.S. Pat. 5,041,701 – *Edge Linearization Device for a Contact Input System*, Aug. 20, 1991.
U.S. Pat. 5,438,168 – *Touch Panel*, Aug. 1, 1995.
U.S. Pat. 5,736,688 – *Curvilinear Linearization Device for Touch Systems*, Apr. 7, 1998.
U.S. Pat. 6,037,930 – *Multimodal touch sensitive peripheral device*, March 14, 2000.
U.S. Pat. 6,408,421 – *High-speed asynchronous decoder circuit for variable-length coded data*, June 18, 2002.
U.S. Pat. 6,865,668 – *Variable-length, high-speed, asynchronous decoder circuit*, March 8, 2005
U.S. Pat. 7,079,133 – *Superscalar 3D Graphics Engine*, July 18, 2006
EP 1 661 131 B1 – *PORTABLE ENTERTAINMENT APPARATUS*, Jan. 21, 2009
U.S. Pat. 7,555,006 – *Method and system for adaptive transcoding and transrating in a video network*, June 30, 2009
U.S. Pat. 7,996,595 – *Interrupt Arbitration for Multiprocessors*, Aug. 9, 2011
EP 2 241 979 B1 – *Interrupt Arbitration for Multiprocessors*, Oct. 10, 2011
U.S. Pat. 8,131,970 – *Compiler Based Cache Allocation*, March 6, 2012
U.S. Pat. 8,180,963 – *Hierarchical read-combining local memories*, May 15, 2012
U.S. Pat. 8,193,941 – *Snoring Treatment*, June 5, 2012
U.S. Pat. 8,203,541 – *OLED display and sensor*, June 19, 2012
U.S. Pat. 8,243,045 – *Touch-sensitive display device and method*, August 14, 2012
U.S. Pat. 8,244,982 – *Allocating processor cores with cache memory associativity*, August 14, 2012
U.S. Pat. 8,260,996 – *Interrupt Optimization for Multiprocessors*, Sept. 4, 2012
101185761 (KR) – *Noise Cancellation for Phone Conversation*, Sept. 19, 2012
101200740 (KR) – *OLED display and sensor*, November 7, 2012
101200741 (KR) – T*ouch-sensitive display device and method*, November 7, 2012
U.S. Pat. 8,321,614 – *Dynamic scheduling interrupt controller for multiprocessors*, Nov. 27, 2012
U.S. Pat. 8,352,679 – *Selectively securing data and/or erasing secure data caches responsive to security compromising conditions*, Jan. 8, 2013
U.S. Pat. 8,355,541 – *Texture Sensing*, Jan. 15, 2013
U.S. Pat. 8,370,307 – *Cloud Data Backup Storage Manager*, Feb. 5, 2013
U.S. Pat. 8,398,451 – *Tactile Input Interaction*, March. 19, 2013
JP 5241032 B2 – *Routing Across Multicore Network Using Real World or Modeled Data*, April 13, 2013
ZL201010124820.3 – *Interrupt Optimization for Multiprocessors*, April 17, 2013
U.S. Pat. 8,428,438 – *Apparatus for Viewing Television with Pause Capability*, April 23, 2013
JP 5266197 B2 – *Data Centers Task Mapping*, May 10, 2013
U.S. Pat. 8,508,498 – *Direction and Force Sensing Input Device*, August 13, 2013
U.S. Pat. 8,547,457 – *Camera Flash Mitigation*, October 1, 2013
U.S. Pat. 8,549,339 – *Processor core communication in multi-core processor*, October 1, 2013
101319048 (KR) – *Camera Flash Mitigation*, October 10, 2013
U.S. Pat. 8,628,478 – *Microphone for remote health sensing*, January 14, 2014
101362017 (KR) – *Thread Shift: Allocating Threads to Cores*, Feb. 5, 2014
101361928 (KR) – *Cache Prefill on Thread Migration*, Feb. 5, 2014
101361945 (KR) – *Mapping Of Computer Threads onto Heterogeneous Resources*, Feb. 5, 2014
JP 5487307 B2 – *Mapping Of Computer Threads onto Heterogeneous Resources*, Feb. 28, 2014
JP 5484580 B2 – *Task Scheduling Based on Financial Impact*, Feb. 28, 2014
JP 5487306 B2 – *Cache Prefill on Thread Migration*, Feb. 28, 2014
101372623 (KR) – *Power Management for Processor*, March. 4, 2014
101373925 (KR) – *Allocating Processor Cores with Cache Memory Associativity*, March 6, 2014
U.S. Pat. 8,676,668 – *Method for the determination of a time, location, and quantity of goods to be made available based on mapped population activity*, March 18, 2014
U.S. Pat. 8,687,533 – *Energy Reservation in Power Limited Networks*, April 1, 2014
101388735 (KR) – *Routing Across Multicore Networks Using Real World or Modeled Data*, April 17, 2014
U.S. Pat. 8,725,697 – *Cloud Data Backup Storage*, May 13, 2014
U.S. Pat. 8,726,043 – *Securing Backing Storage Data Passed Through a Network*, May 13, 2014
ZL201010124826.0 – *Dynamic scheduling interrupt controller for multiprocessors*, May 14, 2014
JP 5547820 B2 – *Processor core communication in multi-core processor*, May 23, 2014
U.S. Pat. 8,738,949 – *Power Management for Processor*, May 27, 2014

Nintendo - Ancora Exh. 1003

U.S. Pat. 8,751,854 – *Processor Core Clock Rate Selection*, June 10, 2014
JP 5559891 B2 – *Thermal Management in Multi-Core Processor*, June 13, 2014
101414033 (KR) – *Dynamic Computation Allocation*, June 25, 2014
JP 5571184 B2 – *Dynamic Computation Allocation*, July 4, 2014
101426341 (KR) – *Processor core communication in multi-core processor*, May 23, 2014
U.S. Pat. 8,799,671 – *Techniques for Detecting Encrypted Data*, Aug 5, 2014
101433485 (KR) – *Task Scheduling Based on Financial Impact*, Aug. 18, 2014
U.S. Pat. 8,824,666 – *Noise Cancellation for Phone Conversation*, Sept. 2, 2014
U.S. Pat. 8,836,516 – *Snoring Treatment*, Sept. 16, 2014
U.S. Pat. 8,838,370 – *Traffic flow model to provide traffic flow information*, Sept. 16, 2014
U.S. Pat. 8,838,797 – *Dynamic Computation Allocation*, Sept. 16, 2014
U.S. Pat. 8,854,379 – *Routing Across Multicore Networks Using Real World or Modeled Data*, Oct. 7, 2014
JP 5615361 B2 – *Thread Shift: Allocating Threads to Cores*, Oct. 15, 2014
U.S. Pat. 8,866,621 – *Sudden infant death prevention clothing*, Oct. 21, 2014
U.S. Pat. 8,881,157 – *Allocating threads to cores based on threads falling behind threads*, Nov. 4, 2014
ZL201080024755.5 – *Camera Flash Mitigation*, Nov 5, 2014
U.S. Pat. 8,882,677 – *Microphone for remote health sensing*, Nov. 11, 2014
U.S. Pat. 8,924,743 – *Securing Data Cache through Encryption*, December 30, 2014
U.S. Pat. 8,994,857 – *Camera Flash Mitigation*, March 31, 2015
JP 5699140 B2 – *Camera Flash Mitigation*, April 8, 2015
ZL201080035189.8 – *Thread Shift: Allocating Threads to Cores*, June 10, 2015
ZL201180005030.6 – *Processor core communication in multi-core processor*, June 10, 2015
U.S. Pat. 9,143,814 – *Method and system for adaptive transcoding and transrating in a video network*, Sept 22, 2015
ZL201080035177.5 – *Mapping Of Computer Threads onto Heterogeneous Resources*, Oct. 14, 2015
U.S. Pat. 9,178,694 – *Securing Backing Storage Data Passed Through a Network*, November 3, 2015
U.S. Pat. 9,189,282 – *Thread-to-core mapping based on thread deadline, thread demand, and hardware characteristics data collected by a performance counter*, November 17, 2015
U.S. Pat. 9,189,448 – *Routing image data across on-chip networks*, November 17, 2015
U.S. Pat. 9,208,093 – *Allocation of memory space to individual processor cores*, December 8, 2015
U.S. Pat. 9,239,994 – *Data Centers Task Mapping*, January 19, 2016
ZL201080036611.1 – *Allocating Processor Cores with Cache Memory Associativity*, January 20, 2016
EP2228779 B1 – *Traffic flow model to provide traffic flow information*, Jan. 27, 2016
U.S. Pat. 9,262,628 – *Operating System Sandbox*, February 16, 2016
GB2485682 – *Mapping Of Computer Threads onto Heterogeneous Resources*, Sept. 28, 2016
U.S. Pat. 9,330,137 – *Cloud Data Backup Storage Manager*, May. 3, 2016
ZL201080035185.X – *Cache Prefill on Thread Migration*, Aug. 24, 2016
U.S. Pat. 9,519,305 – *Processor Core Clock Rate Selection*, December 13, 2016
U.S. Pat. 9,569,270 – *Mapping thread phases onto heterogeneous cores based on execution characteristics and cache line eviction count*, February 14, 2017
GB2485683 – *Thread Shift: Allocating Threads to Cores*, Oct. 18, 2017
U.S. Pat. 9,852,435 – *Telemetrics based location and tracking*, December 26, 2017.
U.S. Pat. 9,915,994 – *Power management for processor*, March 13, 2018
U.S. Pat. 9,927,254 – *Traffic flow model to provide traffic flow information*, March 27, 2018
EP2254048 B1 – *Thread Mapping in Multi-Core Processors*, August 29, 2018
U.S. Pat. 10,860,432 – *Cloud Data Backup Storage Manager*, December 8, 2020

Nintendo - Ancora Exh. 1003

Expert testimony by deposition or at trial – April 15, 2016 - -present

Case	Venue	Case Number
INTELLECTUAL VENTURES II LLC v JP MORGAN CHASE & CO., et al.,	Southern District of NY	13 Civ. 3777
Certain Table Saws Incorporating Active Injury Mitigation Technology and Components Thereof (Sawstop v Bosch)	ITC	337-TA-965
INTER PARTES REVIEW OF U.S. PATENT NO. RE43,931 (TCL v Ericsson)	PTO	IPR2015-01602 IPR2015-01637 IPR2015-01641 IPR2015-01646 IPR2015-01674 IPR2015-01676 IPR2015-01761 IPR2015-01806
T-Mobile, USA, Inc. v. Huawei Device USA, Inc. and Huawei Technologies Co., Ltd.	W. D. Washington	14-cv-1351
AVM Technologies, LLC v. Intel Corporation and Related Matters	District of Delaware	15-33-RGA
TCL v. Ericsson, et al.	C.D. California	8:14-cv-341
Sonos Inc. v D&M Holdings Inc. d/b/a the D&M Group; D&M Holdings U.S. Inc. and Denon Electronics (USA) LLC.	District of Delaware	14-1330-RGA
Waymo v Uber et. al	N. D. CA	3:17-cv-00939
Certain Graphics Systems,Components Thereof and Consumer Products Containing the Same	ITC	337-TA-1044
Intellisoft, Ltd. v Acer	Superior Court of California for the County of Santa Clara	l-14-CV-272381
INTER PARTES REVIEW OF U.S. PATENT NO. 7,987,294 (D&M Holdings v Sonos)	PTO	IPR2017-01045
Joe Andrew Salazar v HTC Corporation	E.D. Texas	2:16-cv-010986-JRG-RSP
Hitachi Maxell, Ltd. v. ZTE Corporation and ZTE (USA), Inc.	E.D. Texas	5:16-cv-00179 5:16-cv-00178
D&M Holdings Inc. d/b/a the D&M Group; D&M Holdings U.S. Inc. v. Sonos Inc.	District of Delaware	16-00141-RGA
INTER PARTES REVIEW OF U.S. PATENT NO. 7,663,506 (Mediatek v AMD)	PTAB	IPR2017-00101 IPR2017-00102
Papst Licensing GmbH & Co. KG v. Samsung Electronics Co.,Ltd. and Samsung Electronics America, Inc.	E.D. Texas	6:15-CV-1102
HTC Corporation v. Telefonaktiebolaget LM Ericsson	E.D. Texas	6: 18-cv-00243-JRG
Seven Networks, LLC v ZTE (USA) Inc and ZTE Corporation	N. D. Texas - Dallas	3:17-CV-1495
AGIS Software Development, LLC v. HTC Corporation	E. D. Texas	2:17-cv-514
Barbaro Technologies, LLC v. Niantic, Inc	N.D. CA	3:18-cv-02955-RS
Immersion Corp. v. Samsung Electronics America, Inc. et al	E.D. Texas	2:17-cv-00572
INTER PARTES REVIEW OF U.S. PATENT NO. 7,171,526 (Northstar Innovations – Micron)	PTAB	IPR2018-01004 IPR2018-01005

Nintendo - Ancora Exh. 1003

CISCO SYSTEMS, INC., vs. UNILOC USA, INC., UNILOC 2017 LLC and UNILOC LICENSING USA LLC	N.D. CA	3:18-cv-04991-SI
INTER PARTES REVIEW OF U.S. PATENT NO. 8,020,014 (Intel/VLSI)	PTAB	IPR2018-01661 IPR2018-01312
INTER PARTES REVIEW OF U.S. PATENT NO Patent 9,294,799 (Comcast/Rovi)	PTAB	IPR2019-00299
Inter Partes Review of U.S. Patent No 7720929 (Unified Patents v Datascape)	PTAB	IPR2019-01115
Solas OLED Ltd,, v. Samsung Display Co,, Ltd,, Samsung Electronics Co, Ltd,, and Samsung Electronics America, Inc,,	PTAB	IPR2019-01668
INTER PARTES REVIEW OF U.S. PATENT NO Patent 8,973,069 (Comcast/Rovi)	PTAB	IPR2019-01434
U.S. Patent No. 8,448,215 IPR (Comcast v Rovi)	PTAB	IPR2019-01353
U.S. PATENT NO. 8,847,898 IPR filing (Samsung v Neodron)	PTAB	IPR2020-00234
U.S. PATENT NO. 8,610,009 IPR filing (Samsung v Neodron)	PTAB	IPR2020-00225
U.S. PATENT NO. 10,365,747 IPR filing (Samsung v Neodron)	PTAB	IPR2020-00308
AGIS Software Development LLC v. Google LLC, AGIS Software Development LLC v. WAZE Mobile Limited,and AGIS Software Development LLC v. Samsung Elecs. Co., Ltd. et al	E.D. Texas	2:19-cv-00361-JRG 2:19-cv-00359-JRG 2:19-cv-00362-JRG
Inter Partes Review of U.S. Patent No 8,112,670 (Sony)	PTAB	IPR2020-00726
U.S. Patent 8,819,505 IPR (Intel v PACT)	PTAB	IPR2020-00525
Inter Partes Review of U.S. Patent No 8,078,540 (Sony)	PTAB	IPR2020-00922
Inter Partes Review of U.S. Patent No 9,037,807 (Intel v PACT)	PTAB	IPR2020-00540
HONG KONG UCLOUDLINK NETWORK TECHNOLOGY LIMITED AND UCLOUDLINK (AMERICA), LTD., vs. SIMO HOLDINGS INC. AND SKYROAM, INC.,	N.D. CA	3:18-cv-05031-EMC

Nintendo - Ancora Exh. 1003

APPENDIX B-6

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 766 165 A2

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
02.04.1997 Bulletin 1997/14

(51) Int. Cl.[6]: **G06F 1/00**

(21) Application number: **96111086.3**

(22) Date of filing: **10.07.1996**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **31.08.1995 JP 224338/95**

(71) Applicant: **FUJITSU LIMITED**
Kawasaki-shi, Kanagawa 211 (JP)

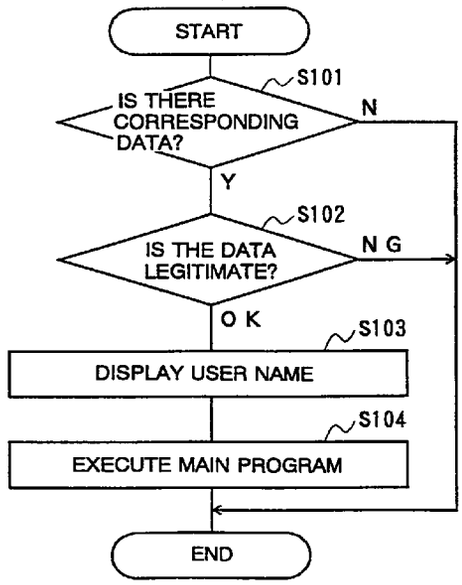
(72) Inventors:
- **Hasebe, Takayuki,**
  c/o Fujitsu Limited
  Kawasaki-shi, Kanagawa 211 (JP)
- **Torii, Naoya,**
  c/o Fujitsu Limited
  Kawasaki-shi, Kanagawa 211 (JP)

(74) Representative: **Seeger, Wolfgang, Dipl.-Phys.**
Georg-Hager-Strasse 40
81369 München (DE)

(54) Licensee notification system

(57) There is disclosed a licensee notification system for implementing a software sales system wherein license information for converting to executable form software that is presented to a user in non-executable form is communicated to the user from a management center on condition of payment of a charge, and the software is converted into executable form at the user terminal using this license information. The subject of the licensee notification system is software that decides whether or not the correspondence relationship between user identification information and signature information stored in the license file is legitimate, and, if it is legitimate, displays the user identification information to the user before starting proper operation; or, if it is not legitimate, does not start proper operation. The licensee notification system is constituted by connecting the management center and user terminals by communication circuits. If license information is requested from the user terminal, the management center transmits license information combining in integral form the user identification information identifying the user and conversion information for converting the software to executable form. The user terminal enables the software using the conversion information contained in this license information and writes user identification information and signature information whose content is determined in accordance with the content of the user identification information to a license file that is referred to when this software is operating.

FIG. 6



EP 0 766 165 A2

Nintendo - Ancora Exh. 1003

Description

BACKGROUND OF THE INVENTION

1.Field of the Invention

The present invention relates to a licensee notification system employed for the sale of software using a high speed communication network such as B-ISDN and a large-capacity storage medium such as a CD-ROM.

2.Description of the Related Art

With the development of high speed communication technology such as B-ISDN (broad-band integrated services digital network) and high-capacity storage media such as CD-ROMs (compact disk read only memory) such means can now be used to distribute computer programs or video data or audio data. For example, video works which were previously supplied on video tape are now being sold stored on CD-ROM. Also, game programs etc, which contain a large amount of picture data, are being sold stored on CD-ROM. The same applies to high speed communication networks, in which the software supplier can now distribute the software by various methods.

One of these methods of software sales is the so-called "locked software" sales system. In the locked software sales system, a CD ROM on which are stored a large number of software items whose functions are restricted is sold cheaply. By using the various items of software of software on the CD-ROM that is purchased, in a condition with the functional restrictions imposed, the end user is able to make a decision as to whether or not he needs each software item. Then, if the end user does require the software, he obtains (purchases) a restriction-removal code corresponding to this software from a management center operated by the software distributor, and is able to use this restriction-removal code to remove the functional restrictions on the software.

Such a sales system may be implemented, as a specific example, using the software sales system shown in Fig. 10. As shown in this Figure, this software sales system comprises user terminals 31 and management center 32. The user terminal 31 and the management center 32 are connected by means of a communication circuit.

When actually purchasing the software (i.e. when purchasing a restriction-removal code), the end user, using a user ID etc, sets up a communication path with the management center and executes the prescribed procedure required to request that a restriction-removal code be sent to the user terminal 31. This procedure includes the input of a "contents ID", which is information for identifying the software item that is to be purchased actually. In response to the execution of such a procedure, the user terminal 31 sends to the management center 32 the contents ID and for example the characteristic information of the user, consisting of the ID of the CPU provided in user terminal 31.

Within the management center 32, there is provided a software database (software DB) in which software decoding keys employed for encoding the various software items are stored in association with the contents ID. When a contents ID is received from user terminal 31, the software decoding key corresponding to the contents ID is read from software database 33. Also, encoding unit 34 in management center 32 generates a user individual key by encoding the user characteristic information from user terminal 31 by the key "Ks". Encoding unit 35 sends the results of the encoding of the software decoding key from software database 33 to user terminal 31 as restriction-removal code, using the user individual key from encoding unit 34.

Encoding unit 36 in user terminal 31 generates a user individual key by encoding the user characteristic information with the key "Ks". Decoding unit 37 uses the user individual key generated by encoding unit 36 to decode the restriction removal code from management center 32, thereby generating the software decoding key. Installation unit 38 then uses this software decoding key to decode the software in CD-ROM corresponding to the contents ID sent to center terminal 32: thus the software is put in a condition where it can be used with the functional restrictions removed, and, in this form, is installed on to a storage device such as a hard disk device.

With such a software sales system, it is possible to determine the software item to be purchased after actually ascertaining its contents: thus, the possibility that the purchased software might be completely different from that intended, as could happen if the purchase were made solely on the basis of the details contained in a catalogue, can be completely eliminated. Also, since the software on the CD ROM is stored in a form which is not executable without knowing special information, illicit installation can be prevented.

However, once the software has been installed, it is an extremely easy operation to copy this. Thus, the problem has arisen of unscrupulous persons copying the software without the consent of the software supplier. Various methods (so-called protection methods) of preventing such illicit copying are known but there is no way to prevent illicit copying by a person possessing knowledge at the level of the BIOS (basic input/output system). Whichever method is used, it can do no more than make it more difficult to perform illicit copying.

For this reason, software is sold in which the name of the authorized user is displayed on start-up, with the object of preventing illicit copying psychologically rather than physically. That is, the aim is to prevent illicit copying of software by displaying the name of the authorized user of the software when the illicitly copied software is executed.

However, even with such software, if the copying is inclusive of the installation software that sets the user

Nintendo - Ancora Exh. 1003

name, when the software is run, it can be made to display the name of the person who made the illicit copy: thus, sufficient effectiveness in preventing illicit copying was not obtained.

SUMMARY OF THE INVENTION

An object of the present invention is to provide a licensee notification system whose psychological effectiveness in preventing illicit copying is very high.

A first licensee notification system according to the present invention consists in a system for implementing a software sales system in which software in non-executable form is presented to a user, and license information for converting the software into executable form is informed to the user on condition of payment of a charge, and the software is converted into executable form using this license information.

The first licensee notification system is constituted of a management center and user terminals; its subject is software which includes instructions that command a terminal to read user identification information in a license file and to notify the user identification information to the user on commencement of its operation.

The management center comprises a license information generating unit that generates license information combining in integrated form user identification information that specifies a user and conversion information for converting software to executable form.

The user terminal comprises a storage unit, a conversion unit, and license file creating unit. In more detail, the storage unit is employed for storing the license file and software converted to executable form. The license information, which is generated by the license information generating unit in the management center, is given to the conversion unit. The conversion unit then converts the software to executable form using the license information and installs it in the storage unit. The license file creating unit creates the license file which contains the user identification information contained in the license information, and stores the license file in the storage unit.

That is, in the first licensee notification system, software is installed in the user terminal so that the user identification information of the legitimate user is notified to the user on its start-up, using the license information which is generated in the management center and contains the user identification information.

A second licensee notification system according to the present invention is constituted of a management center and user terminal; its subject is software which includes instructions that commands the user terminal to read user identification information in the prescribed location in the software and to notify the user identification information to the user on commencement of its operation.

The management center comprises a license information generating unit that generates license information combining in integrated form user identification information identifying a user and conversion information for converting software into executable form.

The user terminal comprises a storage unit, a conversion unit and a software rewriting unit. Of these, the storage unit is employed for storing the software after this has been converted to executable form. The conversion unit converts the software to executable condition using the license information generated by the license information generating unit in the management center, and then installs it in the storage unit. The software rewriting unit rewrites the information of the prescribed location of the software that has been installed by the conversion unit with the user identification information contained in the license information.

That is, in this second licensee notification system, installation is performed with the content of the software rewritten such that the user identification information of the legitimate user is notified on start-up, using the license information which is generated in the management center and contains the user identification information.

The third licensee notification system according to the present invention has as its subject software that, on commencement of operation, includes instructions commanding the user terminal to read user identification information in a license file and to notify the user identification information to the user.

The management center in the third licensee notification system comprises a license information generating unit that generates license information consisting of an integral combination of conversion information for converting the software to executable form and user identification information identifying a user.

The user terminal comprises a storage unit for storing a license file, a license file creating unit, and a software execution unit. The license file creating unit creates the license file containing the license information generated by the license information generating unit, and stores the license file in the storing unit. The software execution unit, when execution of the software is designated, converts the software to executable form using the license information stored in the license file and expands it into memory, and commences operation in accordance with the expanded software.

That is, in the third licensee notification system, the software, which is presented to the user in non-executable form, is converted to executable form in accordance with the license information containing the user identification information every time execution is designated.

The fourth licensee notification system according to this invention is constituted of management center and user terminal. The subject of the system is software which judges the legitimacy of user identification information on the basis of signature information stored in a license file on commencement of operation and, if the user identification information is legitimate, commences proper operation after notifying this user identification information to the user, and, if the user identification

Nintendo - Ancora Exh. 1003

information is not legitimate, terminates operation.

The management center comprises a license information generating unit that generates license information combining in integral form the user identification information identifying the user and signature information whose content is determined in accordance with the user identification information.

The user terminal comprises a storage unit for storing the license file and a license file creating unit that creates the license file containing the user identification information contained in the license information generated by the license information generating unit and stores the license file in the storage unit.

That is, in the fourth licensee notification system, the license information which is necessary for running the software normally is generated on the basis of the user identification information in the management center and is informed to the user terminal.

It may be noted that although in the first to the fourth licensee notification system any means could be employed for notification of the license information, if notification of license information is performed using a communication circuit, a system that is simple to operate can be formed.

Also, it is possible to employ information including the name of the user as user identification information. It is also possible to employ a unit that generates license information including user identification information encoded with a characteristic key of the software. In this case, software is presented to user which including instructions that command the user terminal to notify to the user the result of decoding the user identification information using the characteristic key.

In the first to the third licensee notification systems, it is also possible to make the software that is presented to the user encoded, and to make the conversion information for decoding the encoded software. Also, it is possible to employ, in such a licensee notification system, license information containing the user identification information in a form that cannot be separated without special information. For example, it is possible to employ information, as license information, which is the result of encoding the conversion information and user identification information, combined in integrated manner.

Also, it is possible to make the first to third licensee notification system a system whose subject is software that, if the signature information stored in the license file does not correspond to the user identification information, terminates operation, and, as the license file creating unit, to employ a unit that generates signature information whose content is determined in accordance with the content of the user identification information, and creates the license file containing the signature information. In this case, it can be made more difficult to alter the user identification information that is notified to the user on start-up of the software. Also, in the case of such software, it is possible to employ as license information generating unit a unit that generates license information containing signature information whose contents are determined in accordance with the contents of the user identification information, and, as license file creating unit, to employ a unit that creates the license file containing signature information contained in the license information.

Also, it is possible to make the second licensee notification system a system whose subject is software that, if signature information stored in the second predetermined location does not correspond to user identification information stored in a prescribed location, terminates its operation, and, as software rewriting unit, to employ a unit that rewrites the information of the prescribed location of the software with the user identification information contained in the license information and that rewrites the information at the second prescribed location of the software with signature information whose content is determined in accordance with the user identification information. Also, in the case of such software, it is possible to employ as license information generating unit a unit that generates license information containing signature information whose content is determined in accordance with the content of the user identification information, and, as software rewriting unit, to employ a unit that rewrites information of the prescribed location with user identification information contained in the license information and that rewrites the information at the second prescribed location in the software by signature information contained in the license information.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a functional block diagram illustrating the layout of a licensee notification system according to a first embodiment of the present invention;

Fig. 2 is a diagram given in explanation of the content of the user database provided in the management center comprised in the licensee notification system according to the first embodiment;

Fig. 3 is a diagram illustrating the content of the software database provided in the management center comprised in the licensee notification system according to the first embodiment;

Fig. 4 is a diagram illustrating the content of a license file provided in a user terminal comprised in the licensee notification system according to the first embodiment;

Fig. 5 is a diagram illustrating the structure of software that is the subject of the licensee notification system according to the first embodiment;

Fig. 6 is a flow chart illustrating the operating sequence of software that is the subject of the licensee notification system according to the first embodiment;

Fig. 7 is a function block diagram illustrating the organization of a user terminal employed in the licensee notification system according to a second embodiment;

Nintendo - Ancora Exh. 1003

Fig. 8 is a diagram illustrating the structure of software that is the subject of the licensee notification system according to the second embodiment;

Fig. 9 is a flow chart showing the operating sequence of software that is the subject of the licensee notification system according to the second embodiment; and

Fig. 10 is a functional block diagram showing the structure of the licensee notification system used in a prior art locked software sales system.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention is described in detail below with reference to the drawings.

First embodiment

Fig. 1 is a functional block diagram of a licensee notification system according to a first embodiment of the present invention. This licensee notification system is a system where CD-ROMs storing a large number of software items of restricted function are sold cheaply, and software sales are effected by selling the information needed to cancel the function restrictions of the software in this CD-ROM. Payment of the fee could be effected by for example notification of the subscriber number of a cash card or notification of a bank account withdrawal number or the like.

As shown in the drawings, the licensee notification system is constituted by user terminals 11 and management center 12 connected by means of a communication circuit. User terminals 11 and management center 12 may be described as computers and commence operation as an ensemble of the function blocks illustrated when prescribed programs are run.

First of all, the operation of management center 12 will be described.

Management center 12 is provided with two databases, called user database (user DB) 13 and software database (software DB) 14. As shown in Fig. 2, user DB 13 stores the correspondence relationship between the user ID, which is identification information given to users of this system by the manager, and the user name, which is the identification information of the user as employed in ordinary society. As shown in Fig. 3, software DB 14 stores the correspondence relationship between the contents ID, which is the identification information of each software item supplied and stored in the CD ROM, and the software decoding key, which is the decoding information needed to decode this software item.

A link-up unit 15 in management center 12 generates license information by combining the two data items: user name and software decoding key. An encoding unit 16 generates a user's individual key by encoding with key "Ks" the user characteristic information (details to be explained later) from user terminal 11. An encoding unit 17 generates coded license information by encoding the license information from link-up unit 15 using the user's individual key generated by encoding unit 16. In the present licensee notification system, a DES (data encryption standard) algorithm is employed for encoding and decoding.

The various function blocks that are not in management center 12 are arranged to operate synchronously when there is a request from user terminal 11 for information for removal of the function restrictions. Specifically, when management center 12 receives a request for information for removal of function restrictions relating to a software item from user terminal 11, it transmits to user terminal 11 coded license information containing the user's name and the software decoding key needed to remove the functional restrictions on the software item.

Next, the operation of user terminal 11 will be described. When user terminal 11 runs the programs for communication and installation, it executes the operation described below.

A request transmission unit 18 in user terminal 11 transmits to management center 12 information including the user ID, contents ID, and user's characteristic information. Request transmission unit 18 commences operation when the keyboard (not shown) of user terminal 11 is operated in accordance with a prescribed procedure that is predetermined as the procedure for request of information for removal of functional restrictions. This request procedure includes keyboard input of the user ID and contents ID; request transmission unit 18 transmits to management center 12 the keyboard input information and the user's characteristic information, which is constituted by the ID of the CPU which is employed in user terminal 11.

As already explained, when a request for information for removal of functional restrictions is received from user terminal 11, management center 12 sends to user terminal 11 encoded license information. As a result, after request transmission unit 18 has been operated, user terminal 11 receives encoded license information from management center 12.

As shown in the drawings, the encoded license information is input to decoding unit 20 in user terminal 11. Decoding unit 20 also inputs the user's individual key, which is generated by encoding unit 19 using the user's characteristic information and "Ks". Using this user's individual key, decoding unit 20 decodes the encoded license information from center terminal 12. The license information, which is the result of this decoding, is input to separating unit 21, which is a unit that performs reverse processing against link-up unit 15 in management center 12. Separating unit 21 separates and extracts the software decoding key and user name from the license information, and respectively supplies the extracted software decoding key and user name to installation unit 22 and license file compilation unit 23.

Installation unit 22, using the software decoding key from separating unit 21, removes the functional restric-

Nintendo - Ancora Exh. 1003

tions on the specific software item (details to be described later) in accordance with the contents ID transmitted by request transmission unit 18. License file compilation unit 23 compiles a license file 24 using the user name and contents ID from separating unit 21.

Fig. 4 shows diagrammatically the contents of license file 24. As shown in the drawing, license file 24 stores information consisting of contents ID and user name, and signature information, which is information encoded using a signature key.

Further detailed description of the operation of installation unit 22 and the operation of the software installed by installation unit 22 is given below using Fig. 5 and Fig. 6. Of these Figures, Fig. 5 is a view showing diagrammatically the structure of software that is the subject of the present licensee notification system and Fig. 6 is a flow chart showing the operating sequence of the CPU in the user terminal when the software that is the subject of the present licensee notification system is actuated.

As shown in Fig. 5, the software that is the subject of the present system includes a license display routine 25 and main program 26. In the main program there are defined the operating procedures relating to the proper functions of this software; in license display routine 25, there is defined the content to be executed prior to execution of main program 26.

When this software is actuated, as shown in Fig. 6, the CPU, first of all, by checking the contents ID in the license file, decides whether or not data corresponding to the software that is being actuated is present in the license file (step S101). Then, if the corresponding data exists (step S101:Y), the CPU performs a check of the legitimacy of the corresponding data (step 102). In this step, the CPU encodes the information consisting of contents ID and user name stored in the license file using the signature key that is set as data in license display routine 25, and if the result of this encoding agrees with the signature information, decides that the data is legitimate.

If it is legitimate (step S102:OK), the CPU displays the user name which is read from the license file (step S103), and commences operation in accordance with the main program (step S104).

Also, if the corresponding data is not present in the license file (step S101:N) or if the content of the license file is found to be not legitimate (step S102:NG), i.e. if the content of the license file is found to be different from the result of the compilation performed by license file compilation unit 23, the CPU terminates operation without displaying the user name or executing the main program.

As described above, with the licensee notification system according to the first embodiment, in the user terminal, installation of the software is performed such that the user name is displayed on start-up, using the encoded license information supplied from the management center. Also, the installed software is executed only when the legitimacy of the license file is verified. As a result, with this licensee notification system, even if the software and license file are copied illicitly after being installed, it is difficult to change the user name appearing on start-up of the software. The person who has made the illicit copy has no alternative but to use the software with the name of another person being displayed. As a result, illicit copying of the software can be prevented if the present licensee notification system is employed.

It should be noted that the licensee notification system of the first embodiment could be modified in various ways.

It would for example be possible to constitute the system such that notification of the contents ID etc to the management center and notification of the encoded license information to the user terminal were performed by another information transmission unit, such as the post. In this case, the user terminal is constituted such that installation is effected using encoded license information input from the keyboard. It is also possible to constitute the system such that the license information is notified in un-encoded form.

It is also possible to arrange that the signature information is generated at the management center end, and encoded license information containing this signature information is notified to the user terminal.

It is also possible to constitute the system such that, instead of the user name and signature information, information representing the user name in encoded form is stored in the license file, and, when the installed software is executed, the information in the license file is decoded by the software and displayed.

It would also be possible to arrange that the software was converted into executable condition not on installation of the software but rather every time execution of the software was specified, the software then being expanded in the memory and operation commenced in accordance with the software now in the memory.

Also, the medium whereby the software is supplied is not restricted to CD-ROM; a supply mode could be adopted in which the software was stored on another recording medium such as a floppy disk, or downloaded through a communication circuit.

Second embodiment

A licensee notification system according to a second embodiment of the present invention is described below with reference to Fig. 7 and Fig. 9. Of these Figures, Fig. 7 is a functional block diagram illustrating the layout of a user terminal wherein a licensee notification system according to the second embodiment is provided. Fig. 8 is a diagram illustrating the structure of software that is the subject of this licensee notification system. Fig. 9 is a flow chart showing the operating procedure of the CPU when the software that is the subject of the present licensee notification system is executed.

In the licensee notification system according to the

Nintendo - Ancora Exh. 1003

second embodiment, a management center of the same construction as management center 12 in the first embodiment is employed. Also, as can be seen from the functional block diagram shown in Fig. 7, the difference of the action of the user terminal 11 is slight, so the description will be confined to the parts of which the details of operation differ with respect to the licensee notification system of the first embodiment.

As shown in Fig. 7, in user terminal 11 according to the second embodiment, the software decoding key and user name that are separated by separating unit 21 are both input to the installation unit 29. Installation unit 29 effects installation by decoding the software in the CD ROM using the software decoding key, and generates the user name in encoded form by encoding the user name. Thus, installation unit 29, as shown diagrammatically in Fig. 8, writes the encoded user name 28 that is thus generated in a prescribed location of license display routine 26.

As shown in Fig. 9, when the software that is the subject of the licensee notification system of the second embodiment is started up, the encoded user name that was written in the prescribed location in license display routine 25 is read and decoded (step S201). Then, after display of the decoded user name has been performed (step S202), main program 27 is executed (step S203).

That is, with this licensee notification system, the user name that is displayed on start-up of the software is set by directly rewriting the content of the software.

Even with the licensee notification system of this second embodiment, enabling of the software such that the user's name is displayed on start-up is effected independently of keyboard input from the user terminal, so it is not possible to alter the user name that is displayed by the software simply by making an illicit copy of the installation software. Also, the installed software is executed only when the legitimacy of the license file has been verified. Consequently, with this licensee notification system, even if the installed software is illicitly copied, it is difficult to alter the user name that is displayed on start-up, so the person who has made the illicit copy has no alternative but to use the software with another person's name displayed. Thus, use of this licensee notification system can psychologically prevent illicit copying.

It should be noted that with this licensee notification system according to the second embodiment, various modifications are possible just as in the case of the licensee notification system according to the first embodiment.

For example, it would be possible to constitute a system such that the notification of the contents ID etc to the management center and the notification of the encoded license information to the user terminal were performed by another information transmission unit such as the post. And it is also possible to constitute a system such that license information is notified in unencoded form.

Also, it is possible to constitute a system such that the software in question is made software wherein operation is stopped if signature information stored in a second prescribed location of the software does not correspond to user identification information stored in a first prescribed location and to arrange that the installation unit 29 writes the user name to the first prescribed location in the software and writes the signature information, consisting of this user name in encoded form, to the second prescribed location.

Claims

1. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software into executable form is informed to the user on condition of payment of a charge, said licensee notification system comprising:

   a management center including
   license information generating means for generating license information combining in integrated form conversion information for converting software to executable form and user identification information specifying the user; and
   a user terminal including
   storage means,
   conversion means for converting the software in non-executable form into executable form using the license information generated by said license information generating means and installing the software in executable form into said storage means, and
   license file creating means for creating a license file containing the user identification information contained in the license information generated by said license information generating means, and for storing the license file in said storage means; and
   wherein the software includes instructions that command the user terminal to read user identification information in the license file and to notify the user identification information to the user on commencement of its operation.

2. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software in non-executable form to executable form is informed to the user on condition of payment of a charge, said licensee notification system comprising:

   a management center including
   license information generating means for generating license information combining in integrated form the conversion information for

Nintendo - Ancora Exh. 1003

converting software to executable form and user identification information specifying the user; and

a user terminal including

storage means,

conversion means that converts the software in non-executable form into executable form using the license information generated by said license information generating means and installing the software in executable form in said storage means, and

software rewriting means for rewriting the information in a prescribed location of the software installed by said conversion means with the user identification information contained in the license information generated by said license information generating means; and

wherein the software includes instructions that commands the user terminal to read user identification information in the prescribed location in the software and to notify the user identification information to the user on commencement of its operation.

3. A licensee notification system for use in a software sales system in which software in non-executable form is presented to a user, and license information for converting the software in non-executable form to executable form is informed to the user on condition of payment of a charge, said licensee notification system comprising:

a management center including

license information generating means for generating license information combining in integrated form conversion information for converting software to executable form and user identification information specifying the user; and

a user terminal including

storage means;

license file creating means for creating a license file containing the user identification information contained in the license information generated by said license information generating means, and for storing the license file in said storing means, and

software execution means for converting, when execution of the software is designated, the software into executable form using the license information in the license file and expanding the software in executable form into memory and executing operation in accordance with the software in the memory; and

wherein the software includes instructions that commands the user terminal to read user identification information in the license file and to notify the user identification information to a user on commencement of its operation.

4. A licensee notification system for use in a software sales system in which software that refers to license information is presented to a user, and the license information in respect of software is informed to the user on condition of payment of a charger said licensee notification system comprising:

a management center including

license information generating means for generating license information combining in integrated form user identification information specifying the user and signature information whose content is determined in accordance with the user identification information; and

a user terminal including

storage means, and

license file creating means for creating the license file containing the license information generated by the license information generating means and storing the license file in the storage means, and

wherein the software includes instructions that command the user terminal to judge the legitimacy of the user identification information in the license file using the signature information in the license file on commencement of its operation and, if the user identification information is legitimate, to commence proper operation after notifying the user identification information to the user, but, if the user identification information is not legitimate, to stop the operation.

5. A licensee notification system according to claim 1, 2, 3 or 4, wherein the software includes instructions that command the user terminal to display the user identification information on a display of the user terminal.

6. A licensee notification system according to claim 1, 2, 3, or 4, wherein the user terminal further comprises:

transmitting means for transmitting a request signal which requests license information to the management center through a communication circuit; and

said license information generating means in the management center, when the request signal is received from the user terminal, generates license information and transmits the license information to the user terminal through the communication circuit.

7. A licensee notification system according to claim 1, 2, 3 or 4, wherein the user identification information includes the name of the user.

8. A licensee notification system according to claim 1,

Nintendo - Ancora Exh. 1003

2, 3 or 4, wherein the license information generating means generates license information including user identification information encoded with a characteristic key of the software; and

the software includes instructions that command the user terminal to inform to the user the result of decoding the user identification information using the characteristic key.

9. A licensee notification system according to claim 1, 2, or 3, wherein the software is presented to the user in encoded form, and the conversion information is information for decoding the software.

10. A licensee notification system according to claim 1, 2, or 3, wherein the license information contains the user identification information in a form that is incapable of being separated without special information.

11. A licensee notification system according to claim 1, 2, or 3, wherein the license information is the result of encoding the conversion information and user identification information, combined in integrated manner.

12. A licensee notification system according to claim 1 or claim 3, wherein the license file creating means creates the license file containing signature information whose content is determined in accordance with the content of the user identification information; and

the software includes instructions that commands the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file.

13. A licensee notification system according to claim 1 or claim 3, wherein said license information generating means generates license information containing signature information whose content is determined in accordance with the content of the user identification information;

said license file creating means creates the license file containing the signature information contained in the license information generated by said license information generating means; and

the software includes instructions that command the user terminal to terminate operation if the signature information in the license file does not correspond to the user identification information in the license file.

14. A licensee notification system according to claim 2, wherein the software rewriting means rewrites the information of the prescribed location of the software with the user identification information contained in the license information and rewrites the information of a second location in the software with the signature information whose content is determined in accordance with the user identification information; and

the software including instructions that command the user terminal to terminate operation if the signature information stored in the second prescribed location does not correspond to the user identification information stored in the prescribed location.

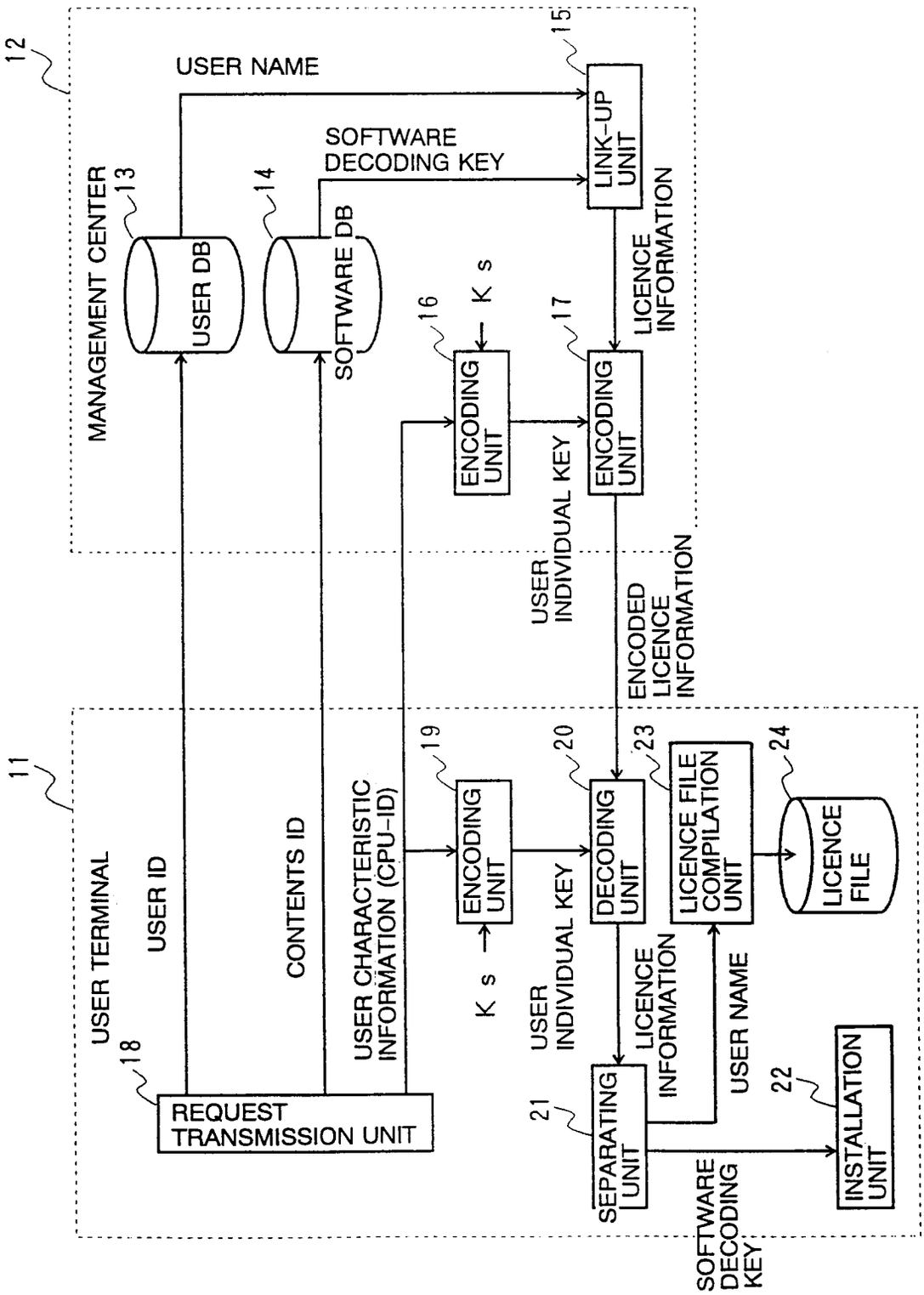
15. A licensee notification system according to claim 2, wherein said license information generating means generates license information including signature information whose content is determined in accordance with the content of the user identification information;

said software rewriting means rewrites the information of the prescribed location with the user identification information contained in the license information and rewrites the information of the second prescribed location in the software by means of the signature information contained in the license information; and

the software includes instructions that command the user terminal to terminate operation if the signature information stored in the second prescribed location does not correspond to the user identification information that is stored at the prescribed location.

Nintendo - Ancora Exh. 1003

FIG. 1



Nintendo - Ancora Exh. 1003

FIG. 2

13

USER ID	USER NAME
M0001111	TOKKYO TARO

FIG. 3

14

CONTENTS ID	DECODING KEY
ABC00001	xxxxxxxxx

FIG. 4

24

CONTENTS ID	USER NAME	SIGNATURE INFORMATION
ABC00001	TOKKYO TARO	zzzzzzzz

Nintendo - Ancora Exh. 1003

FIG. 5

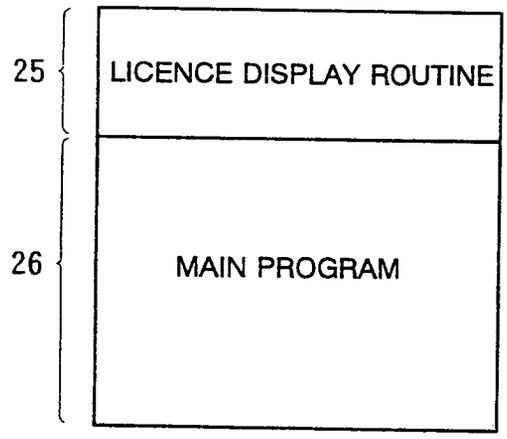
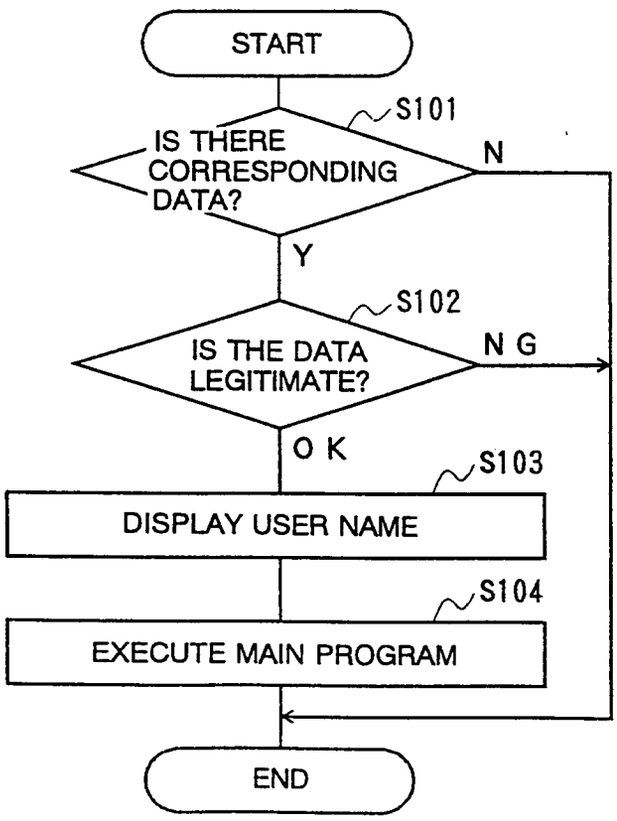
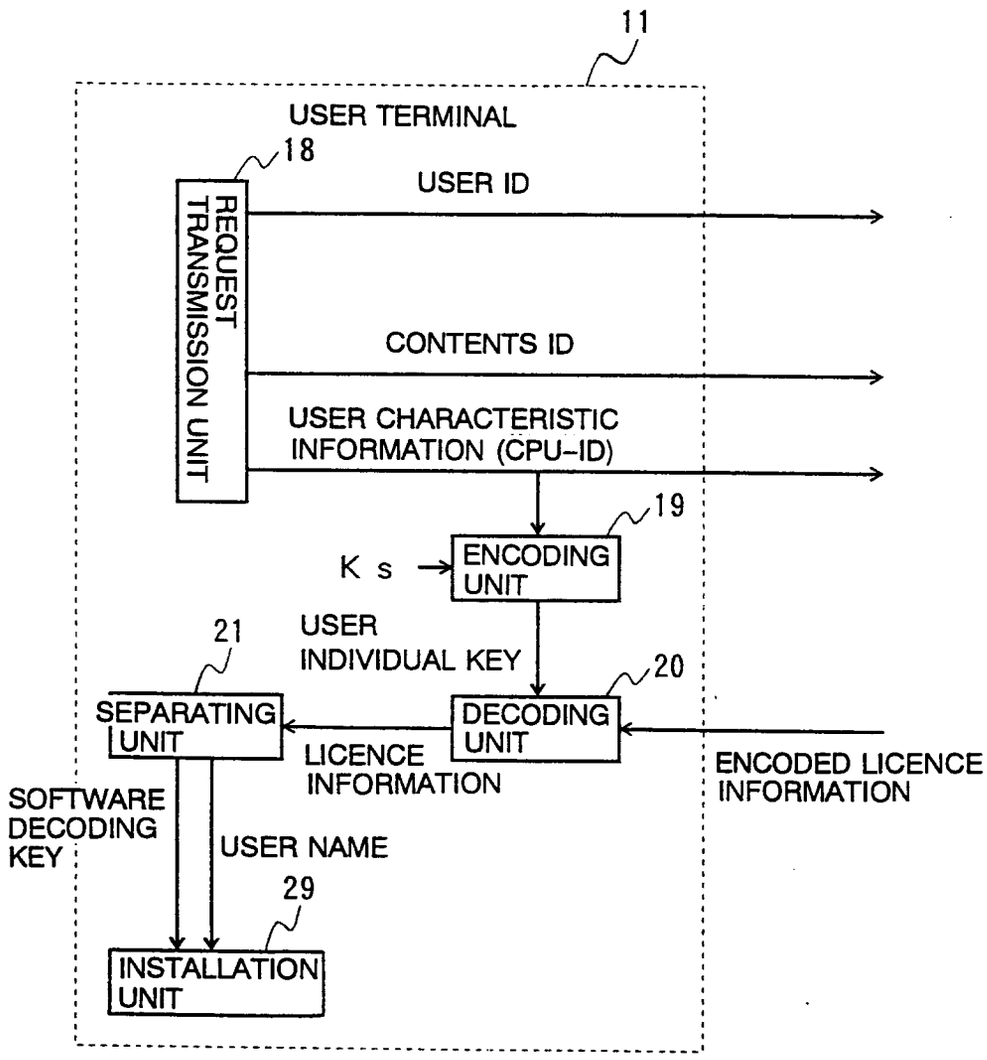


FIG. 6



Nintendo - Ancora Exh. 1003

FIG. 7



Nintendo - Ancora Exh. 1003

FIG. 8

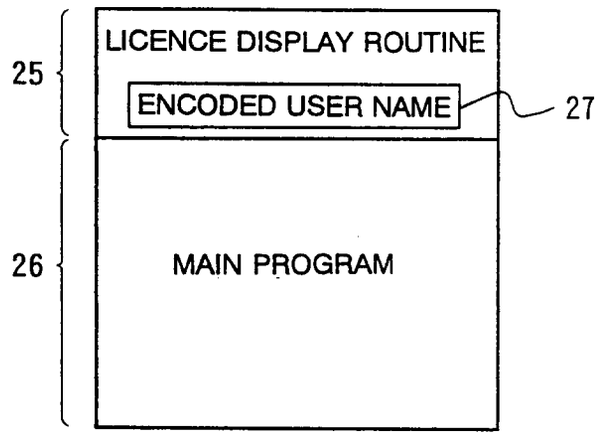
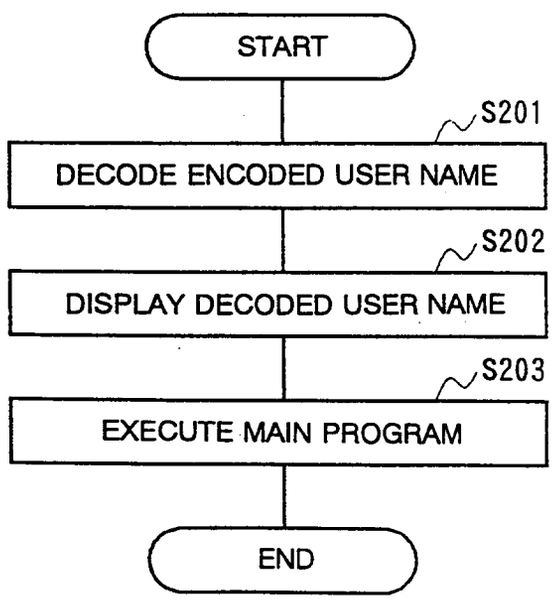
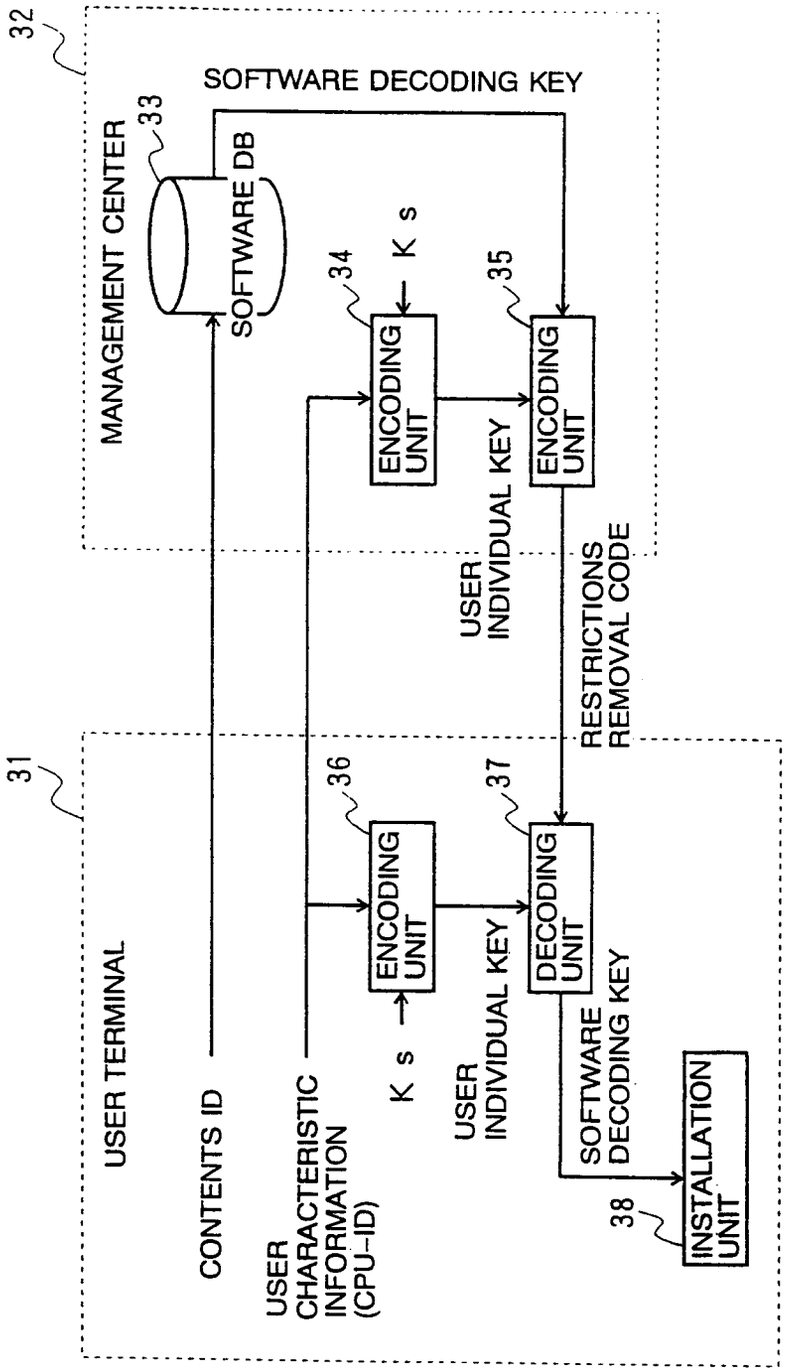


FIG. 9



Nintendo - Ancora Exh. 1003

FIG. 10



Nintendo - Ancora Exh. 1003

APPENDIX B-7

US005724425A

United States Patent [19]

Chang et al.

[11] Patent Number: 5,724,425

[45] Date of Patent: Mar. 3, 1998

[54] **METHOD AND APPARATUS FOR ENHANCING SOFTWARE SECURITY AND DISTRIBUTING SOFTWARE**

[75] Inventors: **Sheue-Ling Chang**, Cupertino; **James Gosling**, Woodside, both of Calif.

[73] Assignee: **Sun Microsystems, Inc.**

[21] Appl. No.: **258,244**

[22] Filed: **Jun. 10, 1994**

[51] **Int. Cl.**[6] **H04L 9/00**; H04L 9/30; H04L 9/32

[52] **U.S. Cl.** **380/25**; 380/4; 380/23; 380/30; 380/49; 380/50

[58] **Field of Search** 380/4, 23, 25, 380/30, 49, 50

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,558,176	12/1985	Arnold et al.	380/4
4,634,807	1/1987	Chorley et al.	380/4
4,670,857	6/1987	Rackman	380/4
5,343,527	8/1994	Moore	380/4

OTHER PUBLICATIONS

Davida et al., "Defending Systems Against Viruses through Cryptographic Authentication", IEEE Symposium, 1989, pp. 312–318.

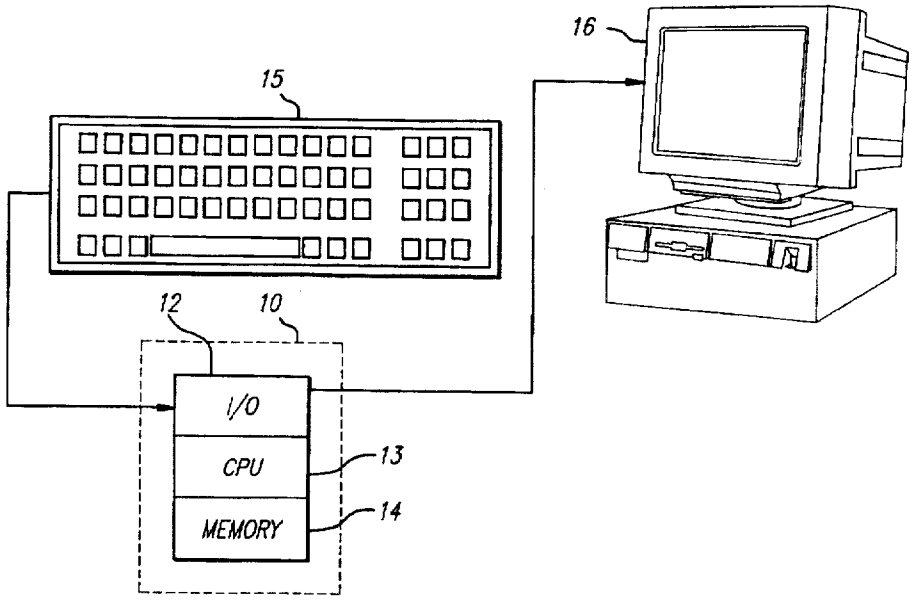
RSA Data Security, Inc., "RSA Certificate Services", Jul. 15, 1993, pp. 1–41.

Primary Examiner—Bernarr E. Gregory
Attorney, Agent, or Firm—McCutchen, Doyle, Brown & Enersen LLP; Ronald S. Laurie, Esq.; Joseph Yang

[57] **ABSTRACT**

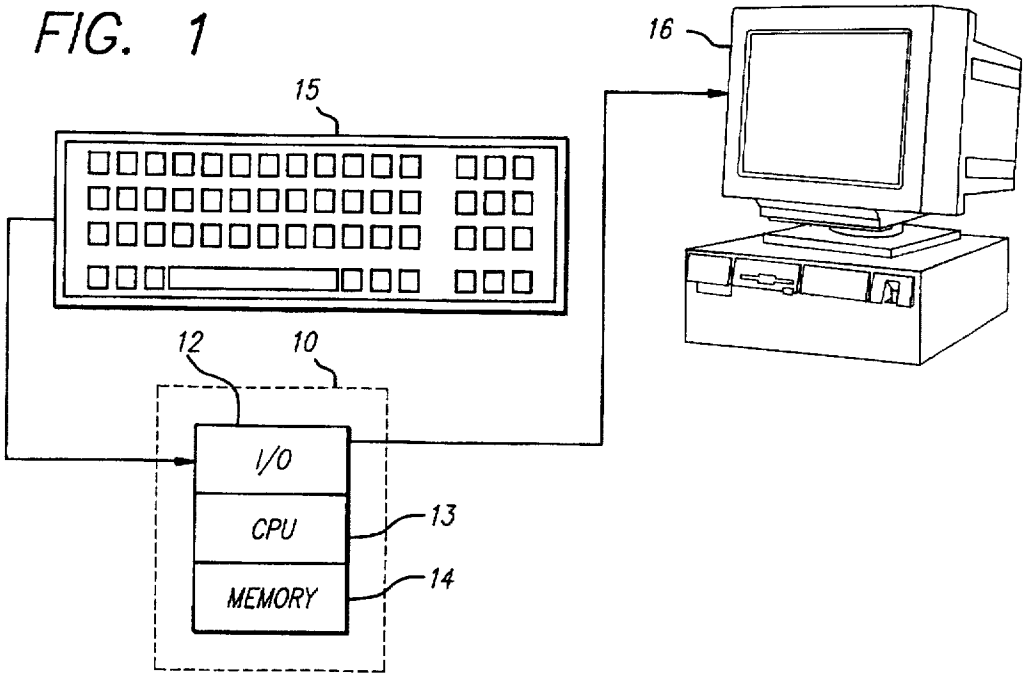
Source code to be protected, a software application writer's private key, along with an application writer's license provided to the first computer. The application writer's license includes identifying information such as the application writer's name as well as the application writer's public key. A compiler program executed by the first computer compiles the source code into binary code, and computes a message digest for the binary code. The first computer then encrypts the message digest using the application writer's private key, such that the encrypted message digest is defined as a digital "signature" of the application writer. A software passport is then generated which includes the application writer's digital signature, the application writer's license and the binary code. The software passport is then distributed to a user using any number of software distribution models known in the industry. A user, upon receipt of the software passport, loads the passport into a computer which determines whether the software passport includes the application writer's license and digital signature. In the event that the software passport does not include the application writer's license, or the application writer's digital signature, then the user's computer system discards the software passport and does not execute the binary code. As an additional security step, the user's computer computes a second message digest for the software passport and compares it to the first message digest, such that if the first and second message digests are not equal, the software passport is also rejected by the user's computer and the code is not executed. If the first and second message digests are equal, the user's computer extracts the application writer's public key from the application writer's license for verification. The application writer's digital signature is decrypted using the application writer's public key. The user's computer then compares a message digest of the binary code to be executed, with the decrypted application writer's digital signature, such that if they are equal, the user's computer executes the binary code.

72 Claims, 5 Drawing Sheets



Nintendo - Ancora Exh. 1003

FIG. 1


15
16
12
10
I/O
CPU
MEMORY
13
14

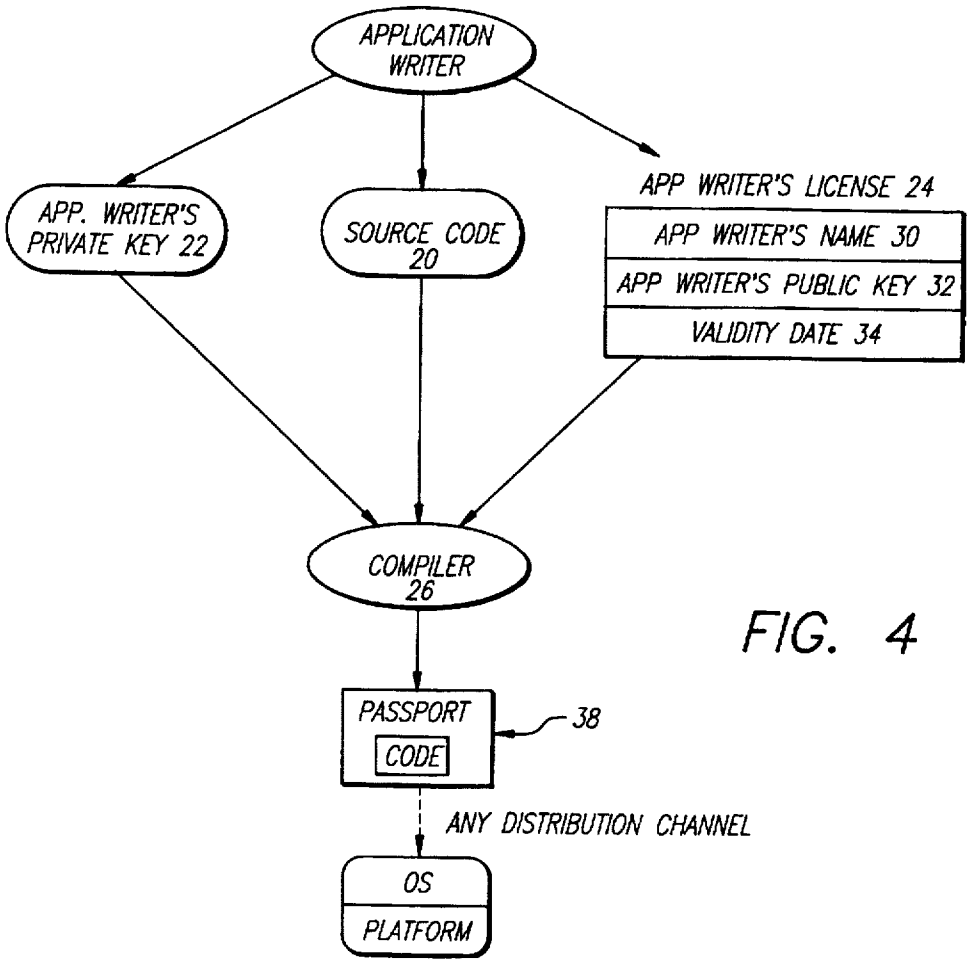
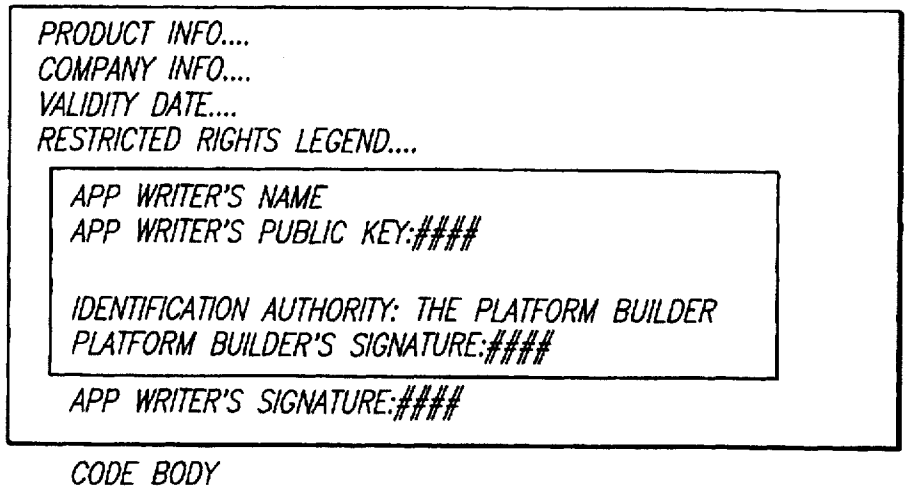

APPLICATION WRITER
APP. WRITER'S PRIVATE KEY 22
SOURCE CODE 20
APP WRITER'S LICENSE 24
APP WRITER'S NAME 30
APP WRITER'S PUBLIC KEY 32
VALIDITY DATE 34
COMPILER 26
PASSPORT
CODE
38
ANY DISTRIBUTION CHANNEL
OS
PLATFORM

FIG. 4

FIG. 2


PRODUCT INFO....
COMPANY INFO....
VALIDITY DATE....
RESTRICTED RIGHTS LEGEND....
APP WRITER'S NAME
APP WRITER'S PUBLIC KEY:####
IDENTIFICATION AUTHORITY: THE PLATFORM BUILDER
PLATFORM BUILDER'S SIGNATURE:####
APP WRITER'S SIGNATURE:####
CODE BODY

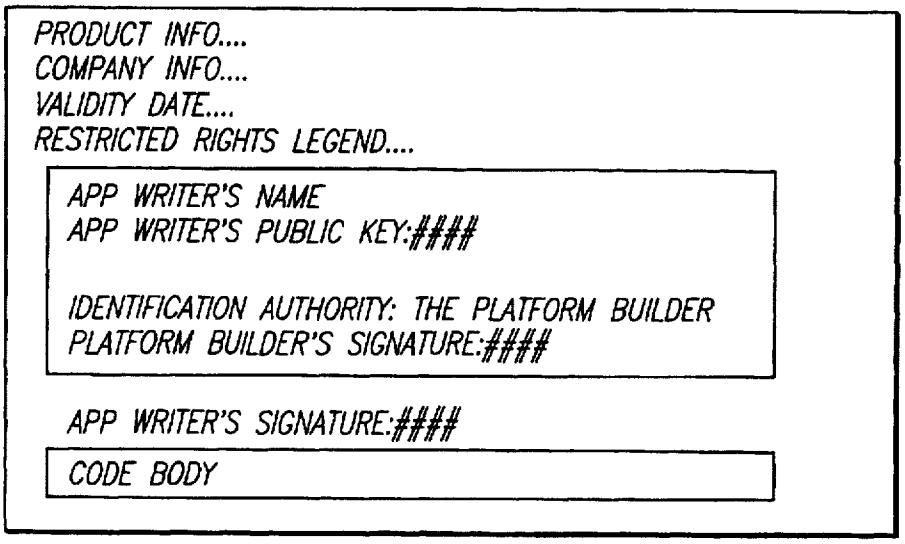
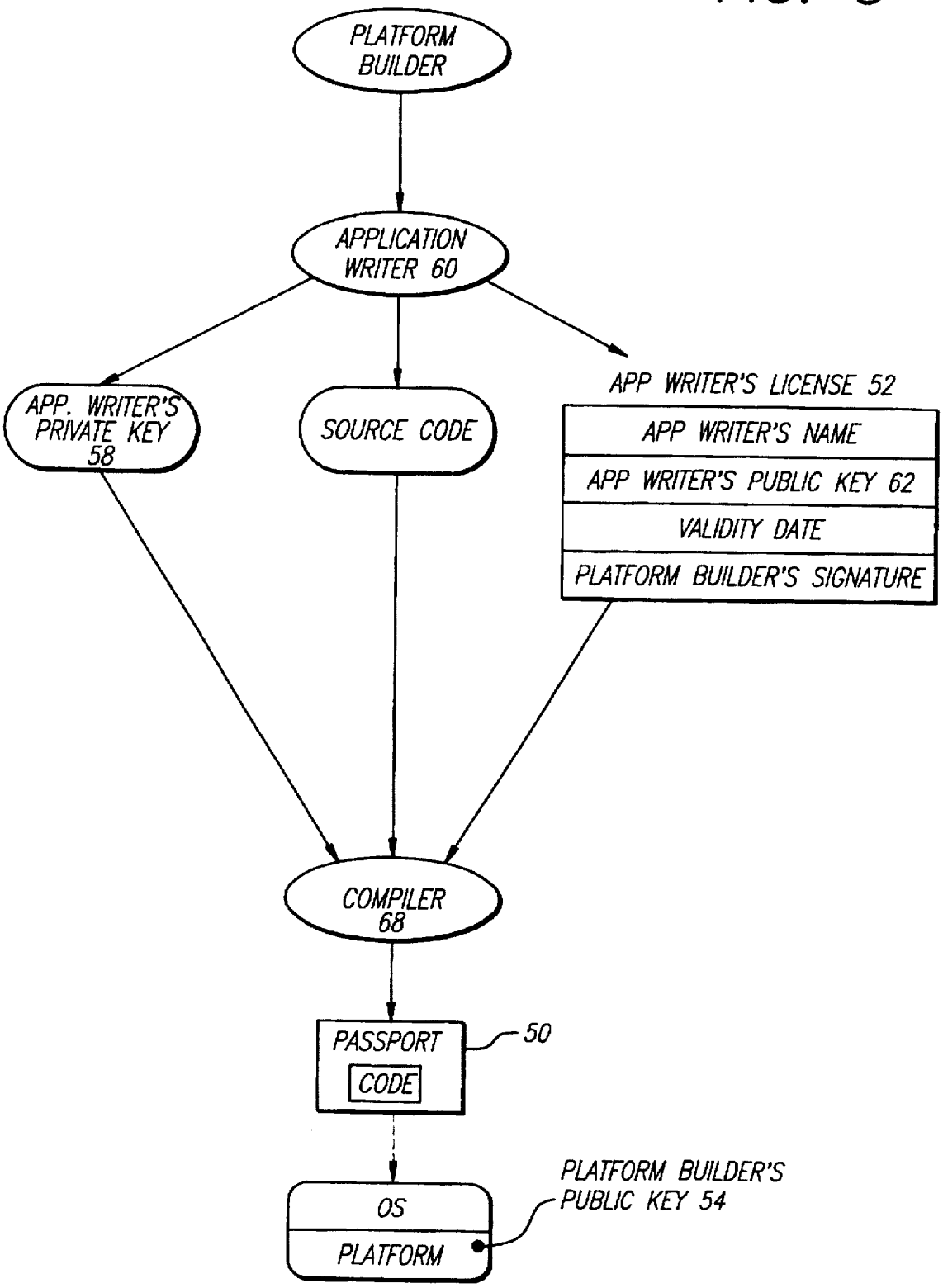

PRODUCT INFO....
COMPANY INFO....
VALIDITY DATE....
RESTRICTED RIGHTS LEGEND....
APP WRITER'S NAME
APP WRITER'S PUBLIC KEY:####
IDENTIFICATION AUTHORITY: THE PLATFORM BUILDER
PLATFORM BUILDER'S SIGNATURE:####
APP WRITER'S SIGNATURE:####
CODE BODY

FIG. 3

FIG. 5


PLATFORM BUILDER
APPLICATION WRITER 60
APP. WRITER'S PRIVATE KEY 58
SOURCE CODE
APP WRITER'S LICENSE 52
APP WRITER'S NAME
APP WRITER'S PUBLIC KEY 62
VALIDITY DATE
PLATFORM BUILDER'S SIGNATURE
COMPILER 68
PASSPORT
CODE
50
OS
PLATFORM
PLATFORM BUILDER'S PUBLIC KEY 54

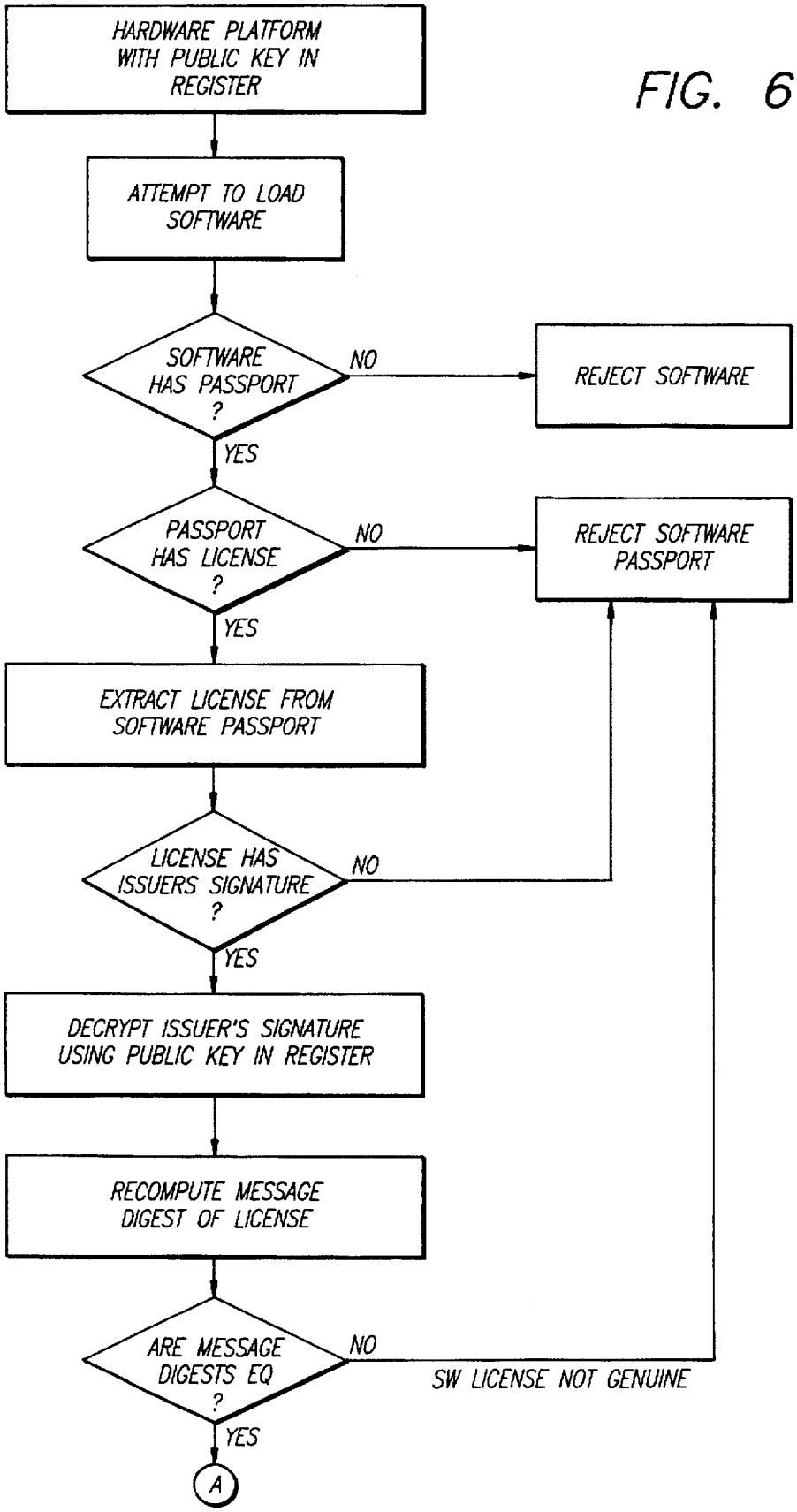

HARDWARE PLATFORM WITH PUBLIC KEY IN REGISTER
ATTEMPT TO LOAD SOFTWARE
SOFTWARE HAS PASSPORT ?
NO
REJECT SOFTWARE
YES
PASSPORT HAS LICENSE ?
NO
REJECT SOFTWARE PASSPORT
YES
EXTRACT LICENSE FROM SOFTWARE PASSPORT
LICENSE HAS ISSUERS SIGNATURE ?
NO
YES
DECRYPT ISSUER'S SIGNATURE USING PUBLIC KEY IN REGISTER
RECOMPUTE MESSAGE DIGEST OF LICENSE
ARE MESSAGE DIGESTS EQ ?
NO
SW LICENSE NOT GENUINE
YES
A

FIG. 6(a)

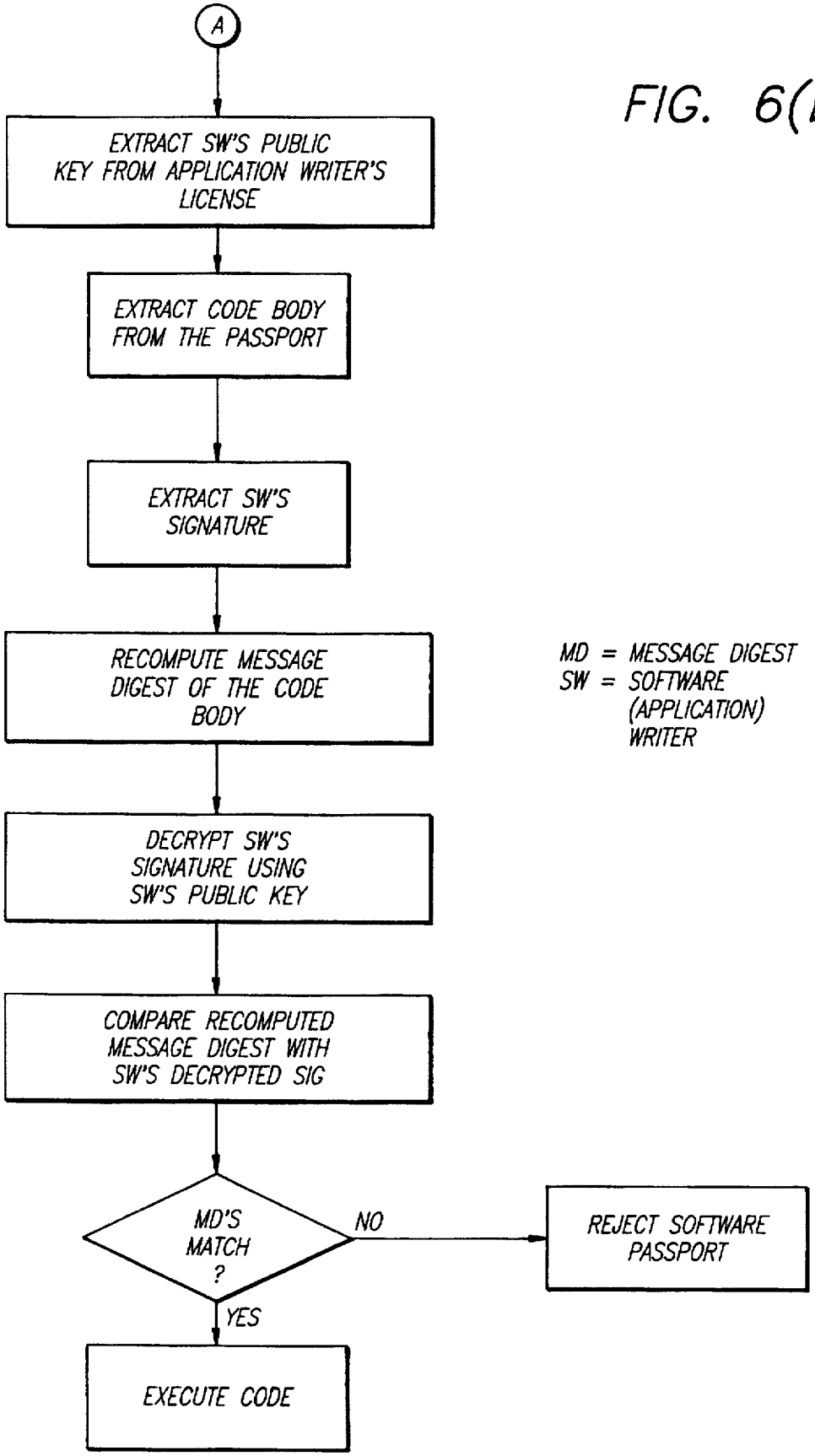

A
EXTRACT SW'S PUBLIC KEY FROM APPLICATION WRITER'S LICENSE
EXTRACT CODE BODY FROM THE PASSPORT
EXTRACT SW'S SIGNATURE
RECOMPUTE MESSAGE DIGEST OF THE CODE BODY
DECRYPT SW'S SIGNATURE USING SW'S PUBLIC KEY
COMPARE RECOMPUTED MESSAGE DIGEST WITH SW'S DECRYPTED SIG
MD'S MATCH ?
NO
REJECT SOFTWARE PASSPORT
YES
EXECUTE CODE
MD = MESSAGE DIGEST
SW = SOFTWARE (APPLICATION) WRITER

FIG. 6(b)

METHOD AND APPARATUS FOR ENHANCING SOFTWARE SECURITY AND DISTRIBUTING SOFTWARE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the use of public key encryption, and more particularly, the present invention relates to the use of public key encryption to achieve enhanced security and product authentication in the distribution of software.

2. Art Background

Public key encryption is based on encryption algorithms that have two keys. One key used for encryption, and the other key is used for decryption. There is a known algorithm that computes the second key given the first. However, without full knowledge of all the parameters, one cannot compute the first key given the second key. The first key is referred to as the "private key", and the second key is referred to as the "public key". In practice, either the private key or the public key may be used to encrypt a message, with the opposite key used to decrypt it. In general, the private key must be kept private, but the public key may be provided to anyone. A variety of public key cryptographic schemes have been developed for the protection of messages and data (See, Whitfield Diffie, "The First Ten Years of Public Key Cryptography" (IEEE Proceedings, Vol. 76, No. 5, 1988) and Fahn, "Answers to Frequently Asked Questions about Today's Cryptography (RSA Laboratories 1992).

Public key cryptography is used to send secure messages across public communication links on which an intruder may eavesdrop, and solves the problem of sending the encryption password to the other side securely.

Public key systems may also be used to encrypt messages, and also to effectively sign messages, allowing the received party to authenticate the sender of the message. One can also use public key cryptography to seal or render tamper-proof a piece of data. In such event, the sender computes a message digest from the data using specially designed cryptographically strong digests designed for this purpose. The sender then uses the private key to encrypt the message digest, wherein this encrypted message digest is called a digital "signature". The sender then packages the data, the message digest and the public key together. The receiver may check for tampering by computing the message digest again, then decrypting the received message digest with the public key. If the recomputed and decrypted message digests are identical, there was no tampering of the data.

"Viruses" and "worms" are computer code cleverly inserted into legitimate programs which are subsequently executed on computers. Each time the program is executed the virus or worm can cause damage to the system by destroying valuable information, and/or further infect and spread to other machines on the network. While there are subtle differences between a virus and a worm, a critical component for both is that they typically require help from an unsuspecting computer user to successfully infect a computer or a corporate network.

Infection of computers by viruses and worms is a general problem in the computer industry today. In addition, corporate networks are vulnerable to frontal assaults, where an intruder breaks into the network and steals or destroys information. Security breaches of any kind on large corporate networks are a particularly worrisome problem, because of the potential for large-scale damage and economic loss.

Moreover, security breaches are more easily accomplished when a corporate network is connected to a public network, such as the Internet. Companies take a variety of measures to guard against breaches of network security, either through frontal assaults or infections, without cutting themselves off from the benefits of being connected to a world-wide network.

The solution adopted by most companies that wish to reap the benefits of connecting to the Internet, while maintaining security, is the installation of a firewall. Firewalls generally restrict Internet file transfers and telnet connections. Such transfers and connections can only be initiated from within the corporate network, such that externally initiated file transfers and telnet connections are refused by the firewall. Firewalls allow electronic mail and network news to freely flow inside the firewall's private network. The use of corporate firewalls allows employees to readily exchange information within the corporate environment, without having to adopt extreme security measures. A good firewall implementation can defend against most of the typical frontal assaults on system security.

One method of preventing viruses and worms from infecting a corporate network is to never execute a program that may contain viruses. In general, programs legitimately deployed throughout the corporate network should be considered virus free. All binary executables, all unreviewed shell scripts, and all source code fetched from outside the firewall are software that may contain a worm or virus.

However, outside binary executables, shell scripts, and source code may enter a corporate firewall through an E-mail attachment. For example, the shell scripts that are used to make and send multiple files using E-mail and the surveytools that start up by activating the E-mail attachment may allow virus entry. Executables can also be directly fetched through the iftp program, through a world-wide web browser such as Mosaic, or from an outside contractor whose network has already been compromised.

In addition, the commercial software release and distribution process presents security and authentication problems. For example, some of the information associated with software, such as the originating company or author, restricted rights legends, and the like are not attached to the code itself. Instead, such information is provided as printed matter, and is separated from the code once the package is opened and the code installed. Even applications that attempt to identify themselves on start-up are susceptible to having the identification forged or otherwise counterfeited.

A user has no mechanism to authenticate that the software sold is actually from the manufacturer shown on the label. Unauthorized copying and the sale of software is a significant problem, and users who believe that they are buying software with a manufacturer's warranty instead purchase pirated software, with neither a warranty nor software support. The problem of authenticating the original source of the software is accentuated when software is intended to be distributed through networks, and a user's source for the software may be far removed from the original writer of the software. In addition, a user does not have that ability to verify that the software purchased contains only the original manufacturer's code. A user also does not have a method for detecting any tampering, such as the existence of a virus, that may cause undesirable effects.

All of the above problems are related to the transport of software both from manufacturers to users and from user to user. Furthermore, the transport problem is independent of the transport medium. The problem applies to all transport media, including floppy disk, magnetic tape, CD-ROM and networks.

Nintendo - Ancora Exh. 1003

As will be described, the present invention provides a method and apparatus for authenticating that software distributed by a manufacturer is a legitimate copy of an authorized software release, and that the software contains only the original manufacturers code without tampering. The present invention solves the above identified problems through the use of a "software passport" which includes the digital signature of the application writer and manufacturer. As will be described, the present invention may also be used to protect intellectual property, in the form of copyrighted computer code, by utilizing cryptographic techniques referred to herein as public key encryption.

SUMMARY OF THE INVENTION

This invention provides a method and apparatus utilizing public key encryption techniques for enhancing software security and for distributing software. The present invention includes a first computer which is provided with source code to be protected using the teachings of the present invention. In addition, a software application writer's private key, along with an application writer's license provided to the first computer. An application writer generally means a software company such as Microsoft Corporation, Adobe or Apple Computer, Inc. The application writer's license includes identifying information such as the application writer's name as well as the application writer's public key. A compiler program executed by the first computer compiles the source code into binary code, and computes a message digest for the binary code. The first computer then encrypts the message digest using the application writer's private key, such that the encrypted message digest is defined as a digital "signature" of the application writer. A software passport is then generated which includes the application writer's digital signature, the application writer's license and the binary code. The software passport is then distributed to a user using any number of software distribution models known in the industry.

A user, upon receipt of the software passport, loads the passport into a computer which determines whether the software passport includes the application writer's license and digital signature. In the event that the software passport does not include the application writer's license, or the application writer's digital signature, then the user's computer system discards the software passport and does not execute the binary code. As an additional security step, the user's computer computes a second message digest for the software passport and compares it to the first message digest, such that if the first and second message digests are not equal, the software passport is also rejected by the user's computer and the code is not executed. If the first and second message digests are equal, the user's computer extracts the application writer's public key from the application writer's license for verification. The application writer's digital signature is decrypted using the application writer's public key. The user's computer then compares a message digest of the binary code to be executed, with the decrypted application writer's digital signature, such that if they are equal, the user's computer executes the binary code. Accordingly, software products distributed with the present invention's software passport permits the user's computer to authenticate the software as created by an authorized application writer who has been issued a valid application writer's license. Any unauthorized changes to the binary code comprising the distributed software is evident through the comparison of the calculated and encrypted message digests.

The present invention is also described with reference to an embodiment used by computing platforms designed to execute only authorized software. A platform builder provides an application writer with a platform builder's digital signature which is included in the application writer's license. The first computer compiles the software into binary code and computes a first message digest for the binary code. The first computer further encrypts the first message digest using the application writer's private key, such that the encrypted first message digest is defined as the application writer's digital signature. A software passport is generated which includes the application writer's digital signature, the application writer's license and the binary code. The software passport is then distributed to a user through existing software distribution channels. The user's computing platform, which may be a computer, a video game box or a set top box, is provided with the platform builder's public key. Upon receipt of the software passport, the computing platform determines if the software passport includes an application writer's license. If it does not, the hardware platform rejects the execution of the code. If a software passport is present, the hardware platform extracts the application writer's license from the passport and determines whether or not the passport includes the platform builder's signature. The platform builder's signature is then decrypted using the public key provided in the platform. The computing platform recomputes the message digest of the application writer's license, and compares the received message digest with the recomputed message digest, such that if the digests are not equal, the software passport is not considered genuine and is rejected. If the message digests are equal, the hardware platform extracts the application writer's public key from the application writer's license, and extracts the application writer's digital signature. The hardware platform then recomputes the message digest of the binary code comprising the application software to be executed, and decrypts the application writer's digital signature using the application writer's public key. The hardware platform then compares the recomputed message digest for the binary code with the application writer's decrypted signature, such that if they are equal, the binary code is executed by the hardware platform. If the recomputed message digest and the application writer's decrypted signature are not equal, the software passport is rejected and the code is not executed.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a data processing system incorporating the teachings of the present invention.

FIG. 2 conceptually illustrates use of the present invention's software passport where the application code and the software passport are provided in separate files.

FIG. 3 conceptually illustrates use of the present invention's use of the software passport where the application code and the software passport are distributed in the same file.

FIG. 4 diagrammatically illustrates the present invention's process for generating a software passport.

FIG. 5 diagrammatically illustrates the use of the present invention for platform producer licensing.

FIGS. 6*a* and 6*b* are flowcharts illustrating the steps executed by the present invention for verifying that a valid software license exists, and that the software writer's ("SW's") signature is valid, prior to permitting the execution of a computer program.

NOTATION AND NOMENCLATURE

The detailed descriptions which follow are presented largely in terms of symbolic representations of operations of

Nintendo - Ancora Exh. 1003

data processing devices. These process descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art.

An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities may take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, displayed and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, messages, names, elements, symbols, operations, messages, terms, numbers, or the like. It should be borne in mind, however, that all of these similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

In the present invention, the operations referred to are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases, the reader is advised to keep in mind the distinction between the method operations of operating a computer and the method of computation itself. The present invention relates to method steps for operating a computer, coupled to a series of networks, and processing electrical or other physical signals to generate other desired physical signals.

The present invention also relates to apparatus for performing these operations. This apparatus may be specially constructed for the required purposes or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. The method/process steps presented herein are not inherently related to any particular computer or other apparatus. Various general purpose machines may be used with programs in accordance with the teachings herein, or it may prove more convenient to construct specialized apparatus to perform the required method steps. The required structure for a variety of these machines will be apparent from the description given below.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, numerous specific details are set forth such as system configurations, representative data, computer code organization, encryption methods, and devices, etc., to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well known circuits and structures are not described in detail in order to not obscure the present invention. Moreover, certain terms such as "knows", "verifies", "compares", "examines", "utilizes", "finds", "determines", "challenges", "authenticates", etc., are used in this Specification and are considered to be terms of art. The use of these terms, which to a casual reader may be considered personifications of computer or electronic systems, refers to the functions of the system as having human-like attributes, for simplicity. For example, a reference herein to an electronic system as "determining" something is simply a shorthand method of describing that the electronic system has been programmed or otherwise modified in accordance with the teachings herein. The reader is cautioned not to confuse the functions described with everyday human attributes. These functions are machine functions in every sense.

Exemplary Hardware

FIG. **1** illustrates a data processing system in accordance with the teachings of the present invention. Shown is a computer **10**, which comprises three major components. The first of these is an input/output (I/O) circuit **12** which is used to communicate information in appropriately structured form to and from other portions of the computer **10**. In addition, computer **10** includes a central processing (CPU) **13** coupled to the I/O circuit **12** and a memory **14**. These elements are those typically found in most general purpose computers and, in fact, computer **10** is intended to be representative of a broad category of data processing devices. Also, the computer **10** may be coupled to a network, in accordance with the teachings herein. The computer **10** may further include encrypting and decrypting circuitry incorporating the present invention, or as will be appreciated, the present invention may be implemented in software executed by computer **10**. A raster display monitor **16** is shown coupled to the I/O circuit **12** and issued to display images generated by CPU **13** in accordance with the present invention. Any well known variety of cathode ray tube (CRT) or other type of display may be utilized as display **16**.

The present invention's software passport identifies a portion of software, or some machine code (hereinafter "code"), in a manner similar to how a physical passport identifies a person. The concept is similar to the real-life passport system which forms the basis of a trust model among different nations. Physical passports enable border entry officers to identify each individual and make certain decisions based on his/her passport. As will be described below, a software passport is a modern release process for distributing software products. A software passport gives a software product an identity and a brand name. The software passport provides the basis of a trust model and allows computer users to identify and determine the genuineness of a software product based on the information contained in its passport.

Referring now to FIG. **2**, the present invention is illustrated in conceptual form for the case where the computer code (comprising a piece of software) and the software passport are in separate files. FIG. **3** illustrates the use of the present invention where the computer code comprising a piece of software and the software passport are in the same file.

As illustrated in FIGS. **2** and **3**, the information included in the present invention's software passport may include:

- product information, such as the software product's name and any other relevant information to the specific product;
- company information including the name of the company or the software application writer who has produced the product;
- a validity date which includes the issue date of the software passport and the expiration date of the passport;
- a restricted rights legend including copyright notices and other similar legends;
- the software code body including executable application code distributed to the user;
- an application writer's license; and,
- a software application writer's digital signature.

It will be appreciated that the components of a software passport are generally self-explanatory, with the application writer's license and digital signature explained in more detail below.

Nintendo - Ancora Exh. 1003

SOFTWARE PRODUCER'S DIGITAL SIGNATURE

A digital "signature" is produced by using certain cryptographic techniques of computing a message digest of a piece of software code (hereinafter "code"), and encrypting the message digest using the signer's private key. There are many known message digest algorithms, such as the MD2, MD4, and MD5 algorithms published by RSA, Inc. The use of private cryptographic techniques makes this signature very difficult to forge since the signer keeps the private key secret. The reader is referred to the papers by Whitfield Diffie, "The First Ten Years of Public Key Cryptography", Vol. 76, No. 5 (IEEE Proceedings, May 1988), which is attached hereto as Appendix A; and Whitfield Diffie, et al., "Authentication and Authenticated Key Exchanges" (1992 Kluwer Academic Publishers) attached hereto as Appendix B, for a detailed description of the operation of Diffie-Helman certificates and public key cryptography.

One may conceptualize the computing of the message digest for a piece of code as a mechanism of taking a photo snapshot of the software. When the code changes, its message digest reflects any differences. In the system of the present invention, this "digital signature" is stamped on the product prior to its release. The digital signature associates a product with the entity that has produced it, and enables consumers to evaluate the quality of a product based on the reputation of the producer. The signature also permits a consumer to distinguish the genuineness of a product.

SOFTWARE PRODUCER'S LICENSE

The present invention's software producer's license (at time referred to herein as the "application writer's license") is an identification similar to the home repair contractor's license issued by a state. A software producer's license identifies and certifies that the producer is authorized to perform certain software production activities. It is contemplated that the software producer's license will be issued by some commonly-trusted authority established by the computer software industry. Before issuing an license to a software producer, this authority performs a defined process to authenticate the person or company, and to verify their job skill; as a state does before issuing a contractor's license. For convenience, in this Specification, this commonly-trusted entity is referred to as the Software Publishing Authority ("SPA").

A software producer's license contains the following information:

the producer's name;

the license's issue date;

the license's expiration date;

the producer's public key;

the name of the issuing authority, SPA; and

the SPA's digital signature.

A software producer's license associates an application writer with a name and a public key. It enables a software producer to produce multiple products, and to sign every product produced. The public key embedded in a license belongs to the person who owns the license. This public key can later be used by any third party to verify the producer's digital signature. A user who has purchased a product can determine the genuineness of a product by using the public key embedded in the producer's identification to authenticate the digital signature.

The SPA's digital signature is generated by computing the message digest of the producer's identification and encrypting the message digest using the SPA's private key. Since the SPA's private key is kept private to the SPA, third parties are not able to easily forge the SPA's signature to produce a fake identification.

In accordance with the teachings of the present invention, a software application writer ("SW") supplies three major pieces of information to a compiler prior to compilation of the code:

the source code written by the application writer;

the application writer's private key; and

the application writer's license.

The code included in a passport may comprise source code in various computer languages, assembly code, machine binary code, or data. The code may be stored in various formats. For example, a piece of source code may be stored in a clear text form in the passport. A portion of binary executable machine code may also be stored in a compacted format in the passport, using certain well known compaction algorithms such as Huffman encoding. The format used in a particular implementation is indicated by a flag in the passport.

Binary executable code may further be stored in a printable-character set format to allow the passport to be printed. A user would then reverse the printable-format to recover the software. Moreover, code protected by intellectual property, such as copyright or patent, may be stored in an encrypted format in the passport. In such case, it is contemplated that a user may be required to pay a license fee prior to gaining access to the software.

Referring now to FIG. **4**, to generate the software passport of the present invention, the original source code **20**, the application writer's private key **22**, and the application writer's license **24** is provided to a compiler **26**. As illustrated, the application writer's license **24** includes the writer's name **30**, the writer's public key **32** and a validity date **34**.

The compiler **26** then compiles the source code **20** into binary code. The compiler **26** further computes the message digest of the binary code, and encrypts the message digest using the private key **22** supplied by the application writer. This encrypted message digest constitutes the application writer's signature.

A digital signature of the application writer is produced and embedded in the passport. The compiler **26** also embeds the application writer's license **24** in the passport. The application writer's license **24** allows any user who has purchased the product to recognize the maker of the product. The application writer's digital signature in the passport allows any user to verify the genuineness of the product. The SPA's digital signature in the application writer's license **24** provides the user with the ability to verify that an application writer is a licensed application writer by using SPA's public key to encrypt the signature.

As shown in FIG. **4**, the generated software passport **38**, including the application code is then distributed using any desired software distribution model. The passport **38** is received by a user and is executed using an operating system (OS) running on a computer system ("platform") such as the system of FIG. **1**.

Referring now to FIG. **5**, the use of the present invention by platform builders will be described. In the electronic game industry and the interactive television cable set-top box industry, platform producers often desire to allow only authorized code to be executed on their particular platform. To be able to control the accessibility of a platform, the received code must be identifiable and the platform must be able to identify the software when it arrives. As illustrated in FIG. **5**, the present invention may be applied in a platform producer licensing scheme with particular application for use in settop box and video game environments.

Referring now to FIGS. **6***a* and **6***b*, a platform producer may issue a "programmer's license" to a set of application

Nintendo - Ancora Exh. 1003

writers (alternatively referred to as "software writers") who are authorized to write application code for a particular platform. A programmer's license issued by a platform producer is similar to the programmer's identification issued by the SPA, except that the license is digitally signed by the platform producer instead of by the SPA. The programmer's license contains the following information:

the producer's name;

the issue license data;

the license expiration date;

the producer's public key;

the issuing authority (the platform producer); and

the platform producer's digital signature.

The platform producer's digital signature is generated by computing the message digest of the license, and encrypting the message digest using the platform producer's private key.

The software produced by a licensed application writer will include a valid passport **50** (see FIGS. **5** and **6**(*a*) which contains a genuine writer's digital signature, and a valid application writer's license **52** issued by the platform builder. Any application writer who is not authorized by the platform builder will not possess a valid license. Therefore, the software passport generated by an unauthorized person will either have no valid license or no valid signature.

The public key **54** of the platform builder is embedded in the platform (e.g., video game) for the verification process. At execution time, the platform extracts the public key **54** embedded in the system to verify that a passport contains a valid application writer's license **52**. The digital signature in the application writer's license is generated by computing the message digest of the license **52** and encrypting the message digest using the platform builder's private key. The system of the present invention can thus recover the original message digest by decrypting the signature using the platform builder's public key **54**. The verification process of the application writer's license may be accomplished by:

1. recomputing the message digest of the application license **52** in the passport **50**,
2. recovering the original message digest, and
3. comparing the old digest with the newly computed digest.

The passport **50** contains a valid application writer's license if the two message digests are the same. Otherwise the license is not valid. The verification process of the present invention is illustrated in the flow chart of FIG. **6**(*a*).

It will be appreciated that even if the passport **50** does contain a valid application writer's license, the application writer might have stolen the license by copying it from some other authorized writer's passport. In this case, the unauthorized writer would not have a correct private key **58** to forge the signature of the authorized writer. It is contemplated that the system will further verify the signature of the application writer **60**. It will be recalled that the application writer's digital signature in the passport was generated by computing the message digest of the passport and encrypting the message digest using the application writer's private key **58**. The original message digest may be recovered by decrypting the signature using the writer's public key **62** embedded in the application writer's license **52**, which is embedded in the passport **50**. The application writer's digital signature may then be verified by:

1. recomputing the message digest of the passport **50**,
2. recovering the original message digest, and
3. comparing the old digest with the new digest.

The signature is valid if the two message digests are the same. Otherwise the passport is not valid and the platform will reject the execution of the software. The steps executed by the present invention to verify the application writer's digital signature are illustrated in flow chart for FIG. **6**(*b*).

It will be further noted that the security scheme of the present invention may be used to protect inventions and authorship protected by intellectual property, such as copyrights and patents. The one additional procedure that is added to protect intellectual property is that the compiler (e.g. a compiler **68** shown in FIG. **5**) generates encrypted byte codes. When a user attempts to run the code on the platform operating system ("OS") the verification procedures are followed as described above with reference to FIGS. **6**(*a*) and **6**(*b*). However, with the code encrypted, the operating system requires an additional approval before it is permitted to run the code. A cryptographic key is required which essentially results in an IP license to run the code. After authenticating the code, the operating system requests the IP license. The operating system verifies that the IP license is signed by the person who authored the code, and then proceeds to decrypt and execute the code. A further feature of the present invention is that third parties do not have the ability to inspect the code since it is encrypted.

Accordingly, the present invention has disclosed a method and apparatus for enhancing software security. Although the present invention has been described with reference to FIGS. **1**–**6**, it will be apparent that may alternatives, modifications and variations may be made in light of the foregoing description.

APPENDIX A

THE FIRST TEN YEARS OF PUBLIC-KEY CRYPTOGRAPHY

WHITFIELD DIFFIE

Invited Paper

Public-key cryptosystems separate the capacities for encryption and decryption so that 1) many people can encrypt messages in such a way that only one person can read them, or 2) one person can encrypt messages in such a way that many people can read them. This separation allows important improvements in the management of cryptographic keys and makes it possible to 'sign' a purely digital message.

Public key cryptography was discovered in the Spring of 1975 and has followed a surprising course. Although diverse systems were proposed early on, the ones that appear both practical and secure today are all very closely related and the search for new and different ones has met with little success. Despite this reliance on a limited mathematical foundation public-key cryptography is revolutionizing communication security by making possible secure communication networks with hundreds of thousands of subscribers.

Equally important is the impact of public key cryptography on the theoretical side of communication security. It has given cryptographers a systematic means of addressing a broad range of security objectives and pointed the way toward a more theoretical approach that allows the development of cryptographic protocols with proven security characteristics.

I. INITIAL DISCOVERIES

Public key cryptography was born in May 1975, the child of two problems and a misunderstanding.

First came the problem of key distribution. If two people who have never met before are to communicate privately using conventional cryptographic means, they

must somehow agree in advance on a key that will be known to themselves and to no one else.

The second problem, apparently unrelated to the first, was the problem of signatures. Could a method be devised that would provide the recipient of a purely digital electronic message with a way of demonstrating to other people that it had come from a particular person, just as a written signature on a letter allows the recipient to hold the author to its contents?

On the face of it, both problems seem to demand the impossible. In the first case, if two people could somehow communicate a secret key from one to the other without ever having met, why could they not communicate their message in secret? The second is no better. To be effective, a signature must be hard to copy. How then can a digital message, which can be copied perfectly, bear a signature?

The misunderstanding was mine and prevented me from rediscovering the conventional key distribution center. The virtue of cryptography, I reasoned, was that, unlike any other known security technology, it did not require trust in any party not directly involved in the communication, only trust in the cryptographic systems. What good would it do to develop impenetrable cryptosystems, I reasoned, if their users were forced to share their keys with a key distribution center that could be compromised by either burglary or subpoena.

The discovery consisted not of a solution, but of the recognition that the two problems, each of which seemed unsolvable by definition, could be solved at all and that the solutions to both problems came in one package.

First to succumb was the signature problem. The conventional use of cryptography to authenticate messages had been joined in the 1950s by two new applications, whose functions when combined constitute a signature.

Beginning in 1952, a group under the direction of Horst Feistel at the Air Force Cambridge Research Center began to apply cryptography to the military problem of distinguishing friendly from hostile aircraft. In traditional Identification Friend or Foe systems, a fire control radar determines the identity of an aircraft by challenging it, much as a sentry challenges a soldier on foot. If the airplane returns the correct identifying information, it is judged to be friendly, otherwise it is thought to be hostile or at best neutral. to allow the correct response to remain constant for any significant period of time, however, is to invite opponents to record a legitimate friendly response and play it back whenever they themselves are challenged. The approach taken by Feistel's group, and now used in the MK XII IFF system, is to vary the exchange cryptographically from encounter to encounter. The radar sends a randomly selected challenge and judges the aircraft by whether it receives a correctly encrypted response. Because the challenges are never repeated, previously recorded responses will not be judged correct by a challenging radar.

Later in the decade, this novel authentication technique was joined by another, which seems first to have been applied by Roger Needham of Cambridge University [112]. This time the problem was protecting computer passwords. Access control systems often suffer from the extreme sensitivity of their password tables. The tables gather all of the passwords together in one place and anyone who gets access to this information can impersonate any of the system's users. To guard against this possibility, the password table is filled not with the passwords themselves, but with the images of the passwords under a one-way function. A one-way function is easy to compute, but difficult to invert. For any password, the correct table entry can be calculated easily. Given an output from the one-way function, however, it is exceedingly difficult to find any input that will produce it. This reduces the value of the password table to an intruder tremendously, since its entries are not passwords and are not acceptable to the password verification routine.

Challenge and response identification and one-way functions provide protection against two quite different sorts of threats. Challenge and response identification resists the efforts of an eavesdropper who can spy on the communication channel. Since the challenge varies randomly from event to event, the spy is unable to replay it and fool the challenging radar. There is, however, no protection against an opponent who captures the radar and learns its cryptographic keys. This opponent can use what he has learned to fool any other radar that is keyed the same. In contrast, the one-way function defeats the efforts of an intruder who captures the system password table (analogous to capturing the radar) but succumbs to anyone who intercepts the login message because the password does not change with time.

I realized that the two goals might be achieved simultaneously if the challenger could pose questions that it was unable to answer, but whose answers it could judge for correctness. I saw the solution as a generalization of the one-way function: a trap-door one-way function that allowed someone in possession of secret information to go backwards and compute the function's inverse. The challenger would issue a value in the range of the one-way function and demand to know its inverse. Only the person who knew the trapdoor would be able to find the corresponding element in the domain, but the challenger, in possession of an algorithm for computing the one-way function, could readily check the answer. In the applications that later came to seem most important, the role of the challenge was played by a message and the process took on the character of a signature, a digital signature.

It did not take long to realize that the trap-door one-way function could also be applied to the baffling problem of key distribution. For someone in possession of the forward form of the one-way function to send a secret message to the person who knew the trapdoor, he had only to transform the message with the one-way function. Only the holder of the trap-door information would be able to invert the operation and recover the message. Because knowing the forward form of the function did not make it possible to compute the inverse, the function could be made freely available. It is this possibility that gave the field its name: public-key cryptography.

The concept that emerges is that of a public-key cryptosystem: a cryptosystem in which keys come in inverse pairs [36] and each pair of keys has two properties.

Anything enclosed with one key can be decrypted with the other.

Given one member of the pair, the public key, it is infeasible to discover the other, the secret key.

This separation of encryption and decryption makes it possible for the subscribers to a communication system to list their public keys in a "telephone directory" along with their names and addresses. This done, the solutions to the original problems can be achieved by simple protocols.

One subscriber can send a private message to another simply by looking up the addressee's public key and using it to encrypt the message. Only the holder of the corresponding secret key can read such a message; even the sender, should he lose the plaintext, is incapable of extracting it from the ciphertext.

A subscriber can sign a message by encrypting it with his own secret key. Anyone with access to the public key

Nintendo - Ancora Exh. 1003

can verify that it must have been encrypted with the corresponding secret key, but this is of no help to him in creating (forging) a message with this property.

The first aspect of public-key cryptography greatly simplifies the management of keys, especially in large communication networks. In order for a pair of subscribers to communicate privately using conventional end-to-end cryptography, they must both have copies of the same cryptographic key and this key must be kept secret from anyone they do not wish to take into their confidence. If a network has only a few subscribers, each person simply stores one key for every other subscriber against the day he will need it, but for a large network, this is impractical.

In a network with n subscribers there are $n(n-1)/2$ pairs, each of which may require a key. This amounts to five thousand keys in a network with only a hundred subscribers, half a million in a network with one thousand, and twenty million billion in a network the size of the North American telephone system. It is unthinkable to distribute this many keys in advance and undesirable to postpone secure communication while they are carried from one party to the other by courier.

The second aspect makes it possible to conduct a much broader range of normal business practices over a telecommunication network. The availability of a signature that the receiver of a message cannot forge and the sender cannot readily disavow makes it possible to trust the network with negotiations and transactions of much higher value than would otherwise be possible.

It must be noted that both problems can be solved without public-key cryptography, but that conventional solutions come at a great price. Centralized key distribution centers can on request provide a subscriber with a key for communicating with any other subscriber and protocols for this purpose will be discussed later on. The function of the signature can also be approximated by a central registry that records all transactions and bears witness in cases of dispute. Both mechanisms, however, encumber the network with the intrusion of a third party into many conversations, diminishing security and degrading performance.

At the time public-key cryptography was discovered, I was working with Martin Hellman in the Electrical Engineering Department at Stanford University. It was our immediate reaction, and by no means ours alone, that the problem of producing public-key cryptosystems would be quite difficult. Instead of attacking this problem in earnest, Marty and I forged ahead in examining the consequences.

The first result of this examination to reach a broad audience was a paper entitled "Multi-User Cryptographic Techniques" [35], which we gave at the National Computer Conference in 1976. We wrote the paper in December 1975 and sent preprints around immediately. One of the preprints went to Peter Blatman, a Berkeley graduate student and friend since childhood of cryptography's historian David Kahn. The result was to bring from the woodwork Ralph Merkle, possibly the single most inventive character in the public-key saga.

Merkle's Puzzles

Ralph Merkle had registered in the Fall of 1974 for Lance Hoffman's course in computer security at U.C. Berkeley. Hoffman wanted term papers and required each student to submit a proposal early in the term. Merkle addressed the problem of public-key distribution or as he called it "Secure Communication over Insecure Channels" [70]. Hoffman could not understand Merkle's proposal. He demanded that it be rewritten, but alas found the revised version no more comprehensible than the original. After one more iteration of this process, Merkle dropped the course, but he did not cease working on the problem despite continuing failure to make his results understood.

Although Merkle's original proposal may have been hard to follow, the idea is quite simple. Merkle's approach is to communicate a cryptographic key from one person to another by hiding it in a large collection of puzzles. Following the tradition in public-key cryptography the parties to this communication will be called Alice and Bob rather than the faceless A and B, X and Y, or I and J, common in technical literature.

Alice manufactures a million or more puzzles and sends them over the exposed communication channel to Bob. Each puzzle contains a cryptographic key in a recognizable standard format. The puzzle itself is a cryptogram produced by a block cipher with a fairly small key space. As with the number of puzzles, a million is a plausible number. When Bob receives the puzzles, he picks one and solves it, by the simple expedient of trying each of the block cipher's million keys in turn until he finds one that results in plaintext of the correct form. This requires a large but hardly impossible amount of work.

In order to inform Alice which puzzle he has solved, Bob uses the key it contains to encrypt a fixed test message, which he transmits to Alice. Alice now tries her million keys on the test message until she finds the one that works. This is the key from the puzzle Bob has chosen.

The task facing an intruder is more arduous. Rather than selecting one of the puzzles to solve, he must solve on average half of them. The amount of effort he must expend is therefore approximately the square of that expended by the legitimate communicators.

The n to n^2 advantage the legitimate communicators have over the intruder is small by cryptographic standards, but sufficient to make the system plausible in some circumstances. Suppose, for example, that the plaintext of each puzzle is 96 bits, consisting of 64 bits of key together with a thirty-two bit block of zeros that enables Bob to recognize the right solution. The puzzle is constructed by encrypting this plaintext using a block cipher with 20 bits of key. Alice produces a million of these puzzles and Bob requires about half a million tests to solve one. The bandwidth and computing power required to make this feasible are large but not inaccessible. On a DS1 (1.544 Mbit) channel it would require about a minute to communicate the puzzles. If keys can be tried on the selected puzzle at about ten-thousand per second, it will take Bob another minute to solve it. Finally, it will take a similar amount of time for Alice to figure out, from the test message, which key has been chosen.

The intruder can expect to have to solve half a million puzzles at half a million tries apiece. With equivalent computational facilities, this requires twenty-five million seconds or about a year. For applications such as authentication, in which the keys are no longer of use after communication is complete, the security of this system might be sufficient.

When Merkle saw the preprint of "Multi-User Cryptographic Techniques" he immediately realized he had found people who would appreciate his work and sent us copies of the paper he had been endeavoring unsuccessfully to publish. We in turn realized that Merkle's formulation of the problem was quite different from mine and, because Merkle had isolated one of the two intertwined problems I had seen, potentially simpler.

Even before the notion of putting trap-doors into one-way functions had appeared, a central objective of my work with Marty had been to identify and study functions that were

Nintendo - Ancora Exh. 1003

easy to compute in one direction, but difficult to invert. Three principal examples of this simplest and most basic of cryptographic phenomena occupied our thoughts.

- John Gill, a colleague in the Electrical Engineering Department at Stanford, had suggested discrete exponentiation because the inverse problem, discrete logarithm, was considered very difficult.
- I had sought suitable problems in the chapter on NP-complete functions in Aho, Hopcroft, and Ullman's book on computational complexity [3] and selected the knapsack problem as most appropriate.
- Donald Knuth of the Stanford Computer Science Department had suggested that multiplying a pair of primes was easy, but that factoring the result, even when it was known to have precisely two factors, was exceedingly hard.

All three of these one-way functions were shortly to assume great importance.

II. EXPONENTIAL KEY EXCHANGE

The exponential example was tantalizing because of its combinatorial peculiarities. When I had first thought of digital signatures, I had attempted to achieve them with a scheme using tables of exponentials. This system failed, but Marty and I continued twisting exponentials around in our minds and discussions trying to make them fit. Marty eventually made the breakthrough early one morning in May 1976. I was working at the Stanford Artificial Intelligence Laboratory on the paper that we were shortly to publish under the title "New Directions in Cryptography" [36] when Marty called and explained exponential key exchange in its unnerving simplicity. Listening to him, I realized that the notion had been at the edge of my mind for some time, but had never really broken through.

Exponential key exchange takes advantage of the ease with which exponentials can be computed in a Galois (finite) field GF(q) with a prime number of q of elements (the numbers {0, 1, . . . , q−1} under arithmetic modulo q) as compared with the difficulty of computing logarithms in the same field. If

$$Y=\alpha^x \bmod q, \text{ for } 1<X<q-1$$

where α is a fixed primitive element of GF(q) (that is the powers of α produce all the nonzero elements 1, 2, . . . , q−1 of GF(q)), then X is referred to as the logarithm of Y to the base α, over GF(q):

$$X=\log_\alpha Y \text{ over } GF(q), \text{ for } 1<Y<q-1,$$

Calculation of Y from X is easy: Using repeated squaring, it takes at most $2\times\log_2 q$ multiplications. For example

$$\alpha^{37}=\alpha^{32+4+1}=((((\alpha^2)^2)^2)^2)^2\times(\alpha^2)^2\times\alpha.$$

Computing X from Y, on the other hand, is typically far more difficult [104], [83], [29]. If q has been chosen correctly, extracting logarithms modulo q requires a precomputation proportional to

$$L(q)=e^{\sqrt{\ln q \ln\ln q}}$$

though after that individual logarithms can be calculated fairly quickly. The function L(q) also estimates the time needed to factor a composite number of comparable size and will appear again in that context.

To initiate communication Alice chooses a random number X_A uniformly from the integers 1, 2, . . . , q−1. She keeps X_A secret, but sends

$$Y_A=\alpha^{X_A} \bmod q$$

to Bob. Similarly, Bob chooses a random number X_B and sends the corresponding Y_B to Alice. Both Alice and Bob can now compute

$$K_{AB}=\alpha^{X_A X_B} \bmod q$$

and use this as their key. Alice computes K_{AB} by raising the Y_B she obtained from Bob to the power X_A

$$\begin{aligned} K_{AB} &= Y_B^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q \\ &= \alpha^{X_B X_A} = \alpha^{X_A X_B} \bmod q \end{aligned}$$

and Bob obtains K_{AB} in a similar fashion

$$K_{AB}=Y_B^{X_B} \bmod q.$$

No one except Alice and Bob knows either X_A or X_B so anyone else must compute K_{AB} from Y_A and Y_B alone. The equivalence of this problem to the discrete logarithm problem is a major open question in public-key cryptography. To date no easier solution than taking the logarithm of either Y_A or Y_B has been discovered.

If q is a prime about 1000 bits in length, only about 2000 multiplications of 1000-bit numbers are required to compute Y_A from X_A, or K_{AB} from Y_A and X_B. Taking logarithms over GF(q), on the other hand, currently demands more than 2^{100} (or approximately 10^{30}) operations.

The arithmetic of exponential key exchange is not restricted to prime fields; it can also be done in Galois Fields with 2^n elements, or in prime product rings [103], [68]. The '2^n' approach has been taken by several people [64], [117], [56] because arithmetic in these fields can be performed with linear shift registers and is much faster than arithmetic over large primes. It has turned out, however, that discrete logarithms can also be calculated much more quickly in '2^n' fields and so the sizes of the registers must be about 50 percent greater.

Marty and I immediately recognized that we had a far more compact solution to the key distribution problem than Merkle's puzzles and hastened to add it to both the upcoming National Computer Conference presentation and to "New Directions." The latter now contained a solution to each aspect of the public-key problem, though not the combined solution I had envisioned. It was sent off to the IEEE TRANSACTIONS ON INFORMATION THEORY prior to my departure for NCC and like all of our other papers was immediately circulated in preprint.

III. TRAP-DOOR KNAPSACKS

Later in the same year, Ralph Merkle began work on his best known contribution to public-key cryptography: building trapdoors into the knapsack one-way function to produce the trap-door knapsack public-key cryptosystem.

The knapsack problem is fancifully derived from the notion of packing gear into a knapsack. A shipping clerk faced with an odd assortment of packages and a freight container will naturally try to find a subset of the packages that fills the container exactly with no wasted space. The simplest case of this problem, and the one that has found application in cryptography is the one dimensional case: packing varying lengths of fishing rod into a tall thin tube.

Given a cargo vector of integers $a=(a_1, a_2, \ldots, a_n)$ it is easy to add up the elements of any specified subvector. Presented with an integer S, however, it is not easy to find a subvector of a whose elements sum to S, even if such a

subvector is known to exist. This knapsack problem is well known in combinatorics and is believed to be extremely difficult in general. It belongs to the class of NP-complete problems, problems thought not to be solvable in polynomial time on any deterministic computer.

I had previously identified the knapsack problem as a theoretically attractive basis for a one-way function. The cargo vector a can be used to encipher an n-bit message $x=(x_1, x_2, \ldots, x_n)$ by taking the dot product $S=a\cdot x$ as the ciphertext. Because one element of the dot product is binary, this process is easy and simply requires n additions. Inverting the function by finding a binary vector x such that $a\cdot x=S$ solves the knapsack problem and is thus believed to be computationally infeasible if a is randomly chosen. Despite this difficulty in general, many cases of the knapsack problem are quite easy and Merkle contrived to build a trapdoor into the knapsack one-way function by starting with a simple cargo vector and converting it into a more complex form [71].

If the cargo vector a is chosen so that each element is larger than the sum of the preceding elements, it is called superincreasing and its knapsack problem is particularly simple. (In the special case where the components are 1, 2, 4, 8, etc., this is the elementary operation of binary decomposition.) For example, if a'=(171, 197, 459, 1191, 2410) and S'=3798 then x_5 must equal 1. If it were 0 then even if x_1, x_2, x_3, and x_4 were all equal to 1, the dot product $a\cdot x$ would be too small. Since $x_5=1$, $S'-a'_5=3797-2410=1387$ must be a sum of a subset of the first four elements of a'. The fact that $1387>a'_4=1191$ means that x_4 too must equal 1. Finally $S'-a'_5-a'_4=196=a'_2$ so $x_3=0$, $x_2=1$, and $x_1=0$.

The simple cargo vector a' cannot be used as a public enciphering key because anyone can easily recover a vector x for which $x\cdot a'=S'$ from a' and S' by the process described above. The algorithm for generating keys therefore chooses a random superincreasing cargo vector a' (with a hundred or more components) and keeps this vector secret. It also generates a random integer m, larger than $\Sigma a'$, and a random integer w, relatively prime to m, whose inverse w^{-1} mod m will be used in decryption. The public cargo vector or enciphering key a is produced by multiplying each component of a' by w mod m

$$a=wa' \bmod m.$$

Alice publishes a transposed version of a as her public key, but keeps the transposition, the simple cargo vector a', the multiplier w and its inverse, and the modulus m secret as her private key.

When Bob wants to send the message x to Alice he computes and sends

$$S=a\cdot x.$$

Because

$$\begin{aligned} S' &= w^{-1}S \bmod m \\ &= w^{-1}\Sigma a_i x_i \bmod m \\ &= w^{-1}\Sigma (wa'_i \bmod m)x_i \bmod m \\ &= \Sigma (w^{-1}wa'_i \bmod m)x_i \bmod m \\ &= \Sigma a'_i x_i \bmod m \\ &= a'\cdot x \end{aligned}$$

when $m>\Sigma a'$, Alice can use her secret information, w^{-1} and m, to transform any message S that has been enciphered with her public key into $S'=w^{-1}\times S$ and solve the easy knapsack problem $S'=a'\cdot x$ to obtain x.

For example, for the secret vector a', above the values w=2550 and m=8443, result in the public vector a=(5457, 4213, 5316, 6013, 7439), which hides the structure present in a'.

This process can be iterated to produce a sequence of cargo vectors with more and more difficult knapsack problems by using transformations (w_1, m_1), (w_2, m_2), etc. The overall transformation that results is not, in general, equivalent to any single (w, m) transformation.

The trap-door knapsack system does not lend itself readily to the production of signatures because most elements S of the ciphertext space $\{0\leqq S\leqq \Sigma a_i\}$, do not have inverse images. This does not interfere with the use of the system for sending private messages, but requires special adaptation for signature application [71], [98]. Merkle had great confidence in even the single iteration knapsack system and posted a note on his office offering a $100 reward to anyone who could break it.

IV. The RSA System

Unknown to us at the time we wrote "New Directions" were the three people who were to make the single most spectacular contribution to public-key cryptography: Ronald Rivest, Adi Shamir, and Leonard Adleman. Ron Rivest had been a graduate student in computer science at Stanford while I was working on proving the correctness of programs at the Stanford Artificial Intelligence Laboratory. One of my colleagues in that work was Zohar Manna, who shortly returned to Israel and supervised the doctoral research of Adi Shamir, at the Weitzman Institute. Len Adleman was a native San Franciscan with both undergraduate and graduate degrees from U.C. Berkeley. Despite this web of near connections, not one of the three had previously crossed our paths and their names were unfamiliar.

When the New Directions paper reached MIT in the fall of 1976, the three took up the challenge of producing a full-fledged public-key cryptosystem. The process lasted several months during which Rivest proposed approaches, Adleman attacked them, and Shamir recalls doing some of each.

In May 1977 they were rewarded with success. After investigating a number of possibilities, some of which were later put forward by other researchers [67], [1], they had discovered how a simple piece of classical number theory could be made to solve the problem. The resulting paper [91] also introduced Alice and Bob, the first couple of cryptography [53].

The RSA cryptosystem is a block cipher in which the plaintexts and ciphertexts are integers between 0 and N−1 for some N. It resembles the exponential key exchange system described above in using exponentiation in modular arithmetic for its enciphering and deciphering operations but, unlike that system, RSA must do its arithmetic not over prime numbers, but over composite ones.

Knowledge of plaintext M, a modulus N, and an exponent e are sufficient to allow calculation of M^e mod N. Exponentiation, however, is a one-way function with respect to the extraction of roots as well as logarithms. Depending on the characteristics of N, M, and e, it may be very difficult to invert.

The RSA system makes use of the fact that finding large (e.g., 200 digit) prime numbers is computationally easy, but that factoring the product of two such numbers appears computationally infeasible. Alice creates her secret and public keys by selecting two very large prime numbers, P and Q, at random, and multiplying them together to obtain a bicomposite modulus N. She makes this product public together with a suitably chosen enciphering exponent e, but keeps the factors, P and Q secret.

Nintendo - Ancora Exh. 1003

The enciphering process of exponentiation modulo N can be carried out by anyone who knows N, but only Alice, who knows the factors of N, can reverse the process and decipher.

Using P and Q, Alice can compute the Euler totient function $\phi(N)$, which counts the number of integers between 1 and N that are relatively prime to N and consequently invertible in arithmetic modulo N. For a bicomposite number this is

$$\phi(N)=(P-1)(Q-1).$$

The quantity $\phi(N)$ plays a critical role in Euler's theorem, which says that for any number x that is invertible modulo N (and for large N that is almost all of them)

$$(X^{\phi(N)} \equiv 1 \pmod N)$$

or slightly more generally

$$x^{k\phi(N)+1} \equiv x \pmod N.$$

Using $\phi(N)$ Alice can calculate [60] a number d such that

$$e \times d \equiv x \pmod{\phi(N)}$$

which is equivalent to saying that

$$e \times d = k \times \phi(N)+1.$$

When the cryptogram M^e mod N is raised to the power d the result is

$$(M^e)^d = M^{ed} = M^{k\phi(N)+1} \equiv M \pmod N$$

the original plaintext M.

As a very small example, suppose P=17 and Q=31 are chosen so that N=PQ=527 and $\phi(N)=(P-1)(Q-1)=480$. If e=7 is chosen then d=343. (7×343=2401=5×480+1). And if M=2 then

$$C=M^e \bmod N=2^7 \bmod 527=128.$$

Note again that only the public information (e, N) is required for enciphering M. To decipher, the private key d is needed to compute

$$\begin{aligned} M &= C^d \bmod N \\ &= 128^{343} \bmod 527 \\ &= 128^{256} \times 128^{64} \times 128^{16} \times 128^{4} \times 128^{2} \times 128^{1} \bmod 527 \\ &= 35 \times 256 \times 35 \times 101 \times 47 \times 128 \bmod 527 \\ &= 2 \bmod 527. \end{aligned}$$

Just as the strength of the exponential key exchange system is not known to be equivalent to the difficulty of extracting discrete logarithms, the strength of RSA has not been proven equivalent to factoring. There might be some method of taking the eth root of M^e without calculating d and thus without providing information sufficient to factor. While at MIT in 1978, M. O. Rabin [86] produced a variant of RSA, subsequently improved by Hugh Williams of the University of Manitoba [113], that is equivalent to factoring. Rivest and I have independently observed [38], [92], however, that the precise equivalence Rabin has shown is a two-edged sword.

V. THE McELIECE CODING SCHEME

Within a short time yet another public-key system was to appear, this due to Robert J. McEliece of the Jet Propulsion Laboratory at Cal Tech [69]. McEliece's system makes use of the existence of a class of error correcting codes, the Goppa codes, for which a fast decoding algorithm is known. His idea was to construct a Goppa code and disguise it as a general linear code, whose decoding problem is NP-complete. There is a strong parallel here with the trapdoor knapsack system in which a superincreasing cargo vector, whose knapsack problem is simple to solve, is disguised as a general cargo vector whose knapsack problem is NP-complete.

In a knapsack system, the secret key consists of a superincreasing cargo vector v, together with the multiplier w and the modulus m that disguise it; in McEliece's system, the secret key consists of the generator matrix G for a Goppa code together with a nonsingular matrix S and a permutation matrix P that disguise it. The public key appears as the encoding matrix G'=SGP of a general linear code.

To encode a data block u into a message x, Alice multiplies it by Bob's public encoding matrix G' and adds a locally generated noise block Z.

To decode, Bob multiplies the received message x by p^{-1}, decodes xp^{-1} to get a word in the Goppa code and multiplies this by S^{-1} to recover Alice's data block.

McEliece's system has never achieved wide acceptance and has probably never even been considered for implementation in any real application. This may be because the public keys are quite large, requiring on the order of a million bits; it may be because the system entails substantial expansion of the data; or it may be because McEliece's system bears a frightening structural similarity to the knapsack systems whose fate we shall discover shortly.

VI. THE FALL OF THE KNAPSACKS

Nineteen eighty-two was the most exciting time for public-key cryptography since its spectacular first three years. In March, Adi Shamir sent out a research announcement: He had broken the single iteration Merkle-Hellman knapsack system [101], [102]. By applying new results of Lenstra at the Mathematische Centrum in Amsterdam, Shamir had learned how to take a public cargo vector and discover a w' and m' that would convert it back into a superincreasing "secret" cargo vector—not necessarily the same one the originator had used, but one that would suffice for decrypting messages encrypted with the public cargo vector.

Shamir's original attack was narrow. It seemed that perhaps its only consequence would be to strengthen the knapsack system by adding conditions to the construction rules for avoiding the new attack. The first response of Gustavus J. Simmons, whose work will dominate a later section, was that he could avoid Shamir's attack without even changing the cargo vector merely by a more careful choice of w and m [16]. He quickly learned, however, that Shamir's approach could be extended to break a far larger class of knapsack systems [16].

Crypto '82 revealed that several other people had continued down the trail Shamir had blazed. Shamir himself had reached the same conclusions. Andy Odlyzko and Jeff Lagarias at Bell Labs were on the same track and Len Adleman had not only devised an attack but programmed it on an Apple II. The substance of the attacks will not be treated here since it is central to another paper in this special section (E. F. Brickell and A. M. Odlyzko "Cryptanalysis: A Survey of Recent Results"). The events they engendered, however, will.

I had the pleasure of chairing the cryptanalysis session at Crypto '82 in which the various results were presented. Ironically, at the time I accepted the invitation to organize such a session, Shamir's announcement stood alone and knapsack systems were only one of the topics to be dis-

Nintendo - Ancora Exh. 1003

cussed. My original program ran into very bad luck, however. Of the papers initially scheduled only Donald Davies's talk on: "The Bombe at Bletchley Park," was actually presented. Nonetheless, the lost papers were more than replaced by presentations on various approaches to the knapsack problem.

Last on the program were Len Adleman and his computer, which had accepted a challenge on the first night of the conference. The hour passed; various techniques for attacking knapsack systems with different characteristics were heard; and the Apple II sat on the table waiting to reveal the results of its labors. At last Adleman rose to speak mumbling something self-deprecatingly about "the theory first, the public humiliation later" and beginning to explain his work. All the while the figure of Carl Nicolai moved silently in the background setting up the computer and copying a sequence of numbers from its screen onto a transparency. At last another transparency was drawn from a sealed envelope and the results placed side by side on the projector. They were identical. The public humiliation was not Adleman's, it was knapsack's.

Ralph Merkle was not present, but Marty Hellman, who was, gamely arose to make a concession speech on their behalf. Merkle, always one to put his money where his mouth was, had long since paid Shamir the $100 in prize money that he had placed on the table nearly six years before.

The press wrote that knapsacks were dead. I was skeptical but ventured that the results were sufficiently threatening that I felt "nobody should entrust anything of great value to a knapsack system unless he had a much deeper theory of their functioning than was currently available." Nor was Merkle's enthusiasm dampened. He promptly raised his bet and offered $1000 to anyone who could break a multiple iteration knapsack [72].

It took two years, but in the end, Merkle had to pay [42]. The money was finally claimed by Ernie Brickell in the summer of 1984 when he announced the destruction of a knapsack system of forty iterations and a hundred weights in the cargo vector in about an hour of Cray-1 time [17]. That Fall I was forced to admit: "knapsacks are flat on their back."

Closely related techniques have also been applied to make a dramatic reduction in the time needed to extract discrete logarithms in fields of type $GF(2^n)$. This approach was pioneered by Blake, Fuji-Hara, Vanstone, and Mullin in Canada [10] and refined by Coppersmith in the U.S. [28]. A comprehensive survey of this field was given by Andy Odlyzko at Eurocrypt '84 [79].

VII. EARLY RESPONSES TO PUBLIC KEY

A copy of the MIT report [90] on the RSA cryptosystem was sent to Martin Gardner, Mathematical Games editor of Scientific American, shortly after it was printed. Gardner promptly published a column [48] based on his reading of both the MIT report and "New Directions." Bearing the title: "A New Kind of Cryptosystem That Would Take Millions of Years to Break," it began a confusion that persists to this day between the two directions explored by the "New Directions" paper: public-key cryptography and the problem of proving the security of cryptographic systems. More significant, however, was the prestige that public-key cryptography got from being announced in the scientific world's most prominent lay journal more than six months before its appearance in the Communications of the ACM.

The excitement public-key cryptosystems provoked in the popular and scientific press was not matched by corresponding acceptance in the cryptographic establishment, however. In the same year that public-key cryptography was discovered, the National Bureau of Standards, with the support of the National Security Agency, proposed a conventional cryptographic system, designed by IBM, as a federal Data Encryption Standard [44]. Hellman and I criticized the proposal on the grounds that its key was too small [37], but manufacturers were gearing up to support the proposed standard and our criticism was seen by many as an attempt to disrupt the standards-making process to the advantage of our own work. Public key in its turn was attacked, in sales literature [74] and technical papers [76], [59] alike, more as though it were a competing product than a recent research discovery. This, however, did not deter NSA from claiming its share of the credit. Its director, in the words of the *Encyclopaedia Britannica* [110], "pointed out that two-key cryptography had been discovered at the agency a decade earlier," though no evidence for this claim was ever offered publicly.

Far from hurting public key, the attacks and counterclaims added to a ground swell of publicity that spread its reputation far faster than publication in scientific journals alone ever could. The criticism nonetheless bears careful examination, because the field has been affected as much by discoveries about how public key cryptosystems should be used as by discoveries about how they can be built.

In viewing public-key cryptography as a new form of cryptosystem rather than a new form of key management, I set the stage for criticism on grounds of both security and performance. Opponents were quick to point out that the RSA system ran about one thousandth as fast as DES and required keys about ten times as large. Although it had been obvious from the beginning that the use of public-key systems could be limited to exchanging keys for conventional cryptography, it was not immediately clear that this was necessary. In this context, the proposal to build hybrid systems [62] was hailed as a discovery in its own right.

At present, the convenient features of public-key cryptosystems are bought at the expense of speed. The fastest RSA implementations run at only a few thousand bits per second, while the fastest DES implementations run at many million. It is generally desirable, therefore, to make use of a hybrid in which the public-key systems are used only during key management processes to establish shared keys for employment with conventional systems.

No known theorem, however, says that a public-key cryptosystem must be larger and slower than a conventional one. The demonstrable restrictions mandate a larger minimum block size (though perhaps no larger than that of DES) and preclude use in stream modes whose chunks are smaller than this minimum. For a long time I felt that "high efficiency" public-key systems would be discovered and would supplant both current public key and conventional systems in most applications. Using public-key systems throughout, I argued, would yield a more uniform architecture with fewer components and would give the best possible damage limitation in the event of a key distribution center compromise [38]. Most important, I thought, if only one system were in use, only one certification study would be required. As certification is the most fundamental and most difficult problem in cryptography, this seemed to be where the real savings lay.

In time I saw the folly of this view. Theorems or not, it seemed silly to expect that adding a major new criterion to the requirements for a cryptographic system could fail to slow it down. The designer would always have more latitude with systems that did not have to satisfy the public key property and some of these would doubtless be faster. Even more compelling was the realization that modes of operation

Nintendo - Ancora Exh. 1003

incompatible with the public-key property are essential in many communication channels.

To date, the "high-efficiency public-key systems" that I had hoped for have not appeared and the restriction of public-key cryptography to key management and signature applications is almost universally accepted. More fundamental criticism focuses on whether public-key actually makes any contribution to security, but, before examining this criticism, we must undertake a more careful study of key distribution mechanisms.

Key Management

The solution to the problem of key management using conventional cryptography is for the network to provide a key distribution center (KDC): a trusted network resource that shares a key with each subscriber and uses these in a bootstrap process to provide additional keys to the subscribers as needed. When one subscriber wants to communicate securely with another, he first contacts the KDC to obtain a session key for use in that particular conversation.

Key distribution protocols vary widely depending on the cost of messages, the availability of multiple simultaneous connections, whether the subscribers have synchronized clocks, and whether the KDC has authority not only to facilitate, but to allow or prohibit, communications. The following example is typical and makes use of an important property of cryptographic authentication. Because a message altered by anyone who does not have the correct key will fail when tested for authenticity, there is no loss of security in receiving a message from the hands of a potential opponent. In so doing, it introduces, in a conventional context, the concept of a certificate—a cryptographically authenticated message containing a cryptographic key-a concept that plays a vital role in modern key management.

1) When Alice wants to call Bob, she first calls the KDC and requests a key for communicating with Bob.
2) The KDC responds by sending Alice a pair of certificates. Each contains a copy of the required session key, one encrypted so that only Alice can read it and one so that only Bob can read it.
3) When Alice calls Bob, she presents the proper certificate as her introduction. Each of them decrypts the appropriate certificate under the key that he shares with the KDC and thereby gets access to the session key.
4) Alice and Bob can now communicate securely using the session key.

Alice and Bob need not go through all of this procedure on every call; they can instead save the certificates for later use. Such cacheing of keys allows subscribers to avoid calling the KDC every time they pick up the phone, but the number of KDC calls is still proportional to the number of distinct pairs of subscribers who want to communicate securely. A far more serious disadvantage of the arrangement described above is that the subscribers must share the secrecy of their keying information with the KDC and if it is penetrated, they too will be compromised.

A big improvement in both economy and security can be made by the use of public-key cryptography. A certificate functions as a letter of introduction. In the protocol above, Alice has obtained a letter that introduces her to Bob and Bob alone. In a network using public-key encryption, she can instead obtain a single certificate that introduces her to any network subscriber [62].

What accounts for the difference? In a conventional network, every subscriber shares a secret key with the KDC and can only authenticate messages explicitly meant for him. If one subscriber has the key needed to authenticate a message meant for another subscriber, he will also be able to create such a message and authentication fails. In a public-key network, each subscriber has the public key of the KDC and thus the capacity to authenticate any message from the KDC, but no power to forge one.

Alice and Bob, each having obtained a certificate from the KDC in advance of making any secure calls, communicate with each other as follows:

1) Alice sends her certificate to Bob.
2) Bob sends his certificate to Alice.
3) Alice and Bob each check the KDC's signature on the certificates they have received.
4) Alice and Bob can now communicate using the keys contained in the certificates.

When making a call, there is no need to call the KDC and little to be gained by cacheing the certificates. The added security arises from the fact that the KDC is not privy to any information that would enable it to spy on the subscribers. The keys that the KDC dispenses are public keys and messages encrypted with these can only be decrypted by using the corresponding secret keys, to which the KDC has no access.

The most carefully articulated attack came from Roger Needham and Michael Schroeder [76], who compared conventional key distribution protocols with similar public-key ones. They counted the numbers of messages required and concluded that conventional cryptography was more efficient than public-key cryptography. Unfortunately, in this analysis, they had ignored the fact that security was better under the public-key protocol they presented than the conventional one.

In order to compromise a network that employs conventional cryptography, it suffices to corrupt the KDC. This gives the intruders access to information sufficient for recovering the session keys used to encrypt past, present, and perhaps future messages. These keys, together with information obtained from passive wiretaps, allow the penetrators of the KDC access to the contents of any message sent on the system.

A public-key network presents the intruder with a much more difficult problem. Even if the KDC has been corrupted and its secret key is known to opponents, this information is insufficient to read the traffic recorded by a passive wiretap. The KDC's secret key is useful only for signing certificates containing subscribers' public keys: it does not enable the intruders to decrypt any subscriber traffic. To be able to gain access to this traffic, the intruders must use their ability to forge certificates as a way of tricking subscribers into encrypting messages with phony public keys.

In order to spy on a call from Alice to Bob, opponents who have discovered the secret key of the KDC must intercept the message in which Alice sends Bob the certificate for her public key and substitute one for a public key they have manufactured themselves and whose corresponding secret key is therefore known to them. This will enable them to decrypt any message that Alice sends to Bob. If such a misencrypted message actually reaches Bob, however, he will be unable to decrypt it and may alert Alice to the error. The opponents must therefore intercept Alice's messages, decrypt them, and reencrypt them in Bob's public key in order to maintain the deception. If the opponents want to understand Bob's replies to Alice, they must go through the same procedure with Bob, supplying him with a phony public key for Alice and translating all the messages he sends her.

The procedure above is cumbersome at best. Active wiretaps are in principle detectable, and the number the intruders must place in the net in order to maintain their

Nintendo - Ancora Exh. 1003

control, grows rapidly with the number of subscribers being spied on. Over large portions of many networks—radio broadcast networks, for example—the message deletions essential to this scheme are extremely difficult. This forces the opponents to place their taps very close to the targets and recreates the circumstances of conventional wiretapping, thereby denying the opponents precisely those advantages of communications intelligence that make it so attractive.

It is worth observing that the use of a hybrid scheme diminishes the gain in security a little because the intruder does not need to control the channel after the session key has been selected. This threat, however, can be countered, without losing the advantages of a session key, by periodically (and unpredictably) using the public keys to exchange new session keys [40].

Public-key techniques also make it possible to conquer another troubling problem of conventional cryptographic security, the fact that compromised keys can be used to read traffic taken at an earlier date. At the trial of Jerry Whitworth, a spy who passed U.S. Navy keying information to the Russians, the judge asked the prosecution's expert witness [27]: "Why is it necessary to destroy yesterday's . . . [key] . . . list if it's never going to be used again? " The witness responded in shock: "A used key, Your Honor, is the most critical key there is. If anyone can gain access to that, they can read your communications."

The solution to this problem is to be found in a judicious combination of exponential key exchange and digital signatures, inherent in the operation of a secure telephone currently under development at Bell-Northern Research [41], [81] and intended for use on the Integrated Services Digital Network.

Each ISDN secure phone has an operating secret-key/public-key pair that has been negotiated with the network's key management facility. The public-key portion is embodied in a certificate signed by the key management facility along with such identifying information as its phone number and location. In the call setup process that follows, the phone uses this certificate to convey its public key to other phones.

1) The telephones perform an exponential key exchange to generate session keys unique to the current phone call. These keys are then used to encrypt all subsequent transmissions in a conventional cryptosystem.
2) Having established an encrypted (though not yet authenticated) channel, the phones begin exchanging credentials. Each sends the other its public-key certificate.
3) Each phone checks the signature on the certificate it has received and extracts from it the other phone's public key.
4) The phones now challenge each other to sign test messages and check the signatures on the responses using the public keys from the certificates.

Once the call setup is complete, each phone displays for its user the identity of the phone with which it is in communication.

The use of the exponential key exchange creates unique session keys that exist only inside the phones and only for the duration of the call. This provides a security guarantee whose absence in conventional cryptography is at the heart of many spy cases: once a call between uncompromised ISDN secure phones is completed and the session keys are destroyed, no compromise of the long term keys that still reside in the phones will enable anyone to decrypt the recording of the call. Using conventional key management techniques, session keys are always derivable from a combination of long-term keying material and intercepted traffic. If long-term conventional keys are ever compromised, all communications, even those of earlier date, encrypted in derived keys, are compromised as well.

In the late 1970s, a code clerk named Christopher Boyce, who worked for a CIA-sponsored division of TRW, copied keying material that was supposed to have been destroyed and sold it to the Russians [66]. More recently, Jerry Whitworth did much the same thing in the communication center of the Alameda Naval Air Station [8]. The use of exponential key exchange would have rendered such previously used keys virtually worthless.

Another valuable ingredient of modern public-key technology is the message digest. Implementing a digital signature by encrypting the entire document to be signed with a secret key has two disadvantages. Because public key systems are slow, both the signature process (encrypting the message with a secret key), and the verification process (decrypting the message with a public key) are slow. There is also another difficulty. If the signature process encrypts the entire message, the recipient must retain the ciphertext for however long the signed message is needed. In order to make any use of it during this period, he must either save a plaintext copy as well or repeatedly decrypt the ciphertext.

The solution to this problem seems first to have been proposed by Donald Davies and Wyn Price of the National Physical Laboratory in Teddington, England. They proposed constructing a cryptographically compressed form or digest of the message [33] and signing by encrypting this with the secret key. In addition to its economies, this has the advantage of allowing the signature to be passed around independently of the message. This is often valuable in protocols in which a portion of the message that is required in the authentication process is not actually transmitted because it is already known to both parties.

Most criticism of public-key cryptography came about because public-key management has not always been seen from the clear, certificate oriented, view described above. When we first wrote about public key, we spoke either of users looking in a public directory to find each other's keys or simply of exchanging them in the course of communication. The essential fact that each user had to authenticate any public key he received was glossed over. Those with an investment in traditional cryptography were not slow to point out this oversight. Public-key cryptography was stigmatized as being weak on authentication and, although the problems the critics saw have long been solved, the criticism is heard to this day.

VIII. APPLICATION AND IMPLEMENTATION

While arguments about the true worth of public-key cryptography raged in the late 1970s, it came to the attention of one person who had no doubt: Gustavus J. Simmons, head of the mathematics department of Sandia National Laboratories. Simmons was responsible for the mathematical aspects of nuclear command and control and digital signatures were just what he needed. The applications were limitless: A nuclear weapon could demand a digitally signed order before it would arm itself; a badge admitting someone to a sensitive area could bear a digitally signed description of the person; a sensor monitoring compliance with a nuclear test ban treaty could place a digital signature on the information it reported. Sandia began immediately both to develop the technology of public-key devices [108], [107], [89] and to study the strength of the proposed systems [105], [16], [34].

The application about which Simmons spoke most frequently, test-ban monitoring by remote seismic observatories [106], is the subject of another paper in this special

Nintendo - Ancora Exh. 1003

section (G. J. Simmons, "How to Insure that Data Acquired to Verify Treaty Compliance are Trustworthy"). If the United States and the Soviet Union could put seismometers on each other's territories and use these seismometers to monitor each other's nuclear tests, the rather generous hundred and fifty kiloton upper limit imposed on underground nuclear testing by the Limited Nuclear Test Ban Treaty of 1963 could be tightened considerably-perhaps to ten kilotons or even one kiloton. The problem is this: A monitoring nation must assure itself that the host nation is not concealing tests by tampering with the data from the monitor's observatories. Conventional cryptographic authentication techniques can solve this problem, but in the process create another. A host nation wants to assure itself that the monitoring nation can monitor only total yield and does not employ an instrument package capable of detecting staging or other aspects of the weapon not covered by the treaty. If the data from the remote seismic observatory are encrypted, the host country cannot tell what they contain.

Digital signatures provided a perfect solution. A digitally signed message from a remote seismic observatory cannot be altered by the host, but can be read. The host country can assure itself that the observatory is not exceeding its authority by comparing the data transmitted with data from a nearby observatory conforming to its own interpretation of the treaty language.

The RSA system was the one best suited to signature applications, so Sandia began building hardware to carry out the RSA calculations. In 1979 it announced a board implementation intended for the seismic monitoring application [**106**]. This was later followed by work on both low- and high-speed chips [**89**], [**94**].

Sandia was not the only hardware builder. Ron Rivest and colleagues at MIT, ostensibly theoretical computer scientists, learned to design hardware and produced a board at approximately the same time as Sandia. The MIT board would carry out an RSA encryption with a one hundred digit modulus in about a twentieth of a second. It was adequate "proof of concept" but too expensive for the commercial applications Rivest had in mind.

No sooner was the board done that Rivest started studying the recently popularized methods for designing large-scale integrated circuits. The result was an experimental nMOS chip that operated on approximately 500 bit numbers and should have been capable of about three encryptions per second [**93**]. This chip was originally intended as a prototype for commercial applications. As it happened, the chip was never gotten to work correctly, and the appearance of a commercially available RSA chip was to await the brilliant work of Cylink corporation in the mid-1980s [**31**].

As the present decade dawned, public-key technology began the transition from esoteric research to product development. Part of AT&T's response to a Carter Administration initiative to improve the overall security of American telecommunications, was to develop a specialized cryptographic device for protecting the Common Channel Interoffice Signaling (CCIS) on telephone trunks. The devices were link encryptors that used exponential key exchange to distribute DES keys [**75**], [**16**].

Although AT&T's system was widely used within its own huge network, it was never made available as a commercial product. At about the same time, however, Racal-Milgo began producing the Datacryptor II, a link encryption device that offered an RSA key exchange mode [**87**]. One device used exponential key exchange, the other RSA, but overall function was quite similar. When the public-key option of the Datacryptor is initialized, it manufactures a new RSA key pair and communicates the public portion to the Datacryptor at the other end of the line. The device that receives this public key manufactures a DES key and sends it to the first Datacryptor encrypted with RSA. Unfortunately, the opportunity for sophisticated digital signature based authentication that RSA makes possible was missed.

Future Secure Voice System

As the early 1980s became the mid-1980s, public-key cryptography finally achieved official, if nominally secret, acceptance. In 1983, NSA began feasibility studies for a new secure phone system. There was fewer than ten-thousand of their then latest system the Secure Telephone Unit II or STU-II and already the key distribution center for the principal network was overloaded, with users often complaining of busy signals. At $12,000 or more apiece, ten-thousand STU-IIs may have been all the government could afford, but it was hardly all the secure phones that were needed. In its desire to protect far more than just explicitly classified communications, NSA was dreaming of a million phones, each able to talk to any of the others. They could not have them all calling the key distribution center every day.

The system to be replaced employed electronic key distribution that allowed the STU-II to bootstrap itself into direct end-to-end encryption with a different key on every call. When a STU-II made a secure call to a terminal with which it did not share a key, it acquired one by calling a key distribution center using a protocol similar to one described earlier.

Although the STU-II seemed wonderful when first fielded in the late seventies, it had some major shortcomings. Some cacheing of keys was permitted, but calls to the KDC entailed significant overhead. Worse, each network had to be at a single clearance level, because there was no way for a STU-II to inform the user of the clearance level of the phone with which it was talking. These factors, as much as the high price and large size, conspired against the feasibility of building a really large STU-II network.

The STU-III is the size of a large conventional telephone and, at about $3000 apiece, substantially cheaper than its predecessor. It is equipped with a two-line display that, like the display of the ISDN secure phone, provides information to each party about the location, affiliation, and clearance of the other. This allows one phone to be used for the protection of information at various security levels. The phones are also sufficiently tamper resistant that unlike earlier equipment, the unkeyed instrument is unclassified. These elements will permit the new systems to be made much more widely available with projections of the number in use by the early 1990s running from half a million to three million [**18**], [**43**].

To make a secure call with a STU-III, the caller first places an ordinary call to another STU-III, then inserts a key-shaped device containing a cryptographic variable and pushes a "go secure" button. After an approximately fifteen second wait for cryptographic setup, each phone shows information about the identity and clearance of the other party on its display and the call can proceed.

In an unprecedented more, Walter Deeley, NSA's deputy director for communications security, announced the STU-III or Future Secure Voice System in an exclusive interview given to the New York Times [**18**]. The objective of the new system was primarily to provide secure voice and low-speed data communications for the U.S. Defense Department and its contractors. The interview did not say much about how it was going to work, but gradually the word began to leak out. The new system was using public key.

The new approach to key management was reported early on [**88**] and one article [**6**] spoke of phones being "repro-

grammed once a year by secure telephone link," a turn of phrase strongly suggestive of a certificate passing protocol, similar to that described earlier, that minimizes the need for phones to talk to the key management center. Recent reports have been more forthcoming, speaking of a key management system called FIREFLY that, [95] "evolved from public key technology and is used to establish pair-wise traffic encryption keys." Both this description and testimony submitted to Congress by Lee Neuwirth of Cylink [78] suggest a combination of key exchange and certificates similar to that used in the ISDN secure phone and it is plausible that FIREFLY too is based on exponentiation.

Three companies: AT&T, Motorola, and RCA are manufacturing the instruments in interoperable versions, and GTE is building the key management system. So far, contracts have been issued for an initial 75,000 began in phones and deliveries began in November 1987.

Current Commercial Products

Several companies dedicated to developing public-key technology have been formed in the 1980s. All have been established by academic cryptographers endeavoring to exploit their discoveries commercially.

The first was RSA Data Security, founded by Rivest, Shamir, and Adleman, the inventors of the RSA cryptosystem, to exploit their patent on RSA and develop products based on the new technology. RSA produces a stand-alone software package called Mailsafe for encrypting and signing electronic mail. It also makes the primitives of this system available as a set of embeddable routines called Bsafe that has been licensed to major software manufacturers [9].

Cylink Corporation of Sunnyvale, Calif., has chalked up the most impressive engineering record in the public-key field. Its first product was the CIDEC HS [32], [63], a high-sped (1.544-Mbit) data encryptor for protecting DS1 telephone trunks. Like AT&T's CCIS encryptor, it uses exponential key exchange to establish DES session keys [77].

Cylink is also first to produce a commercially available RSA chip [7], [31]. The CY1024 is, despite its name, a 1028 bit exponential engine that can be cascaded to perform the calculations for RSA encryptions on moduli more than sixteen thousand bits long. A single CY1024 does a thousand bit encryption in under half a second-both modulus size and speed currently being sufficient for most applications.

The cryptography group at Waterloo University in Ontario have brought the fruits of their labors to market through a company called Cryptech. Their initial inroads into the problem of extracting logarithms over finite fields with 2^n elements [10] made it necessary to employ larger fields. This in turn inspired them to develop high-speed exponentiation algorithms. The result is a system providing both exponential key exchange and half megabit data encryption with the same system [56].

IX. MULTIPLYING, FACTORING, AND FINDING PRIMES

The successes of the RSA system and of exponential key exchange over prime fields have led to significant development in three areas: multiplying, factoring, and finding prime numbers.

Factoring the modulus has remained the front runner among attacks on the RSA system. As factoring has improved, the modulus size required for security has more than doubled, requiring the system's users to hunt for larger and larger prime numbers in order to operate the system securely. As the numbers grow larger, faster and faster methods for doing modular arithmetic are required. The result has been not only the development of a technical base for public-key cryptography, but an inspiration and source of support for number theory [61], [65].

Factoring

In addressing the question of how large the primes in the RSA system should be, Rivest, Shamir, and Adleman's original memo spoke of a number d such that: "determining the prime factorization of a number n which is the product of just two prime numbers of length d (in digits) is 'computationally impossible'." When MIT/LCS/TM-82 first appeared, it contained the statement "Choosing d=40 seems to be satisfactory at present." In a second printing the recommended value of d was changed to 50 and in a third took a sharp leap to 100. This escalation is symbolic of the direction of factoring in the late 1970s and early 1980s.

In 1975, the factoring of a 39 digit number [73]constituted a landmark. The advent of the RSA system, however, was to usher in a decade of rapid progress in this field. By the end of that decade, numbers twice as long could be factored, if not with ease, at least with hours of Cray-1 time [34]. These factorizations confirmed, by actual computer implementation, the number theorists' predictions about factoring speed.

Several factoring techniques of comparable performance have become available in recent years [85]. All factor, in time, proportional to

$$L(n) = e^{\sqrt{\ln n \ln\ln n}}$$

a figure that has already been seen in connection with discrete logarithms. The one that has been most widely applied is called quadratic sieve factoring [34] and lends itself well to machine implementation. One of factoring's gurus, Marvin Wunderlich, gave a paper in 1983 [116] that examined the way in which quadratic sieve factoring could exploit parallel processing to factor a hundred digit number in two months. In the same lecture, Wunderlich also explained the importance of uniformity in factoring methods applied in cryptanalysis. To be used in attacking RSA, a factoring method must be uniform, at least over the class of bicomposite numbers. If it is only applicable to numbers of some particular form, as many methods used by number theorists have been, the cryptographers will simply alter their key production to avoid numbers of that form.

More recently, Carl Pomerance [85] has undertaken the design of a modular machine employing custom chips and specialized to factoring. The size of the numbers you can factor is dependent on how much of such a machine you can afford. He has begun building a $25,000 implementation that he expects to factor 100 digit numbers in two weeks [96]. Ten million dollars worth of similar hardware would be able to factor hundred and fifty digit numbers in a year, but Pomerance's analysis does not stop there. Fixing one year as a nominal upper limit on our patience with factoring any one number, he is prepared to give a dollar estimate for factoring a number of any size. For a two hundred digit number, often considered unapproachable and a benchmark in judging RSA systems, the figure is one hundred billion dollars. This is a high price to be sure, but not beyond human grasp.

Prime Finding

Prime finding has followed a somewhat different course from factoring. This is in part because there are probabilistic techniques that identify primes with sufficient certainty to satisfy all but perhaps the pickiest of RSA users and in part because primality is not in itself a sufficient condition for numbers to be acceptable as RSA factors.

Nintendo - Ancora Exh. 1003

Fermat's Little Theorem guarantees that if n is prime then for all 0<b<n

$$b^{n-1} \equiv 1 \pmod{n}$$

and any number that exhibits this property for some b is said to pass the pseudoprime test to base b. Composite numbers that pass pseudoprime tests to all bases exist, but they are rare and a number that passes several pseudoprime tests is probably a prime.

The test can be refined by making use of the fact that if n is an odd prime only the numbers 1 and −1 are square roots of 1, whereas if n is the product of distinct odd primes, the number of square roots of unity grows exponentially in the number of factors. If the number n passes the pseudoprime test to base b, it can be further examined to see if

$$b^{\frac{n-1}{2}} \equiv 1 \pmod{n}$$

Tests of this kind are called strong pseudoprime tests to base b and very few composite numbers that pass strong pseudoprime tests to more than a few bases are known.

Although there has been extensive work in the past decade on giving genuine proofs of primality [84], [2], [51], the strong pseudoprime tests take care of the primality aspect of choosing the factors of RSA moduli. Another aspect arises from the fact that not all prime numbers are felt to be equally good. In many RSA implementations, the factors of the modulus are not random large primes p, but large primes chosen for particular properties of the factors of p-1 [91], [52].

High-Speed Arithmetic

Because of the progress in factoring during the decade of public-key's existence, the size of the numbers used in RSA has grown steadily. In the early years, talk of hundred digit moduli was common. One hundred digit numbers, 332 bits, did not seem likely to be factored in the immediate future and, with the available computing techniques, systems with bigger moduli ran very slowly. Today, hundred digit numbers seem only just out of reach and there is little discussion of moduli smaller than 512 bits. Two hundred digits, 664 bits, is frequently mentioned, and Cylink has not only chosen to make its chip a comfortable 1028 bits, but also to allow up to sixteen chips to be used in cascade. If this expansion has been pushed by advances in factoring, it has been made possible by advances in arithmetic.

Most of the computation done both in encryption and decryption and in the ancillary activity of manufacturing keys is exponentiation and each exponentiation, in turn, is made up of multiplications. Because, as discussed in the section of exponential key exchange, numbers can be raised to powers in a small number of operations by repeated squaring, it is the speed of the underlying multiplication operation that is crucial.

According to Rivest [94] multiplication on a fixed word length processor takes time proportional to the square length of the operands or $O(k^2)$. If dedicated serial/parallel hardware is constructed for the purpose, this time can be reduced to O(k). In this case, the number of gates required is also proportional to the lengths of the operands, O(k). The fastest implementations [15] run in time O(log k), but here the hardware requirements grow sharply to $O(k^2)$ gates.

X. DIRECTIONS IN PUBLIC-KEY RESEARCH

Public-key cryptography has followed a curious course. In its first three years, three systems were invented. One was broken; one has generally been considered impractical; and the third reigns alone as the irreplaceable basis for a new technology. Progress in producing new public-key cryptosystems is stymied as is the complementary problem of proving the one system we have secure, or even of proving it equivalent to factoring in a useful way.

Stymied though it may be in its central problems, however, the theoretical side of public-key cryptography is flourishing. This is perhaps because the public-key problem changed the flavor of cryptography. It may be difficult to produce good conventional cryptosystems, but the difficulty is all below the surface. It is typically easier to construct a transformation that appears to satisfy the requirements of security than it is to show that a proposed system is no good. The result is a long development cycle ill-suited to the give and take of academic research. Systems that even appear to exhibit the public-key property however, are difficult to find and this sort of difficulty is something the theoretical computer scientists can get their teeth into. The early taste of success that came with the development of RSA has inspired the search for solutions to other seemingly paradoxical problems and led to active exploration of a variety of new cryptographic disciplines.

This is not to say that contemporary research is not motivated by application. A constant caution in conventional cryptography is that the strength of a cryptosystem in one mode of operation does not guarantee its strength in another. It is widely felt, for example, that a conventional block cryptosystem such as DES is a suitable component with which to implement other modes of operation, but no proofs have been offered. This burdens anyone who chooses the system as a building block with a separate certificational examination of every configuration in which it is to be used. One objective of research in public-key cryptography has been to demonstrate the equivalence of many such secondary cryptographic problems to those that define the strength of the system. Substantial progress has been made in proving that the strength of cryptographic protocols is equivalent to the strength of the RSA system and that the protection provided by RSA is uniform [4].

There is another sort of applied flavor to even the purest of cryptographic research—a search for ways of transplanting our current social and business mechanisms to a world in which communication is primarily telecommunication. The digital signature was the first great success in this direction, which can be characterized as asking: What can we do with paper, pencil, coins, and handshakes that would be hard to do without them. And, how can we do it without them?

In 1977, I gave a talk on the problem of developing a purely electronic analog of the registered mail receipt, in the current topics session of the International Symposium on Information Theory at Cornell. My message was pessimistic, arguing for both the importance and the intractability of the problem, but fortunately my pessimism was premature. A year and a half later, the MIT group penned a note entitled "Mental Poker" [99]. It did not solve the problem of receipts for registered mail, but did show how to do something just as surprising: gamble over the telephone in a way that prevented either party from cheating without being discovered. This as it turned out was just the beginning.

To my delight, the problem of registered mail was rediscovered in Berkeley in 1982 as part of a larger category of problems that could be solved by ping-pong protocols and the emergence of this subject was one of the highlights of Crypto '82 [20]. Despite problems with protocols that were either broken or impossibly expensive [55], progress has been sufficient to provide hope that registered mail, contract signing, and related problems will one day have practical solutions.

Nintendo - Ancora Exh. 1003

In separate 1979 papers, G. R. Blakley at the University of Texas and Adi Shamir at MIT [11], [100] opened yet another direction of investigation: how secret information can be divided among several people in such a way that any k of them, but no fewer, can recover it. Although this field of secret sharing, unlike that of ping-pong protocols emerged full grown with probably correct and easily implementable protocols, it has been the subject of continuing examination [5], [26], [45], [5].

David Chaum, currently at the Center for Mathematics and Computer Science in Amsterdam, has applied public-key technology to a particularly challenging set of problems [21], [22]. In a society dominated by telecommunication and computers, organizations ranging from credit bureaus to government agencies can build up dossiers on private citizens by comparing notes on the credentials issued to the citizens. This dossier building occurs without the citizens' knowledge or consent and, at present, the only protection against abuses of this power lies in legal regulation. Chaum has developed technical ways of permitting an individual to control the transfer of information about him from one organization to another. Without action on the part of an individual to whom credentials have been issued, no organization is able to link the information it holds about the individual with information in the databanks of any other organization. Nonetheless, the systems guarantee that no individual can forge organizational credentials. Chaum's techniques address problems as diverse as preventing spies from tracing messages through electronic mail networks [19], [24] and protecting the privacy of participants in transactions with systems that recapture in electronic media both the assurance and the anonymity of cash [21].

The work drawing most attention at present is probably the field best known under the name of zero-knowledge proofs [49], [50], though similar theories, based on different assumptions about the capabilities of the participants, have been developed independently [23], [13], [14]. One of the idea's originators, Silvio Micali at MIT, described it as "the inverse of a digital signature." A zero-knowledge proof permits Alice to demonstrate to Bob that she knows something, but gives him no way of conveying this assurance to anybody else. In the original example, Alice convinced Bob that she knew how to color a map with three colors, but gave him no information whatever about what the coloring was.

The view that a zero-knowledge proof is the inverse of a digital signature now seems ironic, because a form of challenge and response authentication, applicable to the signature problem, has become the best known outgrowth of the field. In this system, the responder demonstrates to the challenger his knowledge of a secret number, without revealing any information about what the number is. Amos Fiat and Adi Shamir have recently brought forth an identification system of this sort, and announced a proof that breaking it is equivalent to factoring [47].

A purist might respond to all this by saying that having failed to solve the real problems in public-key cryptography, cryptographers have turned aside to find other things about which to write papers. It is a situation that has been seen before in mathematics. At the end of the last century, mathematical analysis ground to a halt against intractable problems in Fourier Theory, differential equations, and complex analysis. What many mathematicians did with their time while not solving the great problems was viewed with scorn by critics who spoke of the development of point set topology and abstract algebra as "soft mathematics." Only at mid-century did it become clear what had happened. In the abstractions a great hammer had been forged and through the 1950s and 1960s the classic problems began to fall under its blows. Perhaps cryptography will be equally lucky.

XI. WHERE IS PUBLIC KEY GOING?

In just over ten years, public-key cryptography has gone from a novel concept to a mainstay of cryptographic technology. It is soon to be implemented in hundreds of thousands of secure telephones and efforts are under way to apply the same mechanisms to data communications on a similar scale [97]. The outlook in the commercial world is equally bright. As early as the fourth quarter of this year, digital signatures may enter retail electronic funds transfer technology in a British experiment with point of sale terminals [57]. The demand for public key is exemplified by a recent conference on smart cards in Vienna, Austria [111], where one question was heard over and over again: When will we have an RSA card?

Now that it has achieved acceptance, public-key cryptography seems indispensable. In some ways, however, its technological base is disturbingly narrow. With the exception of the McEliece scheme and a cumbersome knapsack system devised explicitly to resist the known attacks [25], virtually all surviving public-key cryptosystems and most of the more numerous signature systems employ exponentiation over products of primes. They are thus vulnerable to breakthroughs in factoring or discrete logarithms. Key exchange systems are slightly better off since they can use the arithmetic of primes, prime products, or Galois fields with 2^n elements and are thus sensitive to progress on the discrete logarithm problem only.

From the standpoint of conventional cryptography, with its diversity of systems, the narrowness bespeaks a worrisome fragility. This worry, however, is mitigated by two factors.

The operations on which public-key cryptography currently depends—multiplying, exponentiating, and factoring—are all fundamental arithmetic phenomena. They have been the subject of intense mathematical scrutiny for centuries and the increased attention that has resulted from their use in public-key cryptosystems has on balance enhanced rather than diminished our confidence.

Our ability to carry out large arithmetic computations has grown steadily and now permits us to implement our systems with numbers sufficient in size to be vulnerable only to a dramatic breakthrough in factoring, logarithms, or root extraction.

It is even possible that RSA and exponential key exchange will be with us indefinitely. The fundamental nature of exponentiation makes both good candidates for eventual proof of security and if complexity theory evolves to provide convincing evidence of the strength of either, it will establish a new paradigm for judging cryptographic mechanisms. Even if new systems were faster and had smaller keys, the current systems might never be superseded altogether.

Such proofs have yet to be found, however, and proposed schemes are continually presented at the cryptographic conferences [12], [114], [80], [30], [82]. Approaches include generalizing RSA to other rings and various attempts to replace exponentials with polynomials, but in general they have not fared well and some of their fates are discussed elsewhere in this special section (E. F. Brickell and A. M. Odlyzko "Cryptanalysis: A Survey of Recent Results"). So far, the goal of improving on the performance of RSA without decreasing its security has yet to be achieved.

An appealing idea that has been put forward by Stephen Wolfram and studied by Papua Guam [54] is the use of

Nintendo - Ancora Exh. 1003

cellular automata. Guam's system is too new to have received careful scrutiny and superficial examination suggests that it may suffer a weakness similar to one seen in other cases [46]. Even should this effort fail, however, the cellular automaton approach is attractive. Cellular automata differ from such widely accepted cryptographic mechanisms as shift registers in that, even if they are invertible, it is not possible to calculate the predecessor of an arbitrary state by simply reversing the rule for finding the successor. This makes them a viable vehicle for trap doors. Cellular automata also lend themselves to study of the randomness properties required of strong cryptographic systems [115].

What will be the outcome of such research? In an attempt to foresee the future of cryptography in 1979, I wrote [39]:

> "Prospects for development of new and more efficient public key cryptographic systems by the latter part of the eighties are quite good. Public key cryptography is more successful today than algebraic coding theory was at the age of four. The major breakthroughs in that field did not begin till the latter part of its first decade, but then progressed rapidly. The similarity of the two fields is reason for optimism that . . . public key cryptography will follow a similar course.
>
> Increasing use of the available public key systems in the 1980s will spread awareness of both their advantages and the performance shortcomings of the early examples. The research response to this awareness will probably produce better public key systems in time for use during the first half of the nineties."

My schedule was clearly too optimistic. If there are public-key cryptosystems with better performance or greater security waiting in the wings, they are proprietary systems that have yet to make even their existence known. Other aspects of the argument are closer to the mark, however. The use of public-key cryptosystems has increased dramatically and with it awareness of their advantages. Judicious use of hybrid systems and improved arithmetic algorithms have reduced the "performance shortcomings" to the status of a nuisance in most applications and the biggest motivation for seeking new systems today is probably the desire not to have all our eggs in one basket. Unless the available systems suffer a cryptanalytic disaster, moreover, the very success of public-key cryptography will delay the introduction of new ones until the equipment now going into the field becomes outmoded for other reasons.

For a discipline just entering its teens, the position of public-key cryptography should be seen not as a fragile, but as a strong one.

REFERENCES

[1] L. M. Adleman and R. L. Rivest, "How to break the Lu-Lee (COMSAT) public key cryptosystem," MIT Laboratory for Computer Science, Jul. 24, 1979.

[2] L. M. Adleman, C. Pomerance, and R. S. Rumley, "On distinguishing prime numbers from composite numbers," *Ann. Math.*, vol. 117, no. 2, pp. 173–206, 1983.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, Mass.: Addison-Wesley, 1974.

[4] W. Alexi, B. Chor, O. Goldreich, and C. P. Schnor, "RSA/Rabin bits are ½+1/(poly(log N)) secure," in 25th *Annual IEEE Symp. on Foundations of Comp. Sci.*, pp. 449–457, 1984.

[5] C. Asmuth and J. Blum, "A modular approach to key safeguarding," *IEEE Trans. Informat. Theory*, vol. IT-29, pp. 208–210, March 1983.

[6] "Contractors ready low-cost, secure telephone for 1987 service start," *Aviat. Week Space Technol., pp.* 114–115, January 1986.

[7] C. Barney, "Cypher chip makes key distribution a snap," *Electronics*, Aug. 7, 1986.

[8] J. Barron, *Breaking the Ring*. Boston, Mass.: Houghton Mifflin, 1987.

[9] D. ben-Aaron, "Mailsafe signs, seals, and delivers files," *Information Week*, pp. 19–22, Sep. 15, 1986.

[10] I. F. Blake, R. Fuji-Hara, R. C. Mullin, and S. A. Vanstone, "Computing logarithms in finite fields of characteristic two," *SIAM J. Alg. Disc. Methods*, vol. 5, no. 2, pp. 276–285, June 1984.

[11] G. R. Blakley, "Safeguarding cryptographic keys," in *National Computer Conf.*, pp. 313–317, 1979.

[12] G. R. Blakley and D. Chaum, Eds., *Advances in Cryptology: Proceedings of Crypto* '84, Berlin, Germany: Springer-Verlag, 1985.

[13] G. Brassard and C. Crépeau, "Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for SAT and beyond," in 27th *Annual IEEE Symp. on the Foundations of Comp. Sci.*, pp. 188–195, 1986.

[14] G. Brassard, C. Crépeau, and D. Chaum, "Minimum disclosure proofs of disclosure proofs of knowledge," Center for Mathematics and Computer Science, Amsterdam, Rep. PM-R8710, December 1987. (To appear as an invited paper in *J. Comput. Syst. Sci.*)

[15] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Crypto* '82 [20], pp. 51–60.

[16] E. F. Brickell and G. J. Simmons, "A status report on knapsack based public key cryptosystems," *Congressus Numerantium*, vol. 7, pp. 3–72, 1983. The CCIS encryptor is mentioned on pp. 4–5.

[17] E. F. Brickell, "Breaking iterated knapsacks," in *Crypto* '84[12], pp. 342–358.

[18] D. Burnham, "NSA seeking 500,000 'secure' telephones," *The New York Times*, Oct. 7, 1984.

[19] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *CACM*, vol. 24, no. 2, pp. 84–88, February 1981.

[20] D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., *Advances in Cryptology, Proceedings of Crypto* '82. New York, N.Y.: Plenum, 1983.

[21] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *CACM*, vol. 28, no. 10, pp. 1030–1044, October 1985.

[22] D. Chaum and J. H. Evertse, "A secure and privacy-protecting protocol for transmitting personal information between organizations," in *Crypto* '86[80], pp. 118–167.

[23] D. Chaum, "Demonstrating that a public predicate can be satisfied without revealing any information about how," in *Crypto* '86[80], pp. 195–199.

[24] -----, "The dining cryptographers problem: Unconditional sender untraceability," *J. Cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[25] B. Chor and R. L. Rivest, "A knapsack type public-key cryptosystem based on arithmetic in finite fields," in *Crypto* '84[12], pp. 54–65.

[26] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in 26th *Annual IEEE Symp. on the Foundations of Comp. Sci.*, pp. 383–395, 1985.

[27] Testimony of David Earl Clark at the trial of Jerry Alfred Whitworth before Judge J. P. Vukasin, Jr., in the U.S. District Court, Northern District of California, Mar. 25, 1986. Reported by V. Pella Balboni, pp. 111–1345.

[28] D. Coppersmith, "Fast evaluation of logarithms in fields of characteristic two," *IEEE Trans. Informat. Theory*, vol. IT-30, pp. 587–594, 1984.

[29] D. Coppersmith, A. M. Odlyzko, and R. Schroeppel, "Discrete logarithms in GF(p)," *Algorithmica*, vol 1, , pp. 1–16, 1986.

[30] N. Cot and I. Ingemarsson, Eds., *Advances in Cryptology, Proceedings of EUROCRYPT* '84. Berlin, Germany: Springer-Verlag, 1985.

[31] "Cidec-HS high speed DES encryption for digital networks," product description, Cylink Corporation, Sunnyvale, Calif.

[32] "Key management development package," product description, Cylink Corporation, Sunnyvale, Calif.

[33] D. W. Davies and W. L. Price, "The applications of digital signatures based on public key cryptosystems," National Physical Laboratory Rep. DNACS 39/80, December 1980.

[34] J. A. Davis, D. B. Holdridge, and G. J. Simmons, "Status report on factoring (at the Sandia National Laboratories)," in *Euro-crypt* '84[30], pp. 183–215.

[35] W. Diffie and M. E. Hellman, "Multiuser cryptographic techniques," in *Proc. Nat. Computer Conf.*, (New York, N.Y.), pp. 109–112, Jun. 7–10, 1976.

[36] -----, "New directions in cryptography," *IEEE Trans Informat. Theory*, vol. IT-22, pp. 644–654, November 1976.

[37] -----, "Exhaustive cryptanalysis of the NBS data encryption standard," *Computer*, vol. 10, no. 6, pp. 74–84, June 1977.

[38] W. Diffie, "Conventional versus public key cryptosystems," in [109], pp. 41–72. Rabin's system is discussed on p. 70, the relative strength of conventional and public-key distribution on pp. 64–66.

[39] -----, "Cryptographic technology: Fifteen-year forecast," in [109], pp. 301–327.

[40] -----, "Securing the DoD transmission control protocol," in Crypto '85 [114], pp. 108–127.

[41] W. Diffie, L. Strawczynski, B. O'Higgins, and D. Steer, "An ISDN secure telephone unit," in *Proc. National Communications Forum* 1987, pp. 473–477.

[42] E. Dolnick, "N. M. scientist cracks code, wins $1000," *The Boston Globe*, Nov. 6, 1984.

[43] Electronic Industries Association, "Comsec and Compusec market study," Jan. 14, 1987.

[44] Federal Register, "Encryption algorithm for computer data protection," vol. 40, no. 52, pp. 12134–12139, Mar. 17, 1975.

[45] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in 28th *Annual IEEE Symp. on the Foundations of Comp. Sci.*, pp. 427–437, 1987.

[46] H. Fell and W. Diffie, "Analysis of a public key approach based on polynomial substitution," in *Crypto* '85[114], pp.108–127.

[47] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Crypto* '86[80], pp. 186–212.

[48] M. Gardner, "A new kind of cipher that would take millions of years to break," *Sci. Amer.*, pp. 120–124 (Mathematical Games), August 1977.

[49] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," in 27th *Annual IEEE Conf. on the Foundations of Comp. Sci.*, pp. 174–187, 1986.

[50] S. Goldwasser, S. Micali, and C. Rackoff, "Knowledge complexity of interactive proofs," in 17*th Symp. on the Theory of Computing*, pp. 291–304, 1985.

[51] S. Goldwasser and J. Killian, "All primes can be quickly certified," in 18*th Symp. on the Theory of Computing*, pp. 316–329, 1986.

[52] J. Gordon, "Strong primes are easy to find," in *Euro-crypt* '84[30], pp. 215–223.

[53] -----, speech at the Zurich Seminar, 1984. In this lecture, which has unfortunately never been published, Gordon assembled the facts of Alice and Bob's precarious lives, which had previously been available only as scattered references in the literature.

[54] P. Guam, "Cellular automaton public key cryptosystem," *Complex Systems*, vol. 1, pp. 51–56, 1987.

[55] J. Hastad and A. Shamir, "The cryptographic security of truncated linearly related variables," in 17*th Symp. on Theory of Computing*, pp. 356–362, 385.

[56] D. Helwig, "Coding chip devised in Waterloo," *The Globe and Mail*, Jan. 1, 1987.

[57] "National EFT POS to use public key cryptography," *Information Security Monitor*, vol. 2, no. 12, p. 1, November 1987.

[58] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," in *Globecom* '87, pp. 361–364, 1987.

[59] C. S. Kline and G. J. Popek, "Public key vs. conventional key encryption," *in National Computer Conf.*, 1979.

[60] D. Knuth, "Semi-numerical algorithms," in *The Art of Computer Programming*, vol. 2, 2nd ed. Reading, Mass.: Addison-Wesley, 1981, pp. 316–336.

[61] N. Koblitz, A Course in *Number Theory and Cryptography*. New York, N.Y.: Springer-Verlag, 1987.

[62] L. M. Kohnfelder, "Toward a practical public key cryptosystem," Bachelors Thesis, MIT Dept. of Electrical Engineering, May 1978.

[63] R. Kopeck, "T1 encryption plan protects data," PC Week, Mar. 3, 1987.

[64] J. Kowalchuk, B. P. Schanning, and S. Powers, "Communication privacy: Integration of public and secret key cryptography," in *National Telecommunications Conf.*, (Houston, Tex.), pp. 49.1.1–5, Nov. 30–Dec. 4, 1980.

[65] E. Kranakis, *Primality and Cryptography*. New York, N.Y.: Wiley, 1986.

[66] R. Lindsey, *The Falcon and the Snowman*. New York, N.Y.: Simon and Schuster, 1979.

[67] S. Lu and L. Lee, "A simple and effective public key cryptosystem," *Comsat Technical Rev.*, vol. 9, no. 1, Spring 1979.

[68] K. S. McCurley, "A key distribution system equivalent to factoring," Department of Mathematics, University of Southern California, Jun. 3, 1987.

[69] R. J. McEliece, "A public key cryptosystem based on algebraic coding theory," JPL DSN Progress Rep. 42–44, pp. 114–116, January–February 1978.

[70] R. Merkle, "Secure communication over insecure channels," CACM, pp. 294–299, April 1978.

[71] R.C. Merkle and M. E. Hellman, "Hiding information and signatures in trap door knapsacks, " *IEEE Trans. Informat. Theory*, vol. IT-24, pp. 525–30, September 1978.

[72] R. Merkle, Letters to the Editor, *Time Magazine, vol.* 120, no. 20, p. 8, Nov. 15, 1982.

[73] M. A. Morrison and J. Brillhart, "A method of factoring and the factorization of F_7," *Math. Comp.*, vol. 29, pp. 18–205, 1975.

[74] "Advanced techniques in network security," Motorola Government Electronics Division, Scottsdale, Ariz., about 1977.

[75] F. H. Myers, "A data link encryption system," in *National Telecommunications Conf.*, (Washington, D.C.), pp. 4.5.1–4.5.8, Nov. 27–29, 1979.

[76] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," CACM, vol. 21, pp. 993–999, December 1978.

[77] L. Neuwirth, "A comparison of four key distribution methods," *Telecommunications*, pp. 110–111, 114–115, July 1986.

[78] "Statement of Lee Neuwirth of Cylink on HR145," submitted to Congressional committees considering HR145, February 1987.

[79] A. M. Odlyzko, "Discrete logarithms in finite fields and their cryptographic significance," in *Eurocrypt* '84[30], pp. 225–314.

[80] -----, Ed., *Advances In Cryptology-CRYPTO* '86. Berlin, Germany: Springer-Verlag, 1987.

[81] B. O'Higgins, W. Diffie, L. Strawczynski, and R. de Hoog, "Encryption and ISDN-A natural fit," in *International Switching Symp.*, (Phoenix, Ariz.), pp. A11.4.1–7, Mar. 16–20, 1987.

[82] F. Pichler, Ed., Advances in *Cryptology-Proceedings of EUROCRYPT* '85. Berlin, Germany: Springer-Verlag, 1986.

[83] S. C. Pohlig and M. E. Hellman, "An improved algorithm for computing logarithms in GF(p) and its cryptographic significance," *IEEE Trans. Informat. Theory*, vol. IT-24, pp. 106–110, January 1978.

[84] C. Pomerance, "Recent developments in primality testing," *The Mathematical Intelligence*, vol. 3, no. 3, pp. 97–105, 1981.

[85]C. Pomerance, J. W. Smith, and R. Tuler, "A pipe-line architecture for manufacturing large integers with the quadratic sieve algorithm," to appear in a special issue on cryptography of the *SIAM J. Computing*.

[86] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," MIT Laboratory for Computer Science, MIT/LCS/TR-212, January 1979.

[87] "Datacryptor II, public key management option," Racal-Milgo, Sunrise Fla., 1981.

[88] "AT&T readying new spy-proof phone for big military and civilian markets," *The Report on AT&T*, pp. 6–7, Jun. 2, 1986.

[89] R. F. Rieden, J. B. Snyder, R. J. Widman, and W. J. Barnard, "A two-chip implementation of the RSA public-key encryption algorithm," in GOMAC (Government Microcircuit Applications Conference), (Orlando, Fla.), pp. 24–27, November 1982.

[90] R. L. Rivest, A. Shamir, and L. Adleman, "On digital signatures and public key cryptosystems," MIT Laboratory for Computer Science, MIT/LCS/TR-212, January 1979.

[91] -----, "A method for obtaining digital signatures and public key cryptosystems," CACM, vol. 21, no. 2, pp. 120–126, February 1978.

[92] R. Rivest, personal communication with H. C. Williams cited on p. 729 in [113].

[93] -----, "A description of a single-chip implementation of the RSA cipher," *Lambda*, vol 1, no. 3, pp. 14–18, Fall 1980.

[94] -----, "RSA chips (past/present/future)," in *Eurocrypt* '84[30], pp. 159–165.

[95] H. L. Rogers, "An overview of the caneware program," paper 31, presented at the 3rd Annual Symp. on Physical/Electronic Security, Armed Forces Communications and Electronics Association, Philadelphia Chapter, August 1987.

[96] "Toward a new factoring record," *Science News*, p. 62, Jan. 23, 1987.

[97] "SDNS: A network on implementation," in 10*th National Computer Security Conf.*, (Baltimore, Md.), pp. 150–174, Sept. 21–24,1987. Session containing six papers on the Secure Data Network System.

[98] A. Shamir, "A fast signature scheme," M.I.T. Laboratory for Computer Science, Technical Memorandum, MIT/LCS/TM-107, July 1978.

[99] A. Shamir, R. L. Rivest, and L. M. Adleman, "Mental poker," MIT Laboratory for Computer Science, Technical Memorandum, MIT/LCS/TM-125, Jan. 29, 1979.

[100] A. Shamir, "How to share a secret," CACM, vol. 22, no. 11, pp. 612–613, November 1979.

[101] -----, "A polynomial time algorithm for breaking Merkle-Hellman cryptosystems (extended abstract)," Research announcement, preliminary draft, Applied Mathematics, Weizmann Institute, Rehovot, Israel, Apr. 20, 1982. This paper appeared with a slightly different title: "A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem (extended abstract)," in Crypto '82 [20], pp. 279–288.

[102] -----, "A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem," *IEEE Trans. Informat. Theory*, vol. IT-30, no. 5, pp. 699–704, September 1984.

[103] Z. Shmuely, "Composite Diffie-Hellman public-key generating systems are hard to break," Computer Science Department, Technion, Haifa, Israel, Technical Rep. 356, February 1985.

[104] R. Silver, "The computation of indices modulo P," Mitre Corporation, Working Paper WP-07062, p. 3, May 7, 1964.

[105] G. J. Simmons and M. J. Norris, "Preliminary comments on the M.I.T. public key cryptosystem," *Cryptologia*, vol. 1, pp. 406–414, October 1977.

[106] G. J. Simmons, "Authentication without secrecy: A secure communications problem uniquely solvable by asymmetric encryption techniques," in *IEEE EASCON* '79 (Washington, DC), pp. 661–662, Oct. 9–11, 1979.

[107] G. J. Simmons and M. J. Norris, "How to cipher faster using redundant number systems," Sandia National Laboratories, SAND-80-1886, August 1980.

[108] G. J. Simmons, "High speed arithmetic utilizing redundant number systems," in *National Telecommunications Conf.*, (Houston, Tex.), pp. 49.3.1–2, Nov. 30–Dec. 4, 1980.

[109] -----, Ed., *Secure Communications and Asymmetric Cryptosystems*. AAAS Selected Symposium 69. Boulder, CO: Westview Press, 1982.

[110] _, "Cryptology" in *Encyclopaedia Britannica*, 16th Edition. Chicago, Ill.; Encyclopaedia Britannica, 1986, pp. 913–924B.

[111] Proceedings of Smart Card 2000, Vienna, Austria, Oct. 19–20, 1988.

[112] M. V. Wilkes, *Time-Sharing Computer Systems*. New York, N.Y.: American Elsevier, 1972.

[113] H. C. Williams, "A modification of the RSA public-key cryptosystem," IEEE Trans. Informat. Theory, vol. IT-26, no. 6, pp. 726–729, November 1980.

[114] _, Eds., *Advances in Cryptology-CRYPTO* '85. Berlin, Germany; Springer-Verlag, 1986.

[115] S. Wolfram, "Cryptography with cellular automata," in *Crypto* '85 [114], pp. 429–432.

[116] M. C. Wunderlich, "Recent advances in the design and implementation of large integer factorization algorithms," in 1983 Symp. on Security and Privacy, (Oakland, Calif.), pp. 67–71, Apr. 25–27, 1983.

[119] K. Yiu and K. Peterson, "A single-chip VLSI implementation of the discrete exponential public key distribution system," in GOMAC (Government Microcircuit

Nintendo - Ancora Exh. 1003

Applications Conference), (Orlando, Fla.), pp. 18–23, November 1982.

Whitfield Diffie was born in Washington, DC, on Jun. 5, 1944. He received the B.S. degree in mathematics from the Massachusetts Institute of Technology, Cambridge, Mass., in 1965.

While at Mitre Corporation from 1965 to 1969, he worked with Carl Engelman in developing the Mathlab symbolic mathematical manipulation system, later expanded at MIT to become Macsyma. In 1969, he transferred to the Stanford University Artificial Intelligence Laboratory to work with John McCarthy on proof checking and proof of correctness of programs. While there he also developed the compiler adopted for the U.C. Irvine Ilisp system. In 1973, Diffie took leave from Stanford and began his work on cryptography while traveling around the U.S. He continued this work as a graduate student under Martin Hellman at Stanford University from 1975 through 1978. Since 1978, Diffie has been the Manager of Secure Systems Research for Bell-Northern Research, Mountain View, Calif. His most recent work has been on key management protocols for telephones designed to operate on the developing Integrated Services Digital Network.

Manuscript received Jan. 19, 1988; revised Mar. 25, 1988.
The author is with Bell-Northern Research, Mountain View, Calif. 94039, USA.
IEEE Log Number 8821645.
0018–9219/88/0500–0560$01.00 (c) 1988 IEEE.

APPENDIX B

Designs, Codes and Cryptography, 2, 107–125 (1992)

© 1992 Kluwer Academic Publishers

Manufactured in the Netherlands

AUTHENTICATION AND AUTHENTICATED KEY EXCHANGES

WHITFIELD DIFFIE*
Sun Microsystems, 2550 Garcia Avenue., Mountain View, Calif. 94043
PAUL C. VAN OORSCHOT AND MICHAEL J. WIENER
Bell-Northern Research, P.O. Box 3511 Station C, Ottawa, Ontario K1Y 4H7 Canada
Communicated by S. A. Vanstone
Received Nov. 22, 1991. Revised Mar. 6, 1992.

Abstract. We discuss two-party mutual authentication protocols providing authenticated key exchange, focusing on those using asymmetric techniques. A simple, efficient protocol referred to as the station-to-station (STS) protocol is introduced, examined in detail, and considered in relation to existing protocols. The definition of a secure protocol is considered, and desirable characteristics of secure protocols are discussed.

1. Introduction

The goal of an authentication protocol is to provide the communicating parties with some assurance that they know each other's true identities. In an authenticated key exchange, there is the additional goal that the two parties end up sharing a common key known only to them. This secret key can then be used for some time thereafter to provide privacy, data integrity, or both. In this paper, we discuss the security of public-key based authentication protocols, with and without an associated key exchange. We restrict our attention to two-party mutual authentication, rather than multi-party and one-way authentication protocols. We assume that individual underlying cryptographic mechanisms are not vulnerable, and restrict our attention to attacks on protocols themselves. An enemy (attacker, intruder, adversary) can see all exchanged messages, can delete, alter, inject, and redirect messages, can initiate communications with another party, and can reuse messages from past communications.

Much work has been done in recent years involving identification and authentication schemes using asymmetric techniques. Identity-based schemes, as introduced by Shamir [29], rely on the existence of a trusted central authority, that holds secret information from which other secrets are generated and distributed to individual users when those users join the system. Günther [15] has proposed an identity-based protocol providing authenticated key establishment, making use of the ideas of Diffie-Hellman key exchange [9] and the ElGamal signature scheme [11]. The authentication is indirect and does not offer perfect forward secrecy (see Section 4), although the latter can be provided at the cost of incorporating an extra exchange of Diffie-Hellman exponentials. Okamoto and Tanaka [22] have proposed an identity-based authenticated key establishment protocol based on exponential key exchange and RSA. They offer versions which provide both indirect and direct authentication, although the latter as presented, employs timestamps (Section 4), and some of the fields in the exchange may be unnecessary or redundant. Interactive identification protocols which provide proof of identity and make use of ideas involving zero-knowledge have been proposed by Fiat and Shamir [12], and more efficient protocols have been subsequently proposed by Guillou and Quisquater [14] and Schnorr [28], among others. These identification protocols differ from authenticated key exchanges in that the former do not provide keys for use in subsequent communications (e.g., for data integrity or data confidentiality).

As has been pointed out by many others, [5], [7], [13], [19], [20], the design of cryptographic protocols in general, and authentication protocols in particular, is extremely error prone. The literature is filled with protocols that have been found to contain security flaws ranging from minor to fatal in severity. Furthermore, aside from security issues, it is a concern in practice that many of the published protocols contain redundancies or are inefficient with respect to the number of communications required, the number of cryptographic operations required (implying high computational demands), or the number and types of fields required in the communicated messages. This motivates the search for authentication protocols that are simple, require a minimum number of communications, a small number of fields in each message or token, and a small number of cryptographic operations. These considerations motivate the present work on public-key based protocols. Similar considerations motivated Bird et al. [5] in their work on symmetric authentication protocols, which helped focus our attention on the idea of matching protocol runs (see Section 3). Our work extends the definition of a secure protocol to public-key based protocols with optional key exchange.

We are concerned with both authentication and key exchange. It is now well accepted that these topics should be considered jointly rather than separately [2]. A protocol providing authentication without key exchange is susceptible to an enemy who waits until authentication is complete and then takes over one end of the communications line. Such an attack is not precluded by a key exchange that is

independent of authentication. Key exchange should be linked to authentication so that a party has assurances that an exchanged key (which might be used to facilitate privacy or integrity and thus keep authenticity alive) is in fact shared with the authenticated party, and not an imposter. For these reasons, it is essential to keep key exchange in mind in the design and analysis of authentication protocols.

In the remainder of this paper, we first provide some background regarding attacks on protocols, in an effort to motivate and give context to what follows. We then proceed with a definition of a secure protocol, and discuss characteristics that we consider desirable in an authentication protocol. We introduce a protocol referred to as the station-to-station protocol, examine it in detail, and justify its features. Some related protocols are discussed, and the proposed protocol is considered in relation to these. We conclude with a summary of principles we feel are important in the design of authentication protocols.

2. Notation and Motivation

Before discussing protocols in more detail, we first define some notation. For historical reasons, we give the two parties involved the names Alice and Bob.

{ } Braces indicate a hash function. {x, y} is the result when a hash function is applied to x concatenated with y.

s_A Alice's secret key for a signature scheme. $s_A(x)$ is Alice's signature on x. $s_A\{x\}$ is Alice's signature on the hashed version of x.

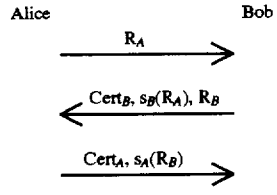
p_A Alice's public key for a signature scheme. If the signature scheme is a public-key cryptosystem, then we define $p_A\{x\}$ and $p_A(x)$ to be Alice's public key encryption function with and without hashing.

$Cert_A$ Alice's certificate, containing Alice's name (and possibly other information), her public key, and a trusted authority T's signature over this information. $Cert_A$=(Alice, p_A, . . . , s_t{Alice, p_A, . . .}). $Cert_A$ binds the name Alice to the public key p_A. If Alice sends her certificate to Bob and provides evidence that she knows the secret key s_A corresponding to p_A, then she has provided evidence to Bob that she is in fact Alice.

$E_k(\cdot)$ Encryption using a symmetric cryptosystem with key K.

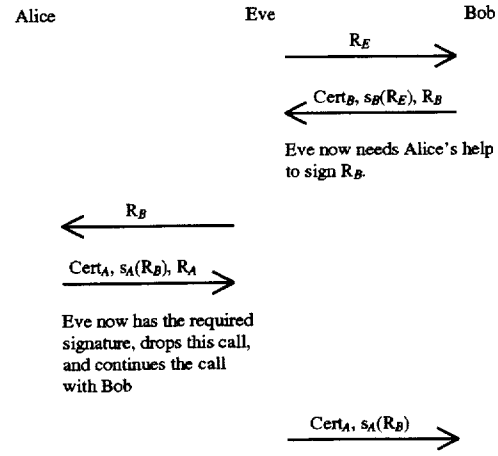
To illustrate an attack on a protocol and motivate what follows, consider the following simple (but flawed) challenge-response protocol where Alice and Bob sign each other's random number challenges.

Insecure simple challenge-response



Alice begins by sending the random challenge R_A to Bob. Bob responds with his certificate, his signature on R_A and a random challenge R_B. Alice uses Bob's public key in $Cert_B$ to verify Bob's signature, and then responds with her certificate and signature on R_B. Finally, Bob verifies Alice's signature.

An enemy Eve can impersonate Alice in a communication with Bob by passing Bob's challenge along to Alice:



Eve begins by initiating the protocol with Bob. When Bob sends the challenge to Eve, Eve initiates another instance of the protocol with Alice and gets Alice to sign Bob's challenge. Eve can then complete the authentication with Bob and successfully impersonate Alice. The main problem here is that the challenged party has no influence over what he will sign. (As a general rule, it is better if both parties have some influence over the quantity signed.) The challenger can abuse this protocol to get a signature on any quantity he chooses.

We now turn our attention to secure protocols.

3. Definition of a Secure Protocol

A particular instantiation of an authentication protocol is referred to as a run. Before presenting a definition of a secure protocol, we first consider the properties of what we consider to be a successful run. In a successful run, two communicating parties, Alice and Bob, exchange a number of messages at the end of which they have assurances of each other's identities and furthermore, optionally share a secret key known only to them. For every completed run, each party either accepts or rejects the other's identity and optionally an exchanged key. In a successful run, the run is completed and both parties accept.

Property 1 of a successful run: Both Alice and Bob accept each other's identities. If the authentication involves key exchange, then they both accept the exchanged key also.

The second property of a successful run concerns the records of a protocol run (assuming the participants had each recorded the exchange). To proceed, we require definitions regarding the use of the work match when applied to records of a run (a slightly different definition is given by Bird et al. [5]).

Matching Messages: We say that a message from one record matches a message from another if one record lists the message as incoming, the other record lists the message as outgoing, and all fields of the message relevant to authentication are the same in both records.

The qualification relevant to authentication is necessary to allow individual messages to match even if they are not bit-wise identical. The motivation here is that if a message contains unsigned fields that are cryptographically irrelevant to authentication, then discrepancies in such fields alone should not preclude a message from meeting the definition of matching.

Matching Records of Runs: We say that two records of a run match if their messages can be partitioned into sets of matching messages (each set containing one message from each record), the messages originated by one participant

Nintendo - Ancora Exh. 1003

appear in the same order in both records, and the messages originated by the other participant appear in the same order in both records. For simplicity, we do not consider protocols in which messages need not arrive in the order in which they were sent.

Note that messages originated by distinct participants do not have to be in the same order with respect to each other. This allows the case where messages in transit cross. In such a case, each participant will record his own message as having been sent before the crossing message is received.

Property 2 of a successful run: If Alice and Bob have recorded the exchange, then their records of the run will match.

We now distinguish between a successful run and a secure run. To consider a run successful by any reasonable definition, the run must be considered secure in the intuitive sense. On the other hand, it is possible for a run to be unsuccessful even in the absence of security breaches (e.g., if both legitimate parties reject for some reason). It is also always possible that an enemy may delay a legitimate message of a run indefinitely. Suppose that in a particular run, Alice accepts Bob's identity, sends the last message of the protocol to Bob, and then an enemy destroys this message. Assuming Bob must receive this message before accepting Alice's identity, Bob will not accept Alice's identity. Intuitively, while this run is unsuccessful, there have been no security breaches; at the time that Alice accepted Bob's identity (before she sent the last message), Bob's record of the partial run matched Alice's record. For our purposes, such a denial of service attack in itself is not considered a security breach; such problems often must be dealt with by physical security and other techniques.

We are now in a position to define what it means for a run of a (symmetric or asymmetric) mutual authentication protocol to be insecure:

DEFINITION 1: A particular run of a protocol is an insecure run if any party involved in the run, say Alice, executes the protocol faithfully, accepts the identity of another party, and either of the following conditions holds:

At the time that Alice accepts the other party's identity (before she sends or receives a subsequent message), the other party's record of the partial or full run does not match Alice's record.

The exchanged key accepted by Alice is known to someone other than the party whose identity Alice accepted. (This condition does not apply to authentication without key exchange.)

Note that under this definition a conventional key exchange protocol requiring a trusted third party [18] is not secure.

It should be clear that Alice's record, which must match that of the other party in the above definition, is the actual record she has at the point in time at which she has received enough information to carry out any computations required to reach the accept state: messages sent or received subsequent to this are irrelevant.

The goal of the enemy is to cause a run to be insecure. The goal of the designer of the protocol is to make the enemy's task impossible (or computationally infeasible) in all instances. Reversing Definition 1, we get a definition of a secure (symmetric or asymmetric) mutual authentication protocol:

DEFINITION 2: A secure protocol is a protocol for which the following conditions hold in all cases where one party, say Alice, executes the protocol faithfully and accepts the identity of another party:

At the time that Alice accepts the other party's identity (before she sends or receives a subsequent message), the other party's record of the partial or full run matches Alice's record.

It is computationally infeasible for the exchanged key if accepted by Alice to be recovered by anyone other than Alice and possibly the party whose identity Alice accepted. (This condition does not apply to authentication without key exchange.)

By themselves, the above definitions are not particularly helpful in deciding whether a given protocol is secure, in that they do not lead to constructive procedures to either verify or expose weaknesses of a protocol. Nonetheless, these definitions can be applied directly in deciding whether a given potential attack is a real attack. For example, in an authentication with key exchange, suppose an enemy merely intercepts Alice's and Bob's messages and then passes them along unchanged. Intuitively, the enemy has not compromised the system in this case; the parties have accepted each other's identities, have matching records of the run, and exclusively share a secret key. Note that by Definition 1, such a run is not insecure. In other cases a supposed attack may become quite convoluted, and it may not be obvious that the attack amounts to just passing along messages. Definition 1 can be used to distinguish such a pass-along nonattack. In Section 5, this definition serves well in identifying real attacks; in particular, the second condition, which appears trivial, is essential.

While formal analysis techniques have been successfully used to uncover weaknesses in some authentication protocols (see Section 6), proof of correctness is more difficult, and depends heavily on proper modelling of goals and assumptions. Another technique available for uncovering weaknesses is that of exhaustive search with respect to interleaving attacks [5]. Unfortunately, since there are as yet no absolute proofs of correctness, confidence in a protocol develops only over time as experts conduct a continuing analysis of the protocol and fail to find flaws.

4. Desirable Protocol Characteristics

In addition to being secure, there are other desirable characteristics for a protocol.

Perfect Forward Secrecy. An authenticated key exchange protocol provides perfect forward secrecy if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs. The property of perfect forward secrecy does not apply to authentication without key exchange.

Direct Authentication. In some authenticated key exchange protocols, authentication is not complete until both parties prove knowledge of the shared secret key by using it in subsequent communications. Such a protocol is called indirect. When authentication is established by the end of each protocol run, the protocol is direct. An indirect protocol can be modified to be direct by adding an exchange of known messages or messages with redundancy encrypted with the exchanged key. For authentication without key exchange, an indirect protocol provides no security because neither party can accept the other's identity.

No Timestamps. While timestamps are convenient for administrative and documentation purposes, it is desirable in practice to avoid relying on their use for security in authentication protocols. Difficulties, precautions, and objections to timestamps are well-documented in the literature [3], [5], [13]. For convenience, we summarize the more notable issues below.

To use timestamps for authentication, all parties must maintain local clocks that are periodically synchronized in a secure manner with a reliable source of time. Between synchronizations with the reliable time source, local clocks

Nintendo - Ancora Exh. 1003

may drift. Two parties, Alice and Bob, must allow a time window for timestamps to compensate for local clock drift and the fact that messages take time to cross a network. Alice will accept any timestamp from Bob that is within a window around the time on Alice's local clock as long as Bob has not used this particular time value before. Alice can either store all time values used by all other parties that are within her current window (which is impractical in some communications environments) or she can store the latest time used by each party and insist on strictly increasing time values from each party. However, in the strictly increasing time values case, if Bob uses a time t far into the future for some reason (e.g., severe clock drift or improper synchronization with the reliable time source), then Bob will not be able to communicate with Alice until time t is within her window. To prevent this problem, Alice would have to store time t and not update her record of the latest time value used by Bob. This could potentially lead to a choice among storing large quantities of data, sacrificing communications availability, or sacrificing security. Concerning communications availability, if two parties' local clocks are too far out of synchronization, then the parties cannot communicate. This tends to make those concerned with communications availability want wide time windows which increases storage requirements. While timestamps are convenient from a theoretical point of view, they present a number of practical problems. Protocols based on random challenges do not suffer from these difficulties.

Recently, formal analysis has been used in the verification of authentication protocols [7], [13]. Starting with a list of initial formal beliefs, the objective is to logically derive the stated protocol goal by consuming the list of protocol steps. One of the basic assumptions on which such analysis is typically based is that the parties involved have the ability to check the freshness of timestamps. In fact, one of the main results of the work by Gaarder and Snekkenes is the identification of the security requirement that time clocks be trustworthy in certain protocols. This means that in practice, the security of timestamp-based protocols relies heavily on the proper implementation of synchronized and secure time clocks. Unfortunately, despite much discussion in the literature regarding timestamp-based protocols (e.g., [8], [16]), when it comes to actually implementing such a protocol, the significance of the security of time clocks is easily lost, and furthermore, the costs associated with a proper implementation can be significant.

5. Station-to-Station Protocol

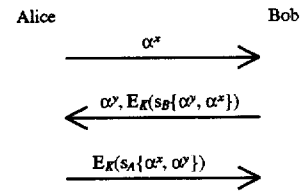
We now introduce a simple, efficient authenticated key exchange protocol called the station-to-station (STS) protocol. The STS protocol has evolved over time; an early version of this work was described at the 1987 International Switching Symposium [21]. We believe that it is secure according to Definition 2 and has a number of other desirable properties. In the remainder of this section, we describe the protocol, discuss its properties, and justify its subtle details by showing how variants of it are vulnerable.

5.1. Basic STS Protocol

The STS protocol consists of Diffie-Hellman key establishment [9], followed by an exchange of authentication signatures. In the basic version of the protocol, we assume that the parameters used for the key establishment (i.e., the specification of a particular cyclic group and the corresponding primitive element α) are fixed and known to all users. While we refer to the Diffie-Hellman operation as exponentiation, implying that the underlying group is multiplicative, the description applies equally well to additive groups (e.g., the group of points of an elliptic curve over a finite field). We also assume in this section that Alice knows Bob's authentic public key, and vice versa; this assumption is dropped in the following section.

The protocol begins with one party, Alice, creating a random number x and sending the exponential α^x to the other party, Bob (see diagram below). Bob creates a random number y and uses Alice's exponential to compute the exchange key $K=\alpha^{xy}$. Bob responds with the exponential α^y and a token consisting of his signature on the exponentials, encrypted with K using a suitable symmetric encryption algorithm E (i.e., $E_K(s_B\{\alpha^y, \alpha^x\})$). Alice computes K, decrypts the token using K, and verifies Bob's signature using Bob's public key. Alice sends to Bob her corresponding encrypted signature on the exponentials, $E_K(s_A\{\alpha^x, \alpha^y\})$. Finally, Bob similarly verifies Alice's encrypted signature using K and Alice's public key. The security of the exponential key exchange relies on the apparent intractability of the discrete logarithm problem [24].

Basic STS Protocol



It is possible to create a more symmetric version of this protocol where the parties exchange exponentials first and then exchange encrypted signatures in separate messages. This would make it permissible for the exponential messages to cross, and then the encrypted signature messages to cross. In such a case, neither Alice nor Bob need know who initiated the call. This is desirable, as situations exist in practice (e.g., in both voice telephony and X.25 data transfer) in which at certain implementation levels, it is not known which party initiated a call. This explains why each party forms his signature with his own exponential listed first. If the exponentials were in the same order in both signatures, then Alice and Bob would have to find a way to agree on whose exponential should be listed first (such as by basing the decision on which party initiated the call).

At this point, consider what assurances the STS protocol provides to the participants. From Bob's point of view, as a result of the Diffie-Hellman key exchange, he shares a key known only to him and the other participant, who may or may not be Alice. Our assumption in this section is that Bob knows Alice's public key (this is achieved in the section below through use of certificates). Because Alice has signed the particular exponentials associated with this run, one of which Bob himself has just created specifically for this run, her signature is tied to this run of the protocol. By encrypting her signature with K, Alice demonstrates to Bob that she was the party who created x. This gives Bob assurance that the party he carried the key exchange out with was, in fact, Alice. Alice gets a similar set of assurances from Bob.

The STS protocol has the desirable characteristics discussed in Section 4. Rather than using timestamps, challenges are used. Because the parties demonstrate knowledge of the exchanged key by encrypting their signatures, the authentication is direct. The STS protocol also offers perfect forward secrecy. The only long-term secret keying material stored by users is their secret keys for the signature scheme. If a secret key is compromised, the security of exchanged keys from earlier runs is not affected because Diffie-Hellman key exchange is used; Diffie-Hellman key exchange has no

Nintendo - Ancora Exh. 1003

long-term keying material. There are two other desirable properties of the STS protocol. The first is that public key techniques are used to make key management simpler and more secure than is possible using conventional cryptography. If parties generate their own secret keys, these keys need never be disclosed (to anyone, including any supposedly trusted party), even during initialization. The second is that there is no need for communicating parties to contact a central facility on a per-call basis. If certificates are used for distributing public keys (see Section 5.2), once a party has its own certificate and the trusted authority's public key, it can exchange keys with, and authenticate other parties without consulting a central facility. The protocol appears to strike an elegant and difficult balance, being simple and secure without utilizing unnecessary or redundant elements.

To illustrate the need for the features of the STS protocol, it is now demonstrated how the protocol is weakened when the following modifications are made: removing the encryption of the signatures, signing only one's own exponential, signing only the other party's exponential, or uncoupling authentication from key exchange.

Removing encryption on signatures. Consider a modified STS protocol where the signatures on the exponentials are not encrypted with the exchanged key K. Because the exponentials are public information, any other party could sign them as well. Suppose that in the last message of the protocol, an enemy Eve substitutes her own signature on the exponentials for Alice's signature. (If the parties exchange public keys using certificates, Eve would have to substitute her own certificate for Alice's certificate.) This may not seem like a serious attack, as Eve does not know the exchanged key. However, if Bob were a bank, Eve could get credit for a deposit Alice might make. Interestingly, even though Bob has been misled here, Alice is the party who may be hurt.

Having informally discussed why the above run is insecure, we now apply Definition 1. Bob executed the protocol faithfully and accepted Eve's identity, but the exchanged key is known to a different party, Alice. By Definition 1, the run is insecure. Because an insecure run is possible, the modified protocol is insecure.

Signing only one's own exponential. Consider the variant of the STS protocol where each party signs only his own exponential (i.e., Alice's encrypted signature is $E_K(s_A\{\alpha^x\})$ and Bob's is $E_K(s_B\{\alpha^y\})$. We know of no general attack that applies to this case, but there is an attack that applies when the signature scheme is RSA[26], the hash function is the identity function, and Diffie-Hellman key exchange is carried out over GF(p). In this case, Eve can impersonate Alice in a run with Bob by using x=0 as the exponent in the key exchange. Eve's exponential is $\alpha^0=1$, and the exchanged key is $K=\alpha^{xy}=1$. Eve requires the following encrypted signature

$$\begin{aligned} E_K(s_A\{\alpha^X\}) &= E_1(s_A\{1\}) \\ &= E_1(s_A(1)) && \text{because the hash function is the identify function} \\ &= E_1(1) && \text{because signing in } RSA \text{ in exponentiation and } 1^z = 1 \text{ for all } z \end{aligned}$$

Eve can compute $E_1(1)$, and hence can impersonate Alice. Although this attack applies only to a specific case, it illustrates a more general problem in signing only one's own exponential: if Eve can obtain a quantity for which she can acquire or compute the discrete logarithm, and can acquire or compute Alice's signature on the quantity, then Eve can use (and reuse) this quantity as an exponential to impersonate Alice. By introducing the second exponential into the data to be signed, an adversary is forced to solve a different instance of the problem in real time each time impersonation is attempted.

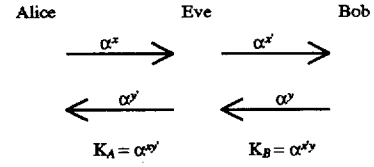
Signing only the other party's exponential. Consider the variant of the STS protocol where each party signs only the other party's exponential (i.e., Alice's encrypted signature is $E_K(s_A\{\alpha^y\})$ and Bob's is $E_K(s_B\{\alpha^x\})$). Again, we know of no general attack which applies to this case, but there are some concerns.

In principle, it is imprudent to sign arbitrary text supplied by a potential adversary. In the case at hand, in order for an adversary to recover the signature, he would have to know the key K. To compute K, the adversary would need to know the discrete logarithm of the quantity being signed. While an adversary would not in general know the logarithm of a particular fixed quantity he might desire signed, it is trivial to produce such quantities by preselecting logarithms, and it would appear undesirable to allow an adversary the freedom to acquire signatures on any quantities whose logarithms are known.

In Section 5.3, it is shown that the STS protocol can be reduced to an authentication-only protocol by replacing exponentials with random numbers and removing the encryption on the signatures. If each party were to sign only the other party's exponential, then the authentication-only variant would be subject to the attack on the simple challenge-response outlined in Section 2. Similarly, signing only one's own exponential does not result in a protocol which reduces to a secure authentication-only variation.

Note that even should it turn out that signing both exponentials does not provide more security than simply signing a single exponential, the only added cost in doing the former is additional hashing, which in general is relatively minor. No additional operations involving the signature scheme, symmetric cryptosystem operations, or data transmission are introduced by signing both exponentials rather than one only.

Uncoupling authentication from key exchange. If the STS protocol is modified so that authentication is uncoupled from key exchange by having the parties sign some quantity that is independent of the exponentials, the resulting protocol is subject to the classical intruder-in-the-middle attack (e.g., [27]) on Diffie-Hellman key exchange:



Eve substitutes her own exponentials for Alice's and Bob's exponentials. This results in Alice and Bob calculating two different keys, both of which can be calculated by Eve. Eve shares key K_A with Alice, and key K_B with Bob. During the authentication phase of the run, Eve can pass Alice's encrypted messages to Bob and vice versa by decrypting the messages with one key and re-encrypting with the other. After authentication, Eve is free to passively eavesdrop or to inject her own messages. By Definition 1, this modified protocol is insecure because while Alice executed the protocol faithfully and accepted Bob's identity, the exchanged key is shared with a different party, Eve. There is a similar problem from Bob's point of view.

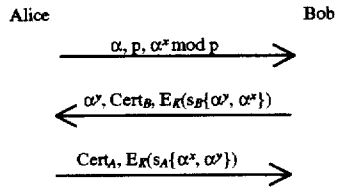
Nintendo - Ancora Exh. 1003

5.2. STS Protocol in Practice

We now describe the use of the STS protocol in practice, for the specific case where the key exchange is carried out in the multiplicative group of a finite field. For clarity, we focus on prime fields GF(p). Two parameters are required then for Diffie-Hellman key exchange: a primitive element e in GF(p), and a suitable prime p. The prime p should be chosen to preclude Pohlig-Hellman type attacks [25]. In light of recent work on the discrete logarithm problem ([24] for prime fields; [23] for fields of characteristic two), it is prudent to use a distinct field for each user (i.e., for GF(p), a distinct prime p, chosen by the user himself). The best known attacks on Diffie-Hellman key exchange over finite fields are the index-calculus techniques involving a massive pre-computation which yields a database specific to a particular field. The database then allows computation of individual logarithms in that field relatively quickly. If a single field is used for an entire network, a single database allows the compromise of all key exchanges—providing great incentive to attempt to construct the database.

To facilitate the distribution of users' public keys and user-specific Diffie-Hellman parameters, certificates may be used. In addition to these items, a certificate should contain the user's name and the signature of the trusted authority over these data items. The reason for the inclusion of the (α, p) pair in the certificate is explained below. The STS protocol is then as follows. To avoid cluttering the formulae the mod p reductions have been omitted.

STS Protocol in practice:



$Cert_A$ = (Alice, pA, α, p, s_T {Alice, p_A, α, p})

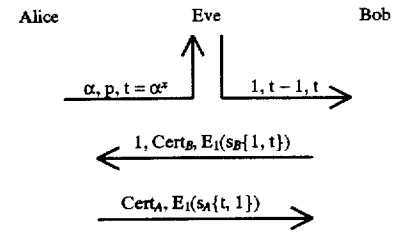
The differences here are as follows. Alice sends her Diffie-Hellman parameters along in the first message; Bob uses these instead of fixed network-wide parameters. Upon receiving the third message, Bob verifies that the Diffie-Hellman parameters sent in the first message agree with those actually in Alice's certificate. In the second message, Bob sends Alice his certificate, from which Alice can extract his authentic public key; Alice verifies authenticity by checking the signature of the trusted authority on Bob's certificate. Similarly, in the third message, Alice sends Bob her certificate, allowing Bob to extract her authenticated public key, after similarly verifying the trusted authority's signature on her certificate. Note that Bob does not need Alice's certificate until the third message, and in fact may not wish to receive it earlier, since this may require having to allocate storage to save the certificate until needed upon receipt of the third message. A further reason for Alice to delay sending her certificate until the third message is to allow both Alice and Bob the option to encrypt their certificates with the exchanged key. Although certificates are, in theory, public information, it may be desirable in some applications to prevent an eavesdropper from seeing them in order to prevent a passive eavesdropper from learning Alice and Bob's identities.

Note that knowledge of the other party's public key is not required to construct and send or to receive and process the first message. If the public key were required at this stage, then introducing certificates would necessitate an additional preliminary message to make the certificate available earlier.

As discussed in Section 5.1, it may be desirable in some cases to allow both parties to send the initial message simultaneously. In this case, some method must be used to establish one of the parties as the dominant party (i.e., the party whose α, p pair will be used). The nondominant party would then continue the protocol with the second message. An example of a simple method would be to choose the party with the larger prime p to be dominant.

It is now shown that the protocol is weakened if Diffie-Hellman parameters are not included in certificates.

Removing Diffie-Hellman parameters from certificates. Without Diffie-Hellman parameters in certificates, the enemy, Eve, has the freedom to modify α and p in Alice's first message. Let Alice's exponential be $t=(\alpha^x \bmod p)$. Suppose that Eve changes α to be 1 and p to be t−1 (see diagram below). Then Bob's exponential is $1^y \bmod (t-1)=1$, and Bob calculates the exchanged key to be $t^y \bmod (t-1)=1$. Alice calculates the exchanged key to be $1^x \bmod p=1$. Because Eve does not modify the exchanged exponentials and Alice and Bob calculate the same exchanged key, Alice and Bob will accept each other's encrypted signatures.



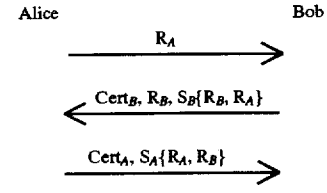
Eve knows the exchanged key and after authentication, she is free to both eavesdrop and inject her own messages. Note that Alice and Bob accepted each other's identities, but their records of the run do not match, and the exchanged key is known to a third party; the modified protocol is thus insecure by our definitions, as well as intuitively.

While it may appear that the above-described substitution is trivial and easily detected by special checks, the potential for compromise remains. More sophisticated or disguised related attacks appear possible, including the possible use of Pohlig-Hellman-weak primes. The fundamental concern is that in order to rely on the believed intractability of the Diffie-Hellman problem, it must be ensured that suitable Diffie-Hellman parameters are in fact used.

5.3. Authentication-Only Version of STS Protocol

It is possible to turn the STS protocol into an authentication-only protocol by replacing the exponentials with random numbers and removing the encryption on signatures:

Authentication-only STS Protocol



This simplified protocol is essentially the same as the three-way authentication protocol currently proposed by ISO [1]. This is discussed further in the following section.

6. Discussion of Other Protocols

From the intruder-in-the-middle attack on unauthenticated Diffie-Hellman key exchange to spoofs in the spirit of

Nintendo - Ancora Exh. 1003

the well-known "grandmaster postal-chess" problem.[1] attacks on authentication protocols are numerous and well-documented in the literature. Burrows, Abadi and Needham analyzed eight protocols and found six to contain redundancies, and four to contain flaws [7, Table 1], including both redundancies and flaws in the CCITT X.509 mutual authentication protocols [30]. To get a flavor of the concerns we have with many of the currently proposed protocols, we briefly discuss two of the four protocols analyzed by Burrows et al.; Kerberos, and one of the X.509 protocols. We also discuss a related ISO protocol.

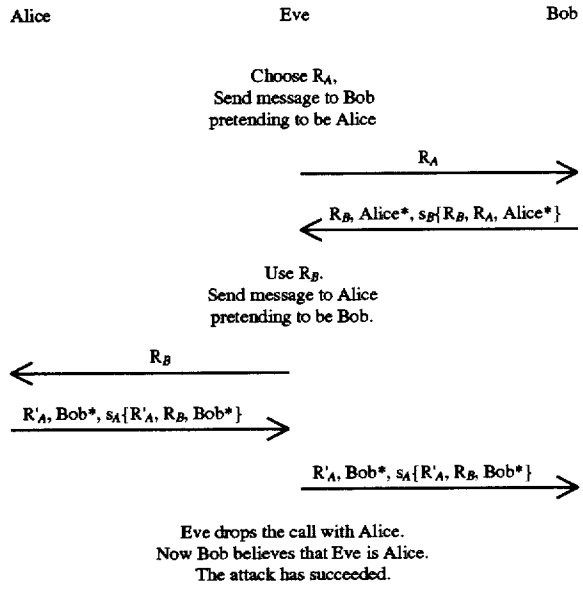
Kerberos protocol. The popular Kerberos protocol [18], based on symmetric cryptosystems, has several features which make it somewhat undesirable in various applications. These include the use of timestamps (discussed earlier), the requirement of an on-line authentication server, and redundancies in the protocol itself. These and further issues are discussed by Bellovin and Merritt [3].

Three-pass CCITT X.509 authentication protocol. The CCITT X.509 recommendation [30] is a very widely known internationally standardized authentication protocol based on public-key cryptography. The one and two-pass X.509 protocols require timestamps, while timestamps are redundant in the three-pass protocol; the specification allows that the timestamp field may be zero in this latter case (making the three-pass protocol practical, although it would be preferable if no field at all had to be allocated for timestamps). Some concerns regarding the protocol are now summarized. The final message of this protocol is Alice's signature on both Bob's challenge and Bob's identity: $s_A\ \{R_B, \text{Bob}\}$.[2] This allows Bob to obtain the signature of Alice on a quantity over which Bob has control. This is undesirable, although it is not clear how to use this to mount a direct attack. A second concern involves the suggested use of the optional encrypted data field in the protocol to accomplish key exchange; this use does not guarantee perfect forward secrecy.[3] A further issue with the use of this field is that there is no guarantee that the sender of the encrypted data actually knows the encrypted data itself, and in fact an adversary can pass off another party's encrypted data as his own [7], [13]. A third concern [17] is the restriction that the signature system used must be capable of both signing and encrypting data, which rules out many candidate signature schemes including the proposed NIST Digital Signature Algorithm [10].

ISO three-way protocol. As noted in Section 5.3, the authentication-only version of the STS protocol is essentially the same as the three-way protocol currently proposed by ISO [1]. The differences are that the ISO protocol allows redundant copies of the random numbers, optional fields for the identity of the intended recipient of a message, and optional fields for arbitrary text. Due to limitations of authentication-only protocols as discussed earlier, in most applications it is expected that the key establishment functionality of the ISO protocol (provided by the optional text fields both within and outside the signed portion of each message) will be employed. Recalling the concern noted above in X.509, care must be taken in the use of these fields; furthermore, note that their use to transfer encrypted session keys does not guarantee perfect forward secrecy.

Attack on a specific authentication protocol. To augment the literature documenting attacks on specific protocols, and to further emphasize how easily flaws can be introduced and overlooked, we now consider the following (flawed) variation of the ISO authentication exchange. In fact, this variation was a preliminary version of the protocol. Here, Alice is allowed to use a new random number R'_A in place of R_A in the third message; R'_A is then also sent along as an additional cleartext field in the third message. In this modified protocol, an enemy Eve can authenticate herself as Alice to an unsuspecting party Bob as follows (see diagram below). Eve call Bob, pretending to be Alice, sending a challenge to Bob; Eve responds to Bob's counter-challenge by calling Alice and getting her to respond correctly to the challenge; Eve then drops the call with Alice and passes the correct response along to Bob, thus completing the authentication from Bob's point of view. Note that the attack is successful even if the identity of the intended recipient of each message is incorporated within the signed portion of each authentication token, as is optionally permissible in the formal definition of the related ISO protocol. To emphasize this, these principals' identities are included, and annotated with asterisks, in the attack detailed below. For simplicity, certificates are not shown.

Regarding other attacks documented in the literature, we note that Bird et al. ([5], Section 4) detail on attack on a specific protocol. This is a specific case of the general class of refection attacks in which a challenger is tricked into providing answers to his own questions [19].



7. Concluding Remarks

Below are some general principles that appear prudent to follow in the design of authentication protocols. While many of these have been previously observed, we find it convenient to collect them here.

1. Authentication and key exchange must be linked. If authentication and key exchange are independent, then an attacker could allow two parties to carry out authentication unhindered, and could take over one party's role in key exchange. This would allow the attacker to impersonate a valid party after authentication and key exchange are completed.

2. Asymmetry in a protocol is desirable. Symmetries in a protocol should be used with caution, due to both the possibility of reflection attacks, and attacks in which responses from one party can be reused within a protocol. As an obvious illustrative example, the authentication responses of each of two parties should not be identical.

3. Messages within a particular protocol run should be logically linked or chained in some manner, to prevent the reuse of previous messages or the introduction of messages

Nintendo - Ancora Exh. 1003

from a parallel run. The objective here is to preclude replay attacks and interleaving attacks. Messages should also be linked to the current time frame (e.g., through incorporation of recently generated random numbers). The specific attack detailed in Section 6 is possible due to a lack of such chaining of messages; similarly, the middleperson attack discussed by Gengio et al. [4] is possible in protocols which fail to address this principle.

4. A party carrying out a cryptographic operation (serving as a signature) should be able to incorporate into the data being operated on a reasonable amount of data which he himself randomly selects. In other words, a protocol should not require a party to carry out a cryptographic operation on inputs which may be entirely under the control of an adversary. This "add your own salt" principle is aimed at preventing an adversary from obtaining responses to specific questions he himself may not be able to answer. This should also prevent so-called chosen-ciphertext attacks ([6, p. 27]). Related to this principle, we note the following principle paraphrased from Moore [20, section II]:

5. Valid signatures should result from the transformation of a message from a message space that is a sparse subset of the domain of the signature function. For example, requiring redundancy, or some other expectation, in the data to be signed, may thwart attacks whereby an adversary attempts to forge new signatures by combining previously obtained valid signatures. For the STS protocol, the hash function selected to hash the exponentials should produce a result smaller than the maximum size of input allowed to the signature process, to allow redundancy to be added to the hash result before signing.

The proposed station-to-station protocol satisfies the above principles, as well as the desirable properties noted in Section 4 (perfect forward secrecy, direct authentication, no requirement of timestamps). Its compatibility with the emerging ISO authentication protocol, and its ability to provide key establishment within this framwork, add to its appeal. Furthermore, the station-to-station protocol uses the minimum number of messages required for a random-number-based challenge-response mutual authentication (three), and requires only one signature generation, one signature verification, and two encryption operations by each party (with an additional signature verification if certificates are used on a per-run basis to bind a user's identity and public key).

Any appropriate signature scheme may be used in the STS protocol, including the Digital Signature Algorithm (DSA) recently proposed by NIST [10]. For reasons of practical efficiency, an obvious candidate signature scheme is RSA [26]. Similarly, any appropriate symmetric encryption algorithm may be used. In some applications it may be desirable to avoid the use of an encryption algorithm. One method to consider for avoiding the need the need for an encryption algorithm E_K is as follows: replace the encrypted signature by a signature plus a message authentication code (MAC) over the signature; i.e., replace $E_K(s)$, where $s=s_B\{\alpha^Y, \alpha^X\}$ (as in Section 5.1), by $(s, M_K(s))$, where M_K is a MAC with key K. The receiving party would then verify both the signature and the MAC over the signature. While allowing one to avoid the requirement of an encrypt/decrypt capability (which e.g., both Kerberos and the X.509 protocols require), a disadvantage of this approach is the additional data transfer it entails.

Acknowledgments

The authors would like to thank their colleagues for their support and ideas related to the protocols in question, and in particular discussions with Carlisle Adams and Warwick Ford.

Notes

* This work was done while Whitfield Diffie was with Northern Telecom, Mountain View, Calif.

1. A novice who engages in two simultaneous chess games with two distinct grandmasters, playing white pieces in one game and black in the other, can take his opponents' moves in each game and use them in the other to guarantee himself either two draws or a win and a loss, and thereby unfairly have his chess rating improved.

2. In an early version of X.509, the final message was simply $s_A\{R_B\}$; the recommendation has since been formally updated.

3. Note that use of RSA [26] in the obvious manner to achieve key exchange similarly does not guarantee perfect forward secrecy.

References

1. Information Technology—Security Techniques. Entity Authentication Mechanisms—Part 3: *Entity Authentication Using a Public-Key Algorithm* (CD 9798-3), November 1991 (ISO/IEC JTCI/SC27 Committee Draft #4).
2. Bauspiess, F. and Knobloch, H.-J. 1990. How to keep authenticity alive in a computer network. Advances in Cryptology—Eurocript 89, (J. J. Quisquater and J. Vandewalle, eds.) *Lecture Notes in Computer Science* 434: 38–46, Berlin/New York: Springer-Verlag.
3. Bellovin, S. M. and Merritt, M. 1990. Limitations of the Kerberos authentication system. *ACM Computer Communication Review* 20 (5):119–132.
4. Bengio, S., Brassard, G., Desmedt, Y. G., Coutier, C., Quisquater, J.-J. 1991. Secure implementation of identification system. *J. Cryptology* 4 (3):175–183.
5. Bird, R., Gopal, I., Herzberg, A., Janson, P., Kutten, S., Molva, R., and Yung, M. Forthcoming. Systematic design of two-party authentication protocols. *Advances in Cryptology—Crypto* '91, Berlin/New York: Springer-Verlag.
6. Brassard, G. 1988. *Modern Cryptology*, Lecture Notes in Computer Science 325. Berlin/New York: Springer Verlag.
7. Burrows, M., Abadi, M., and Needham, R. 1990. A logic of authentication. *ACM Transactions on Computer Sciences* 8 (1):18–36.
8. Denning, D. E. and Sacco, G. M. 1981. Timestamps in key distribution protocols. *Comm. ACM* 24 (8):533–536.
9. Diffie, W. and Hellman, M. E. 1976. New directions in cryptography. *IEEE Trans. Info. Theory* IT-22 (6):644–654.
10. (proposed U.S. FIPS) Digital Signature Standard (DSS), announced in *Federal Register*, vol. 56, no. 169 (Aug. 30, 1991), 42980–42982.
11. ElGamal, T. 1988. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Info. Theory* IT-31 (4):469–472.
12. Fiat, A. and Shamir, A. 1987. How to prove yourself: practical solutions to identification and signature problems. *Advances in Cryptology—Crypto* 86 (A. Odlyzko, ec.), *Lecture Notes in Computer Science* 263:196–194, Berlin/New York: Springer-Verlag.
13. Gaarder, K. and Snekkenes, E. 1991. Applying a formal analysis technique to CCITT X.509 strong two-way authentication protocol. *J. Cryptology* 3 (2):81–98.
14. Guillou, L. C.and Quisquater, J.-J. 1988. A practical zero-knowledge protocol fitted to security microprocessing minimizing both transaction and memory. *Advances in Cryptology—Eurocrypt* '88, (C. G. Günther, (ed.), *Lecture Notes in Computer Science* 330:123–128, Berlin/New York: Springer-Verlag.

Nintendo - Ancora Exh. 1003

15. Günther, C. G. 1990. An identity-based key-exchange protocol. *Advances in Cryptology—Eurocrypt* 89, (J.-J. Quisquater and J. Vanewalle, eds.). *Lecture Notes in Computer Science* 434:29–37, Berlin/New York:Springer-Verlag.

16. Haber, S. and Stornetta, W. S. 1991. How to time-stamp a digital document. *J. Cryptology* 3 (2):99–111.

17. I'Anson, C. and Michell, C. 1990. Security defects in CCITT Recommendation X.509—The Directory Authentication Framework. *Computer Communication Review* 20 (2):30–34.

18. Kohl, J. and Neuman, B. C. 1991. The Kerberos network authentication service. MIT Project Athena Version 5.

19. Mitchell, C. 1989. Limitations of challenge-response entity authentication. *Electronic Letters* 25 (17):195–196.

20. Moore, J. H. 1988. Protocol failures in cryptosystems. *Proc. of the IEEE* 76 (5):594–602.

21. O'Higgins, B., Diffie, W., Strawczynski, L. and de Hoog, R. 1987. Encryption and ISDN-A Natural fit. *In Proc.* 1987 *International Switching Symposium*, Pheonix Ariz., pp. A1141–7.

22. Okamoto, E. and Tanaka, K. 1989. Key distribution system based on identification information. *IEEE J. Selected Areas in Comm.* 7 (4):481–485.

23. Odlyzko, A. M. 1985. Discrete logarithms in finite fields and their cryptographic significance. *Advances in Cryptology—Eurocrypt* 84, (T. Beth, N. Cot and I. Ingemarsson, eds.), *Lecture Notes in Computer Science* 209:224–314, Berlin/New York: Springer-Verlag.

24. LaMacchia, B. A. and Odlyzko, A. M. 1991. Computation of discrete logarithms in prime fields. *Designs, Codes and Cryptography I* (1):47–62.

25. Pohlig, S. C. and Hellman, M. 1978. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory* IT-24:106–110.

26. Rivest, R. L., Shamir, A. and Adleman, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM* 21:120–126.

27. Rivest, R. L. and Shamir, A. 1984. How to expose an eavesdropper. *Comm. ACM* 27 (4):383–395.

28. Schnorr, C.P. 1990, 1991. Efficient signature generation by smart cards. *J. Cryptology* 4 (3):161–174; see also: Efficient identification and signatures for smart cards. *Advances in Cryptology—Crypto* 89, (G. Brassard, ed.), *Lecture Notes in Computer Science* 435:239–251, Berlin/New York: Springer-Verlag.

29. Shamir, A. 1985. Identity-based cryptosystems and signature schemes. *Advances in Cryptology—Crypto* 84, (G. R. Blakley and D. Chaum, ed.), *Lecture Notes in Computer Science* 196:47–53, Berlin/New York: Springer-Verlag.

30. *CCITT Blue Book Recommendation* X.509, *The Director-Authentication Framework*, 1988. Geneva, March 1988; amended by resolution of Defect 9594/016 (IQ 1991). Also ISO 9594-8.

I claim:

1. A computer-readable medium storing a data structure for secure distribution of software from a distributor to a recipient, said data structure comprising:

(a) a cryptographically secured representation of said software, said cryptographically secured representation having been secured by a first encryption key;

(b) a cryptographic certification, by a certifier, of a first decryption key corresponding to said first encryption key; and

(c) an identifier of said distributor; said cryptographically secured representation, cryptographic certification and identifier collectively defining a software passport which enables said recipient thereof (i) to cryptographically verify said first decryption key using a second, preexisting decryption key unrelated to said distributor and obtained by said recipient without specific knowledge of said certifier, and (ii) to cryptographically verify said software using said verified first decryption key.

2. The computer-readable medium of claim 1 wherein said software passport includes said first decryption key.

3. The computer-readable medium of claim 2 wherein said software includes a binary representation of a computer program.

4. The computer-readable medium of claim 3 wherein said software passport includes a validity date of said computer program.

5. The computer-readable medium of claim 2 wherein said first decryption key and said first encryption key are a public-private cryptographic key pair.

6. The computer-readable medium of claim 5 wherein said cryptographically secured representation includes a message digest of at least a portion of said software, said message digest having been encrypted with said first encryption key.

7. The computer-readable medium of claim 2 wherein said cryptographic certification is secured by a second encryption key corresponding to said second decryption key, and wherein said second keys are a private-public cryptographic key pair.

8. The computer-readable medium of claim 2 wherein said cryptographic certification includes a message digest of said first decryption key, said message digest having been encrypted with said second encryption key.

9.The computer-readable medium of claim 2 wherein said software has been encrypted.

10. The computer-readable medium of claim 9 wherein said software is subject to intellectual property protection.

11. The computer-readable medium of claim 9 wherein said sofaware is subject to an access fee.

12. The computer-readable medium of claim 2 wherein said cryptographic certification includes said identifier.

13. The computer-readable medium of claim 2 wherein said identifier includes information about said software.

14. The computer-readable medium of claim 2 wherein said cryptographic certification includes a validity date thereof.

15. The computer-readable medium of claim 1 wherein said cryptographic certification represents an assurance of a skill of said distributor by said certifier.

16. The computer-readable medium of claim 15 wherein said first decryption key and said first encryption key are a public-private cryptographic key pair.

17. The computer-readable medium of claim 16 wherein said cryptographically secured representation includes a message digest of at least a portion of said software, said message digest having been encrypted with said first encryption key.

18. The computer-readable medium of claim 15 wherein said cryptographic certification is secured by a second encryption key corresponding to said second decryption key, and wherein said second keys are a private-public cryptographic key pair.

19. The computer-readable medium of claim 18 wherein said cryptographic certification includes a message digest of said first decryption key, said message digest having been encrypted with said second encryption key.

20. The computer-readable medium of claim 15 wherein said software has been encrypted.

21. The computer-readable medium of claim 1 wherein said second decryption key is stored at a computing platform of said recipient, and where said certifier is a provider of least a portion of said computing platform.

22. The computer-readable medium of claim 21 wherein said second decryption key is a public key of said platform provider and where said first decryption key and said first encryption key are a public-private cryptographic key pair.

23. A method for secure software distribution from a distributor to a recipient comprising the steps of:

(a) receiving, at a recipient's location, a plurality of elements including:

(i) software;

(ii) a cryptographically secured representation of said software, said cryptographically secured representation having been secured by a first encryption key;

(iii) an identifier of said distributor; and

(iv) a cryptographic certification, by a certifier, of a first decryption key corresponding to said first encryption key;

said received elements defining a software passport including at least elements (ii), (iii) and (iv); and

(b) cryptographically verifying said first decryption key using a second, preexisting decryption key unrelated to said distributor and obtained by said recipient without specific knowledge of said certifier; and

(c) cryptographically verifying said software using said verified first decryption key.

24. The method of claim 23 where said software passport includes said first decryption key.

25. The method of claim 24 where said software includes a binary representation of a computer program.

26. The method of claim 25 where said software passport includes a validity date of said computer program, and where said step of verifying said software includes checking said validity date.

27. The method of claim 25 where said step of verifying said software includes checking for the presence of said cryptographically secured representation.

28. The method of claim 27 where said step of verifying said software includes:

(a) decrypting said cryptographically secured representation using said first decryption key to yield a first message digest of at least a portion of said software;

(b) computing a second message digest on said at least a portion of said received software; and

(c) comparing said first and second message digests.

29. The method of claim 25 where said step of verifying said software includes checking for the presence of said cryptographic certification.

30. The method of claim 29 where said step of verifying said software includes:

(a) decrypting said cryptographic certification using said second decryption key to yield a first message digest of said first decryption key;

(b) computing a second message digest on said received first decryption key; and comparing said first and second message digests.

31. The method of claim 25 where said binary representation of said computer program has been encrypted.

32. The method of claim 31 where said computer program is subject to intellectual property protection.

33. The method of claim 32 where said computer program is subject to an access fee.

34. The method of claim 24 where said identifier is included in said cryptographic certification.

35. The method of claim 24 where said identifier includes information about said software.

36. The method of claim 24 where said software passport is received over a network.

37. The method of claim 24 where said cryptographic certification includes a validity date thereof.

38. The method of claim 24 where said second decryption key is stored at a computing platform of said recipient, and where said certifier is a provider of at least a portion of said computing platform.

39. The method of claim 38 where said second decryption key is a public key of said platform provider and where said first decryption key and said first encryption key are a public-private cryptographic key pair.

40. The method of claim 23 where said cryptographic certification represents an assurance of a skill of said distributor by said certifier.

41. The method of claim 40 where said step of verifying said software includes checking for the presence of said cryptographically secured representation.

42. The method of claim 41 where said step of verifying said software includes:

(a) decrypting said cryptographically secured representation using said first decryption key to yield a first message digest of at least a portion of said software;

(b) computing a second message digest on said at least a portion of said received software; and

(c) comparing said first and second message digests.

43. The method of claim 40 where said step of verifying said software includes checking for the presence of said cryptographic certification.

44. The method of claim 43 where said step of verifying said software includes:

(a) decrypting said cryptographic certification using said second decryption key to yield a first message digest of said first decryption key;

(b) computing a second message digest on said received first decryption key; and

(c) comparing said first and second message digests.

45. The method of claim 40 where said binary representation of said computer program has been encrypted.

46. A method for licensing of a software distributor by a certifier, comprising the steps of:

(a) receiving, at a certifier's location, an identifier of said distributor;

(b) verifying a qualification of said distributor against a predetermined licensing criterion; and

(c) performing a first cryptographic operation on said identifier to produce a cryptographic certification of said distributor;

(d) said cryptographic certification enabling cryptographic verification by a recipient thereof using a preexisting decryption unrelated to said distributor, and obtained by said recipient without specific knowledge of said certifier.

47. The method of claim 46 where said identifier includes a public key associated with said distributor.

48. The method of claim 47 where said step of performing said first cryptographic operation includes:

(a) computing a message digest on said public key; and

(b) encrypting said message digest with an encryption key corresponding to said preexisting decryption key.

49. The method of claim 47 where: (a) said preexisting decryption key is pre-stored at a computing platform of a recipient of said certification and (b) said certifier is a provider of at least a portion of said platform.

Nintendo - Ancora Exh. 1003

50. A method for secure software distribution from a distributor to a recipient, comprising the steps of:

(a) identifying software that is to be distributed to a recipient;

(b) using a first encryption key to perform a first cryptographic operation on said software to form a cryptographically secured representation of said software;

(c) obtaining, from a certifier, a cryptographic certification of a first decryption key

corresponding to said first encryption key; and

(d) generating a software passport for said recipient, said software passport including at least said cryptographically secured representation, said cryptographic certification, and an identifier of said distributor; where said software passport enables said recipient thereof (i) to cryptographically verify said first decryption key using a second, preexisting decryption key unrelated to said distributor and obtained without specific knowledge of said certifier, and (ii) to cryptographically verify said software using said verified first decryption key.

51. The method of claim 50 where said software passport includes said first decryption key.

52. The method of claim 51 where said software includes a binary representation of a compute program.

53. The method of claim 52 where said software passport further includes a validity date of said computer program.

54. The method of claim 51 where said first decryption key and said first encryption key are a public-private cryptographic key pair.

55. The method of claim 54 where said step of using a first encryption key to perform a first cryptographic operation includes:

(a) computing a message digest of at least a portion of said software; and

(b) encrypting said message digest with said first cryptographic key.

56. The method of claim 51 where said cryptographic certification is secured by a second encryption key corresponding to said second decryption key, and where said second keys are a private-public cryptographic key pair.

57. The method of claim 56 where said cryptographic certification includes a message digest of said first decryption key, said message digest having been encrypted with a second encryption key.

58. The method of claim 51 where said software has been encrypted.

59. The method of claim 58 where said software is subject to intellectual property protection.

60. The method of claim 58 where said software is subject to an access fee.

61. The method of claim 51 where said step of obtaining said cryptographic certification includes receiving said identifier.

62. The method of claim 51 where said identifier includes information about said software.

63. The method of claim 51 where said software passport is received over a network.

64. The method of claim 51 where said cryptographic certification includes a validity date thereof.

65. The method of claim 51 where said second decryption key is stored at a computing platform of said recipient, and where said certifier is a provider of at least a portion of said computing platform.

66. The method of claim 65 where said second decryption key is a public key of said platform provider and where said first decryption key and said first encryption key are a public-private cryptographic key pair.

67. The method of claim 50 where said cryptographic certification represents an assurance of a skill of said distributor by said certifier.

68. The method of claim 67 where said first decryption key and said first encryption key are a public-private cryptographic key pair.

69. The method of claim 68 where said step of using a first encryption key to perform a first cryptographic operation includes:

(a) computing a message digest on at least a portion of said software; and

(b) encrypting said message digest with said first encryption key.

70. The method of claim 67 where said cryptographic certification is secured by a second encryption key corresponding to said second decryption key, and where said second keys are a public-private cryptographic pair.

71. The method of claim 70 where said cryptographic certification includes a message digest of said first decryption key, said message digest having been encrypted with said second encryption key.

72. The method of claim 67 where said software has been encrypted.

* * * * *

APPENDIX B-8

US006138236A

United States Patent [19]

Mirov et al.

[11] Patent Number: 6,138,236

[45] Date of Patent: *Oct. 24, 2000

[54] **METHOD AND APPARATUS FOR FIRMWARE AUTHENTICATION**

[75] Inventors: **Russell Norman Mirov**, Los Altos; **Gregory Charles Onufer**, Sunnyvale, both of Calif.

[73] Assignee: **Sun Microsystems, Inc.**, Palo Alto, Calif.

[*] Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

[21] Appl. No.: **08/674,026**

[22] Filed: **Jul. 1, 1996**

[51] **Int. Cl.**[7] **G06F 11/30**; H04L 9/00

[52] **U.S. Cl.** **713/200**; 713/202

[58] **Field of Search** 713/200, 201, 713/202, 155, 161, 180; 380/282, 281

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,448,045	9/1995	Clark	235/382
5,481,612	1/1996	Campana et al.	380/25
5,535,409	7/1996	Larvoire et al.	395/800
5,537,540	7/1996	Miller et al.	395/183.14
5,586,327	12/1996	Bealkowski et al.	395/652
5,621,796	4/1997	Davis et al.	380/24
5,633,930	5/1997	Davis et al.	380/24
5,643,086	7/1997	Alcorn et al.	380/25 X
5,768,382	6/1998	Schneier et al.	380/23

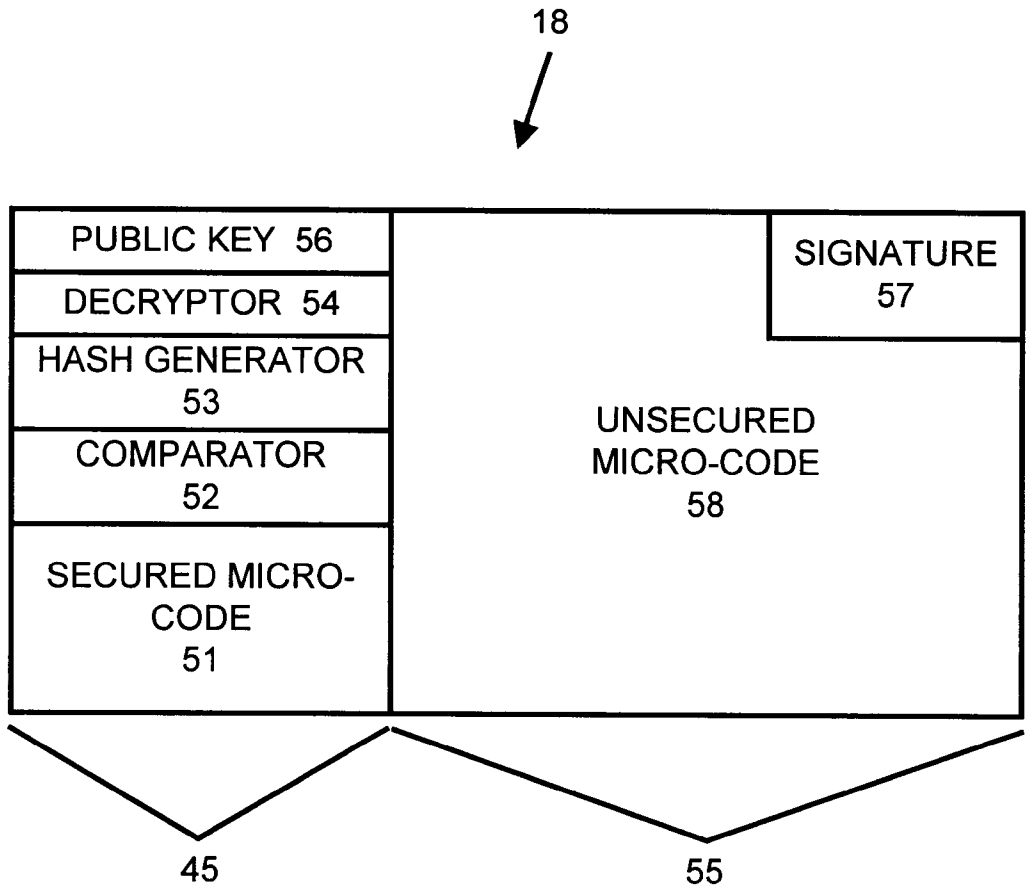
Primary Examiner—Dieu-Minh T. Le

Attorney, Agent, or Firm—Park & Vaughan LLP

[57] **ABSTRACT**

An apparatus for firmware authentication and methods of operating the same result in software upgradability to firmware without compromising the integrity of the firmware. The apparatus for firmware authentication of a boot PROM comprises a software programmable data section having a plurality of micro-code. An authentication section having a hash generator configured to generate a data hash in response to the plurality of micro-code programmed in the software programmable data section to authorize execution of the plurality of micro-code of the data section.

17 Claims, 4 Drawing Sheets



Nintendo - Ancora Exh. 1003

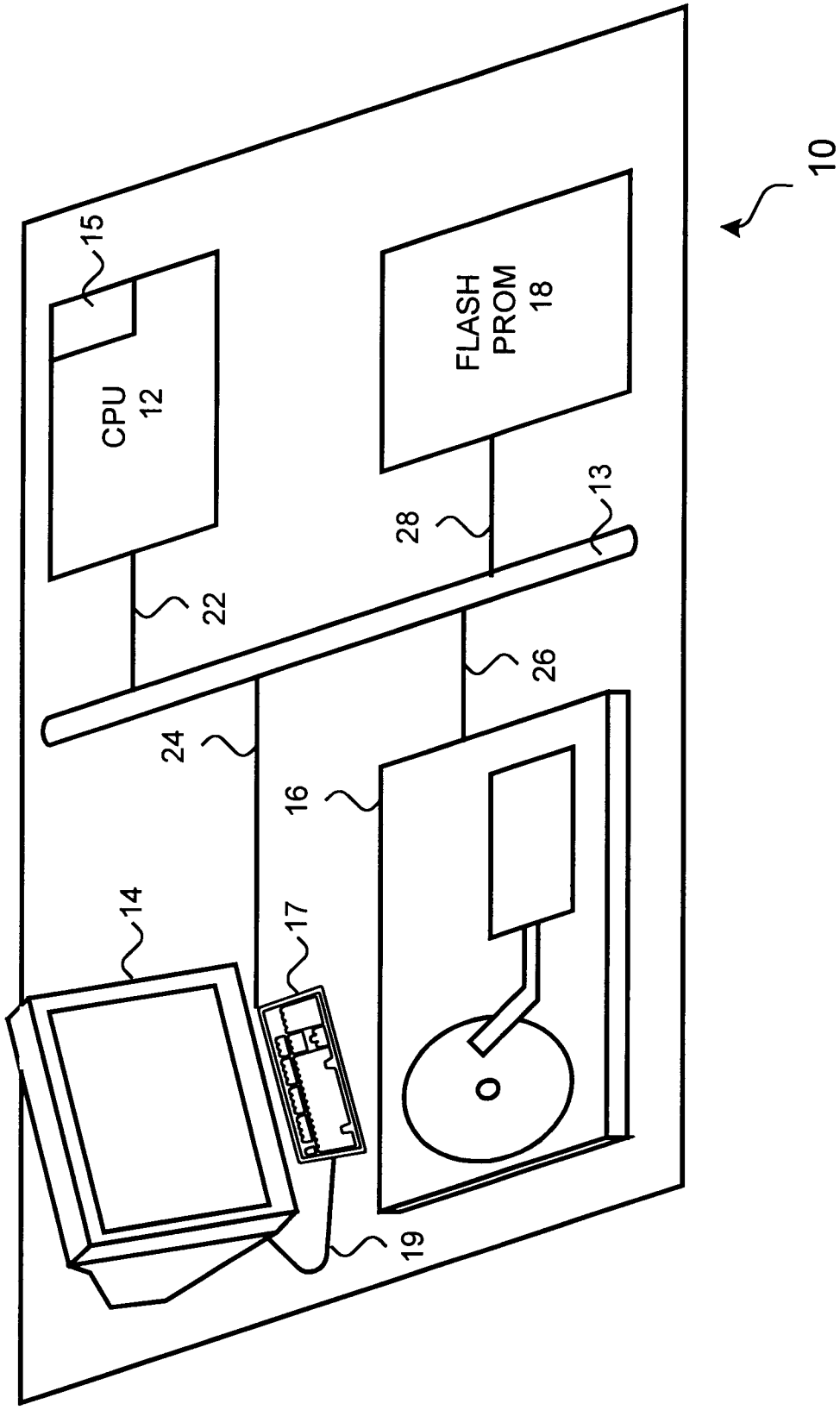


FIG. 1

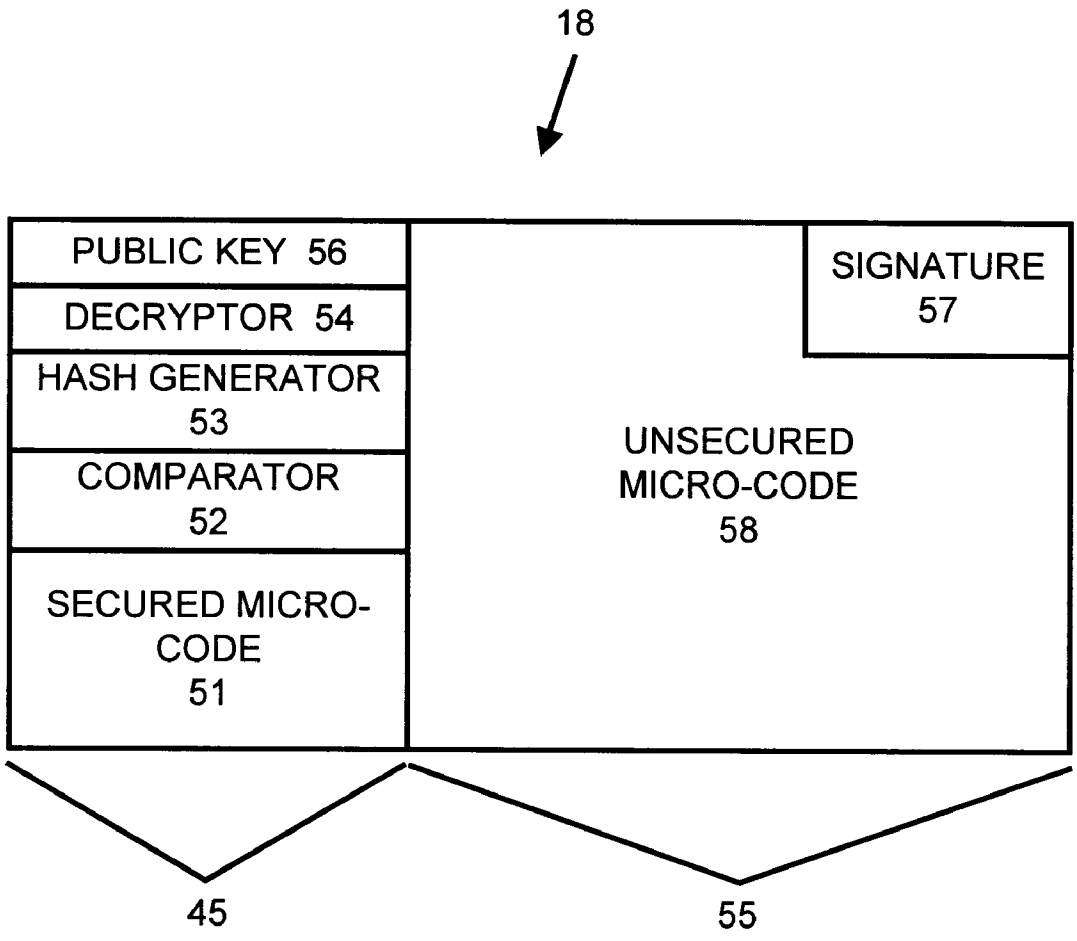

18
PUBLIC KEY 56
DECRYPTOR 54
HASH GENERATOR 53
COMPARATOR 52
SECURED MICRO-CODE 51
SIGNATURE 57
UNSECURED MICRO-CODE 58
45
55

FIG. 2

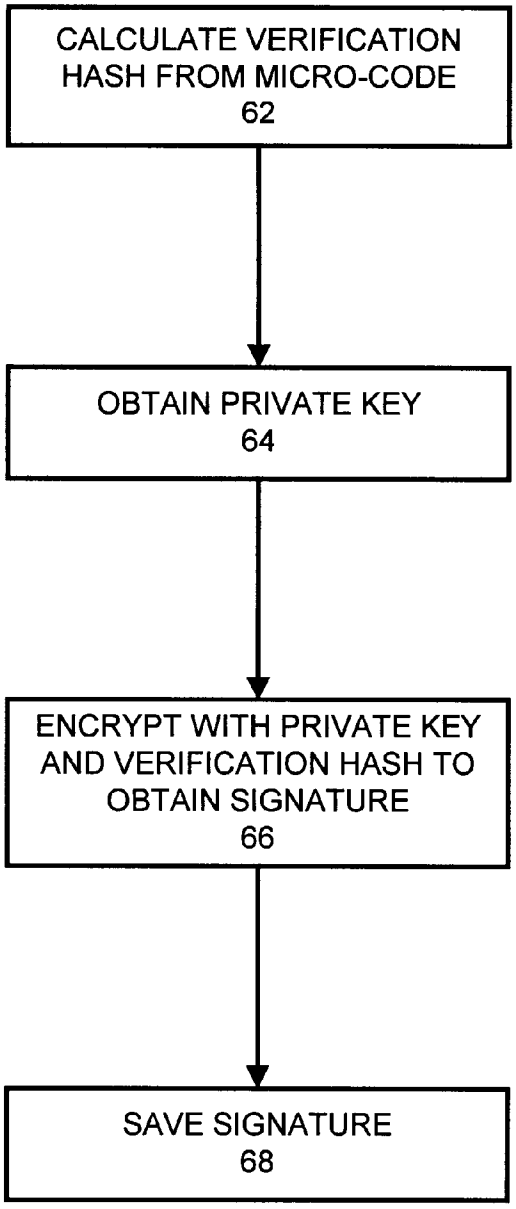

CALCULATE VERIFICATION HASH FROM MICRO-CODE
62
OBTAIN PRIVATE KEY
64
ENCRYPT WITH PRIVATE KEY AND VERIFICATION HASH TO OBTAIN SIGNATURE
66
SAVE SIGNATURE
68

FIG. 3

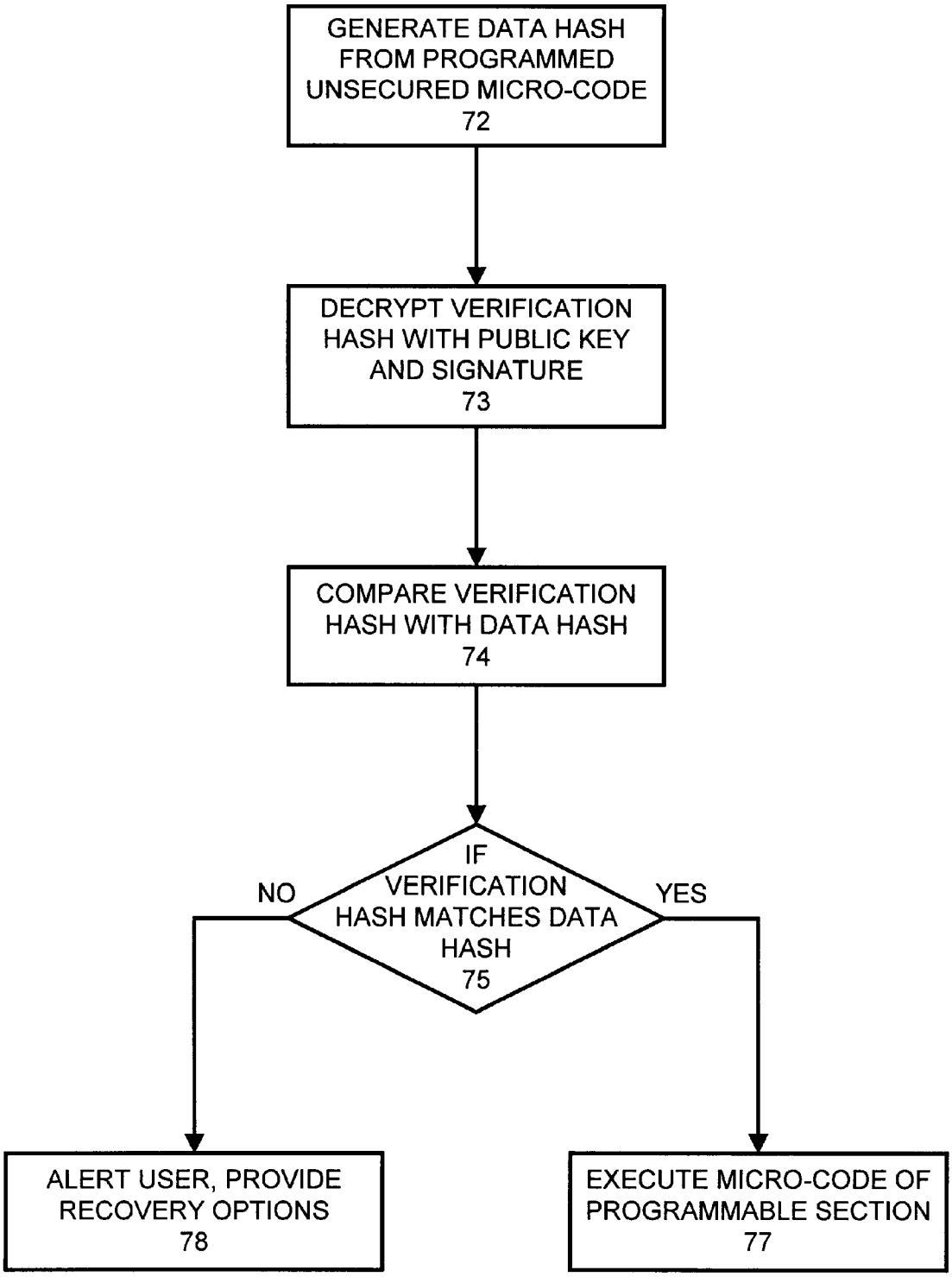

GENERATE DATA HASH FROM PROGRAMMED UNSECURED MICRO-CODE 72
DECRYPT VERIFICATION HASH WITH PUBLIC KEY AND SIGNATURE 73
COMPARE VERIFICATION HASH WITH DATA HASH 74
IF VERIFICATION HASH MATCHES DATA HASH 75
NO
YES
ALERT USER, PROVIDE RECOVERY OPTIONS 78
EXECUTE MICRO-CODE OF PROGRAMMABLE SECTION 77

FIG. 4

METHOD AND APPARATUS FOR FIRMWARE AUTHENTICATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to authentication of programmed micro-code and more particularly to confirm the integrity of programmable micro-code written in a memory device.

2. Description of the Related Arts

Computer systems during initial power up rely on a sequence of instructional routines which build on each previously executed instructional routine until the computer system is initialized. Micro-code, also referred to as firmware or boot code, is the first level of the instructional routines that are executed when the computer system is initially powered up. The micro-code stored in non-volatile memory devices such as a memory IC (integrated circuit) directs the computer system to certain boot blocks located on a disk drive. As these boot blocks on the disk drive are executed, successively larger blocks of boot data are loaded until finally the operating system, such as an Unix or Microsoft Windows of the computer system is loaded.

The micro-code for the initial boot up instructions of a computer system is typically stored in a boot ROM (read only memory) or boot PROM (programmable read only memory). An example of a PROM is a flash PROM, often referred to as flash memory. Needs arise when the micro-code for the initial boot up instructions requires updating. Those computer systems having ROMs require new ROMs. Replacing old ROMs with newly supplied ROMs is expensive. Furthermore, the computer system has to be disassembled to gain access to replace the ROMs.

In computer systems with boot PROMs that employ flash technology, updating new micro-code entails accessing the flash PROM using software and programming the flash PROM with new micro-code. However, because the micro-code contained in the boot PROM is the first code that is executed, reasons to limit programming access to the flash PROM include: 1) inadvertent programming can cause the computer system become completely inactive; 2) security sensitive environments require that the micro-code be tamper-proof to prevent security risks. Thus, safeguards are currently in place to prevent modification of the boot PROM.

These safeguards include using boot ROMs to store the micro-code or setting hardwire jumpers that prevent software modification of boot PROMs. In order to modify the micro-code, boot ROMs must be replaced with new boot ROMs containing the updated micro-code. In the case of boot PROMs, user intervention is required to manually switch the jumpers of the boot PROMs to enable programming access to the boot PROMs for the new micro-code. In either case, user intervention is required to physically open the computer system and make the necessary changes. The changes range from the replacement of old boot ROMs with new boot ROMs to changing jumper settings of the flash boot PROM to enable and disable programming of the flash boot PROM. Thus, the safeguards require additional time and effort from the users to implement modifications to the micro-code. The process of providing upgrades to the micro-code programming is cumbersome and time-consuming.

Therefore, it is desirable to provide an apparatus for authenticating firmware programmed in a boot PROM and methods of operating the same that enable programming

access to the boot PROM without compromising the authenticity of the firmware that overcome the disadvantages of disassembling the computer system.

SUMMARY OF THE INVENTION

The present invention provides an apparatus for firmware authentication and methods for operating the same which result in software upgradability to firmware without compromising the integrity of the firmware. The novel application for authentication of firmware is based on cryptography. Thus, according to one aspect of the invention, a boot PROM (programmable read only memory) having programming instructions for initiating a computer system is provided. A software programmable data section has a plurality of micro-code. An authentication section having a hash generator generates a data hash in response to the plurality of micro-code programmed in the software programmable data section to authorize execution of the plurality of micro-code of the data section.

According to another aspect of the invention, the software programmable data section includes a predetermined digital signature, and the authentication section includes a predetermined public key and a decryptor which provides an verification hash in response to the predetermined signature and the public key. The authentication section also includes a comparator which compares the data hash with the verification hash to authenticate the plurality of micro-code of the software programmable data section. If the data hash and the verification hash do not match, a message alerts the user of the mismatch indicating that the micro-code is not authenticated.

According to another aspect of the invention, the authentication section includes a plurality of trusted micro-code which initiates execution of the plurality of micro-code of the software programmable data section in response to proper authentication of the data hash. The proper authentication of the data hash by the authentication section of the plurality of trusted micro-code affords the plurality of micro-code programmed in the software programmable data section to a level of trusted code. Thus, the trusted code of the software programmable data section can be used to authenticate another set of downstream code that is executed during the boot up sequence for the computer system.

According to yet another aspect of the invention, the software programmable data section includes a flash memory which enables software reprogramming of the plurality of micro-code. Other programmable storage mediums are also usable for the storage of the micro-code. The authentication section includes a ROM (read only memory) that provides a base line for trusted code.

An apparatus and method for firmware authentication are provided by authenticating the software programmable data section of the boot PROM with a trusted ROM section of the boot PROM. The ability to provide software programmability of the boot PROM affords ease in upgradability that saves time, effort, and energy. Upgrading with newer versions of the boot PROM affords support for new functions and eliminates bugs and other inconsistencies that can plague older versions of the boot PROM. Thus, the newer boot PROMs provide for a smoother and more efficient operating computer system.

Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

Nintendo - Ancora Exh. 1003

BRIEF DESCRIPTION OF THE DRAWING

FIG. **1** illustrates a system level block diagram of a computer system;

FIG. **2** illustrates a block diagram of a flash PROM of the computer system in accordance with the present invention;

FIG. **3** illustrates a flow diagram for generating a signature in accordance with the present invention; and

FIG. **4** illustrates a flow diagram for authenticating unsecured microcode of the programmable section of the flash PROM.

DETAILED DESCRIPTION OF THE INVENTION

The invention will be described with respect to the Figures in which FIG. **1** generally shows a simplified computer system **10**. The computer system **10** includes a CPU (central processing unit) **12**, display **14**, hard disk **16** and a flash PROM (programmable read-only memory) **18**. The computer system **10** is for illustrative purposes as many variations to the architecture of the computer system **10** are available and known in the art. CPU bus **22** couples the CPU **12** to data bus **13**. The CPU **12** includes a memory **15** which stores instructions and data for processing by the CPU **12**. Disk drive bus **26** couples the disk drive **16** to the data bus **13**. The disk drive **16** provides non-volatile data storage for the computer system **10**. Data transfers occur between the CPU **12** and the disk drive **12** as the data is processed by CPU **12**. Display bus **24** couples the display **14** to the data bus **13**. The display **14** receives output data for display. The display **14** includes a keyboard **17** coupled to the display via cable **19**. The keyboard **17** provides an user interface to computer system **10**. PROM bus **28** couples the flash PROM **18** to data bus **13**. The flash PROM **18** includes initialization instructions for the computer system **10**.

During start-up of the computer system **10**, micro-code instructions stored in the flash PROM **18** are executed. The micro-code instructions include boot code that directs execution of particular boot blocks of the hard disk **16**. Once the instructions contained in the boot blocks of the hard disk **16** are executed and loaded into the memory **15**, higher level instructions and code are executed and loaded into memory **15** such as operating systems for Windows 95, Unix, or Macintosh based computers. The higher level instructions and code may be executed from a network server. Thus, in an alternative embodiment, computer system **10** is one of a number of computer systems coupled to a network.

In a network, the computer system **10** may not include the disk drive **16**, as data transfers are through a network server. The network server includes wired network connections, RF (radio frequency) network connections, and IR (infrared) network connections. Other computer systems include hand held systems such as PDAs (Personal Data Assistants) and computer systems that include micro-code to initialize the computer system.

FIG. **2** illustrates a block diagram of the flash PROM **18**. The flash PROM **18** is divided into two main sections: a authentication section **45** and a programmable section **55**. The authentication section **45** is a ROM (read-only memory). The micro-code instructions contained in the authentication section **45** are read-only. The micro-code instructions contained in the programmable section **55** are re-writable. For example, the programmable section **55** includes a flash memory that is software programmable with new micro-code.

The authentication section **45** authenticates the programmable section **55** to verify that the micro-code instructions which boot the computer system **10** are trusted because the programmable section **55** is software programmable. The authentication section **45** includes a plurality of secure micro-code **51**, a comparator **52**, a hash generator **53**, a decryptor **54** and a public key **56**. The unsecured section **55** includes a digital signature **57** and a plurality of unsecured micro-code **58**.

During initialization of the computer system **10**, the secure micro-code **51** of the authentication section **45** executes and directs the hash generator **53** to generate a data hash of the unsecured micro-code **58** programmed in the programmable section **55** of the flash PROM **18**. The secure micro-code **51** also directs the decryptor **54** to calculate a verification hash. The decryptor applies the public key **56** of the authentication section **45** and the digital signature **57** of the programmable section **55** and calculates the verification hash.

Once the verification hash and the data hash are generated, the micro-code **51** directs the comparator **52** to compare the verification hash with the data hash. If the verification hash matches the data hash, the unsecured micro-code **55** is properly verified and permitted to execute. If the comparison of the verification hash and the data hash fails, the unsecured micro-code **58** is corrupted or had been altered without proper authorization.

Public-key cryptography verifies that the digital signature **57** and the public key **56** decrypts to a verification hash which matches the data hash of the micro-code programmed in the programmable section **55** of the flash PROM **18**. The data hash generator **53** generates the data hash. A digital signature **57** of the programmable section **55** is provided when the programmable section **55** is programmed. During authorized programming of the programmable section **55**, an initial hash from the authorized programming micro-code is generated. Next, a proper digital signature **57** is encrypted from a secret key and the initial hash of the authorized programming micro-code **58** using public key cryptography techniques. The proper digital signature **57** and the authorized programming micro-code **58** are written to the programmable section **55**.

The authentication section **45** of the flash PROM **18** is initially programmed with the secure micro-code **51**, the comparator **52**, the hash generator **53**, the decryptor **54**, and the public key **56**. Whenever the computer system **10** is initialized, the authentication section **45** verifies that the data hash of the unsecured micro-code **58** matches the verification hash to ensure the integrity of the unsecured micro-code **58** and authenticate that the unsecured micro-code **58** had not been altered. As the unsecured micro-code **58** of the programmable section **55** is authenticated, the trust level of the unsecured micro-code **58** is raised to a level of trusted. Thus, the authenticated micro-code **58** can be used to authenticate other initialization code down stream in the start-up sequence of the computer system **10**.

FIG. **3** shows a flow diagram for generating a digital signature **57** for the micro-code **58**. The diagram begins with generation of the verification hash from the micro-code **58** in step **62**. Next, the private key is obtained for the generation of a verification hash from the micro-code **58** in step **64**. In step **66**, the verification hash is encrypted using public key cryptography techniques and the private key to obtain the digital signature **57**. Finally, in step **68**, the digital signature **57** is programmed with the micro-code **58** to the programmable section **55** of the flash PROM **18**.

FIG. **4** shows a flow diagram for authenticating the unsecured micro-code **58** of the programmable section **55**.

Nintendo - Ancora Exh. 1003

The diagram begins with generation of the data hash from the unsecured micro-code **58** contained in the programmable section **55** in step **72**. In step **73**, the verification hash is decrypted with the public key **56** contained in the authentication section **45** and the digital signature **57** contained in the programmable section **55**. Step **74** provides a comparison of the verification hash with the data hash. In decision step **75**, if the verification hash matches the data hash then step **77** authorizes the execution of the micro-code **58** contained in the programmable section **55**. If in decision step **75**, the verification hash does not match the data hash; step **78** provides a message to the user that an error occurred during authentication of the programmable section **55** and offers a recovery solution for the user to obtain valid micro-code.

A flash PROM **18** having an authentication section **45** and a programmable section **55** affords ease in updating the flash PROM **18** with new micro-code without compromising security. Implementing public-key cryptography having a private key and a public key to verify the programmable section **55** with the authentication section **45** assures that the programmable section of the micro-code is proper and authentic. The integrity of the unsecured micro-code **58** of the programmable section **55** is also verified when the verification hash matches the data hash. As the trust level of the unsecured micro-code **58** is raised to a level of trusted, other boot data such as the boot blocks of the disk drive **16** used for initializing the computer system **10** can be similarly authenticated using the now trusted micro-code **58** of the programmable section **55**. Thus, a propagation of a series of security checks during the boot-up sequence can be implemented to ensure that each sequence executes properly authenticated boot code

While the foregoing detailed description has described several embodiments of the apparatus and methods of firmware authentication in accordance with this invention, it is to be understood that the above description is illustrative only and not limiting of the disclosed invention. Obviously, many modifications and variations will be apparent to the practitioners skilled in this art. Accordingly, the apparatus and methods of firmware authentication has been provided which authenticates the programmable section of a flash PROM with a read-only section of the flash PROM by application of public-key cryptography. By affording a programmable section of the flash PROM to be software programmable, updates to the firmware are accomplished without compromising the integrity of the firmware. No longer are system operators required to disassemble computer systems to perform updates to system start-up firmware.

What is claimed is:

1. A boot PROM (programmable read only memory) having programming instructions for initializing a computer system containing the boot PROM, the boot PROM comprising:

a software programmable data section having a plurality of micro-code configured to initialize the computer system; and

an authentication section having a hash generator configured to generate a data hash of the plurality of micro-code programmed in the software programmable data section, wherein the authentication section authenticates the plurality of micro-code with the data hash to authorize execution of the plurality of micro-code.

2. The boot PROM according to claim **1**, wherein:

the software programmable data section includes a predetermined signature; and

the authentication section includes a predetermined public key and a decryptor configured to provide a verification hash from the predetermined signature and the public key.

3. The boot PROM according to claim **2**, wherein the authentication section includes a comparator configured to compare the data hash with the verification hash to authenticate the plurality of micro-code of the software programmable data section.

4. The boot PROM according to claim **2**, wherein the predetermined signature includes an encryption of a private key and an initial hash of a plurality of initial micro-code programmed to the software programmable data section.

5. The boot PROM according to claim **1**, wherein the authentication section includes a plurality of trusted micro-code configured to initiate execution of the plurality of micro-code of the software programmable data section in response to proper authentication of the plurality of micro-code.

6. The boot PROM according to claim **5**, wherein the proper authentication of the micro-code programmed in the software programmable data section by the authentication section raises the plurality of micro-code to a level of trusted code.

7. The boot PROM according to claim **1**, wherein the software programmable data section includes a flash memory configured to enable software reprogramming of the plurality of micro-code.

8. The boot PROM according to claim **1**, wherein the authentication section includes a ROM (read only memory).

9. A method of operating a boot PROM (programmable read only memory) having programming instructions for initializing a computer system comprising the steps:

generating a data hash from a plurality of micro-code programmed in a software programmable data section of the boot PROM; and

authenticating the plurality of micro-code of the software programmable data section in an authentication section of the boot PROM to authorize execution of the plurality of micro-code to initialize the computer system.

10. The method of operating a boot PROM according to claim **9**, wherein:

the software programmable data section includes a predetermined signatures; and

the step of authenticating includes generating a verification hash from the predetermined signature and a public key stored in the authentication section of the boot PROM.

11. The method of operating a boot PROM according to claim **10**, wherein the step of authenticating includes comparing the data hash with the verification hash to authenticate the plurality of micro-code of the software programmable data section.

12. The method of operating a boot PROM according to claim **10** further comprising the step of encrypting with a private key an initial hash of a plurality of initial micro-code programmed to the software programmable data section to provide the predetermined signature.

13. The method of operating a boot PROM according to claim **9**, wherein the authentication section includes a plurality of trusted micro-code, further comprising the step of:

propagating a level of trust to the plurality of micro-code of the software programmable data section in response to proper authentication of the plurality of micro-code.

14. The method of operating a boot PROM according to claim **9**, wherein the software programmable data section includes a flash memory, further comprising the step of:

Nintendo - Ancora Exh. 1003

reprogramming the plurality of micro-code in the software programmable data section.

15. The method of operating a boot PROM according to claim **9** wherein the authentication section includes a ROM (read only memory).

16. A memory module for initializing a computer system, comprising:

a programmable section configured to store:

a set of initialization instructions which, when executed by a processor, initialize a computer system in which the memory module and processor are installed; and

a digital signature of said set of initialization instructions, wherein said digital signature is generated by encrypting a verification hash of said set of initialization instructions with a private encryption/decryption key; and

a read-only section configured to store:

a public encryption/decryption key corresponding to the private encryption/decryption key;

a decryptor configured to decrypt said digital signature with said public encryption/decryption key to reproduce said verification hash;

a hash generator configured to generate a test hash from said set of initialization instructions;

a comparator configured to compare said test hash and said verification hash; and

a set of trusted micro-code configured to execute before the computer system is operable by a user;

wherein, upon execution of said trusted micro-code, said initialization instructions are executed if said test hash matches said verification hash.

17. A method of initializing a computer system with authenticatable initialization instructions, comprising:

executing a set of trusted micro-code stored in a read-only portion of a memory module installed in the computer system before the computer system is operable by a user;

generating a test hash from a set of initialization instructions stored in a programmable portion of said memory module;

retrieving a digital signature from said reprogrammable portion of said memory module;

decrypting said digital signature with a public key stored in said read-only portion of said memory module to retrieve a verification hash of an initial set of initialization instructions stored in said programmable portion of said memory module;

comparing said test hash and said verification hash; and

if said test hash and said verification hash match, executing said set of initialization instructions to initialize the computer system.

* * * * *

Nintendo - Ancora Exh. 1003

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO : 6,138,236

DATED : October 24, 2000

INVENTOR(S): Mirov et al.

It is certified that errors appear in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In claim 10 (at column 6, line 44), replace "signatures" with --signature--.

Signed and Sealed this

Eighth Day of May, 2001

Attest:

Nicholas P. Godici

NICHOLAS P. GODICI

Attesting Officer

Acting Director of the United States Patent and Trademark Office

APPENDIX B-9

US005802592A

United States Patent [19]

Chess et al.

[11] Patent Number: 5,802,592

[45] Date of Patent: Sep. 1, 1998

[54] **SYSTEM AND METHOD FOR PROTECTING INTEGRITY OF ALTERABLE ROM USING DIGITAL SIGNATURES**

[75] Inventors: **David M. Chess**, Mohegan Lake; **Gregory Bret Sorkin**; **Steve Richard White**, both of New York, all of N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **656,626**

[22] Filed: **May 31, 1996**

[51] **Int. Cl.**[6] **G06F 12/14**; G06F 12/16; G06F 11/30

[52] **U.S. Cl.** **711/164**; 711/102; 711/103; 395/183.12; 395/183.14; 395/183.21; 395/652; 395/633

[58] **Field of Search** 711/102, 163, 711/103, 164; 395/651–653, 186, 188.01, 183.09, 183.12, 183.14, 183.21

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,327,531	7/1994	Bealkowski et al.	395/182.04
5,379,342	1/1995	Arnold et al.	380/2
5,396,558	3/1995	Ishiguro et al.	380/25
5,522,076	5/1996	Dewa et al.	395/652
5,579,522	11/1996	Christeson et al.	395/652
5,634,079	5/1997	Buxton	395/892

FOREIGN PATENT DOCUMENTS

0 515 760 A1	12/1992	European Pat. Off. .
0 588 339 A2	3/1994	European Pat. Off. .

OTHER PUBLICATIONS

Aarons et al., Security strategies: hardware protection for PCs, PC Magazine, v6, p. 104(12), Apr. 28, 1987.

Rosch, Internal Security: The Growing Mass of Stored PC Data Makes Protecting It a Modern Necessity, PC Week, v2, n18, pp. 89–91, May 7, 1985.

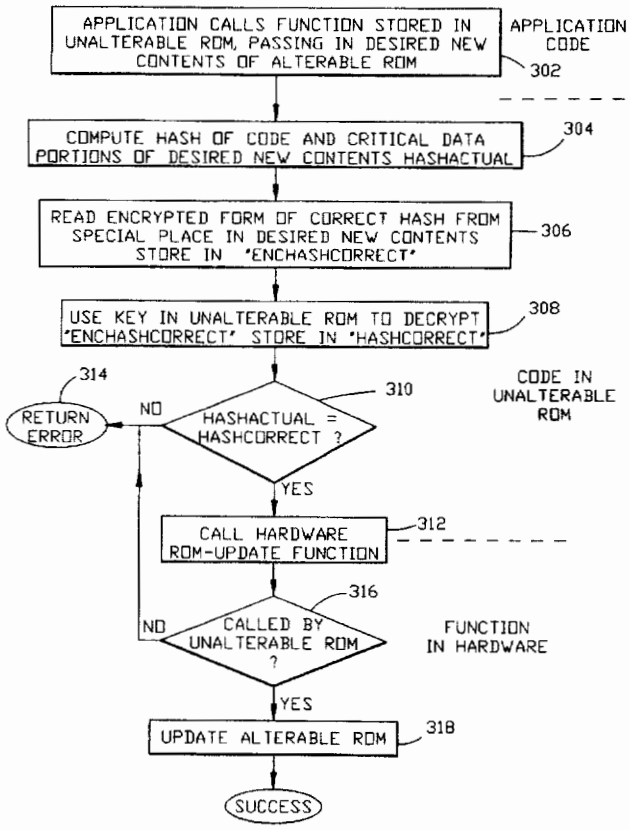
Clark et al., BITS: A smartcard protected operating system, Communications of the ACM, v37, n11, pp. 66–70, Nov. 1994.

Primary Examiner—Eddie P. Chan
Assistant Examiner—Reginald G. Bragdon
Attorney, Agent, or Firm—Perman & Green, LLP

[57] **ABSTRACT**

A system and method for verifying the integrity of a computer system's BIOS programs stored in alterable read only memory (such as FLASH ROM), and preventing malicious alteration thereof. The system and method regularly check the contents of the alterable read only memory using a digital signature encrypted by means of an asymmetrical key cryptosystem.

28 Claims, 3 Drawing Sheets



Nintendo - Ancora Exh. 1003

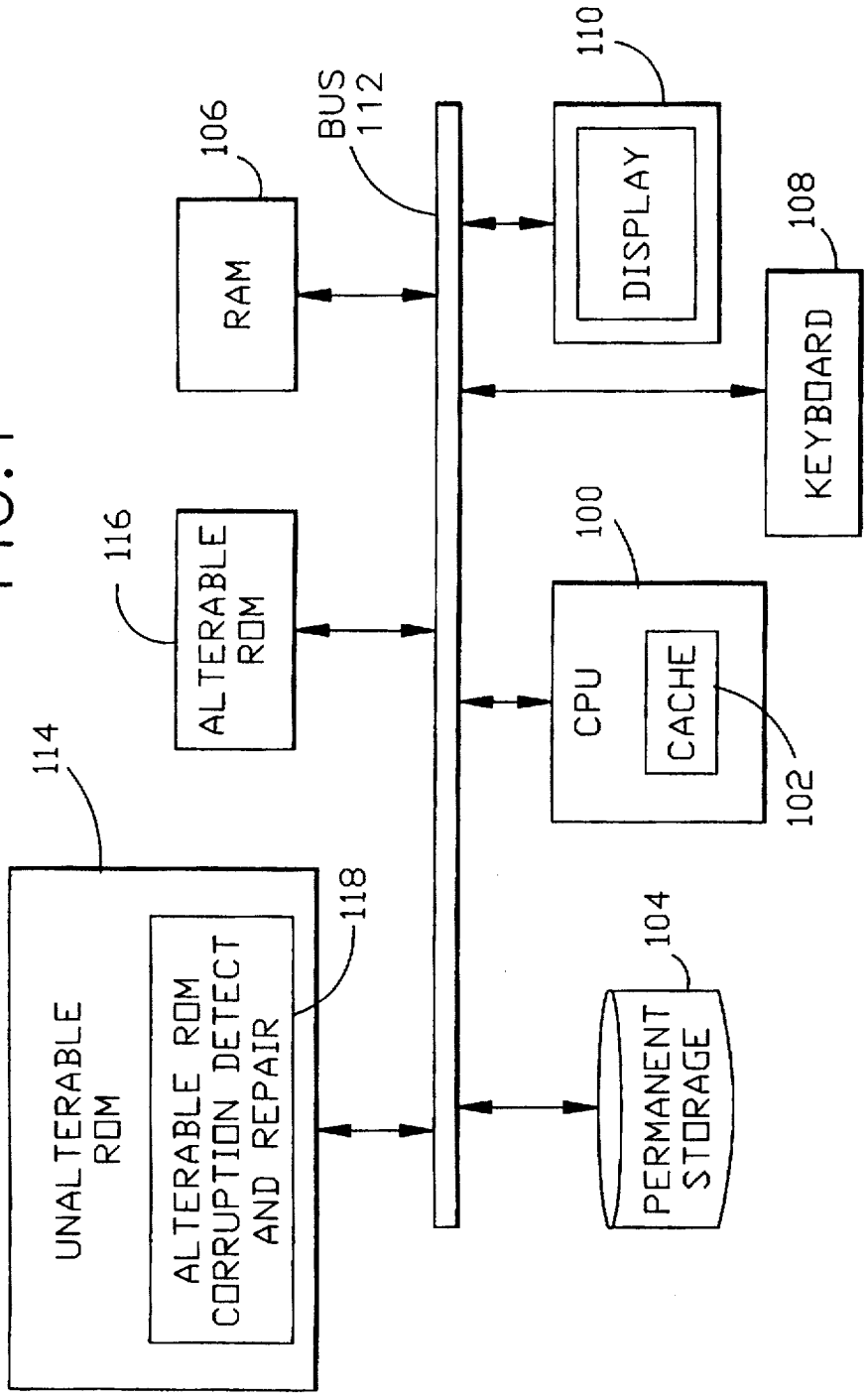

UNALTERABLE ROM
ALTERABLE ROM CORRUPTION DETECT AND REPAIR
114
118
ALTERABLE ROM
116
RAM
106
BUS 112
CPU
CACHE
100
102
PERMANENT STORAGE
104
DISPLAY
110
KEYBOARD
108

FIG. 1

FIG.2

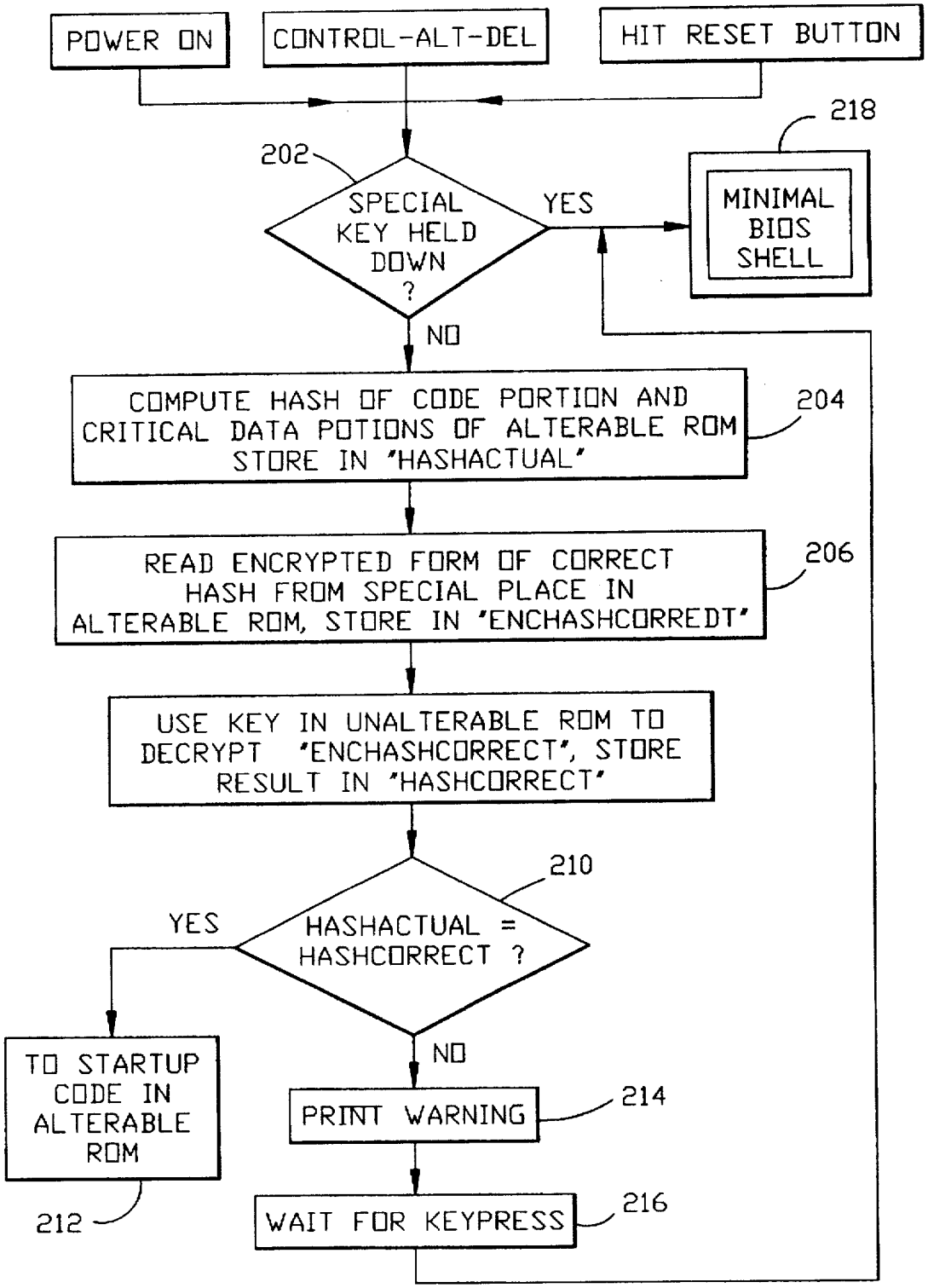
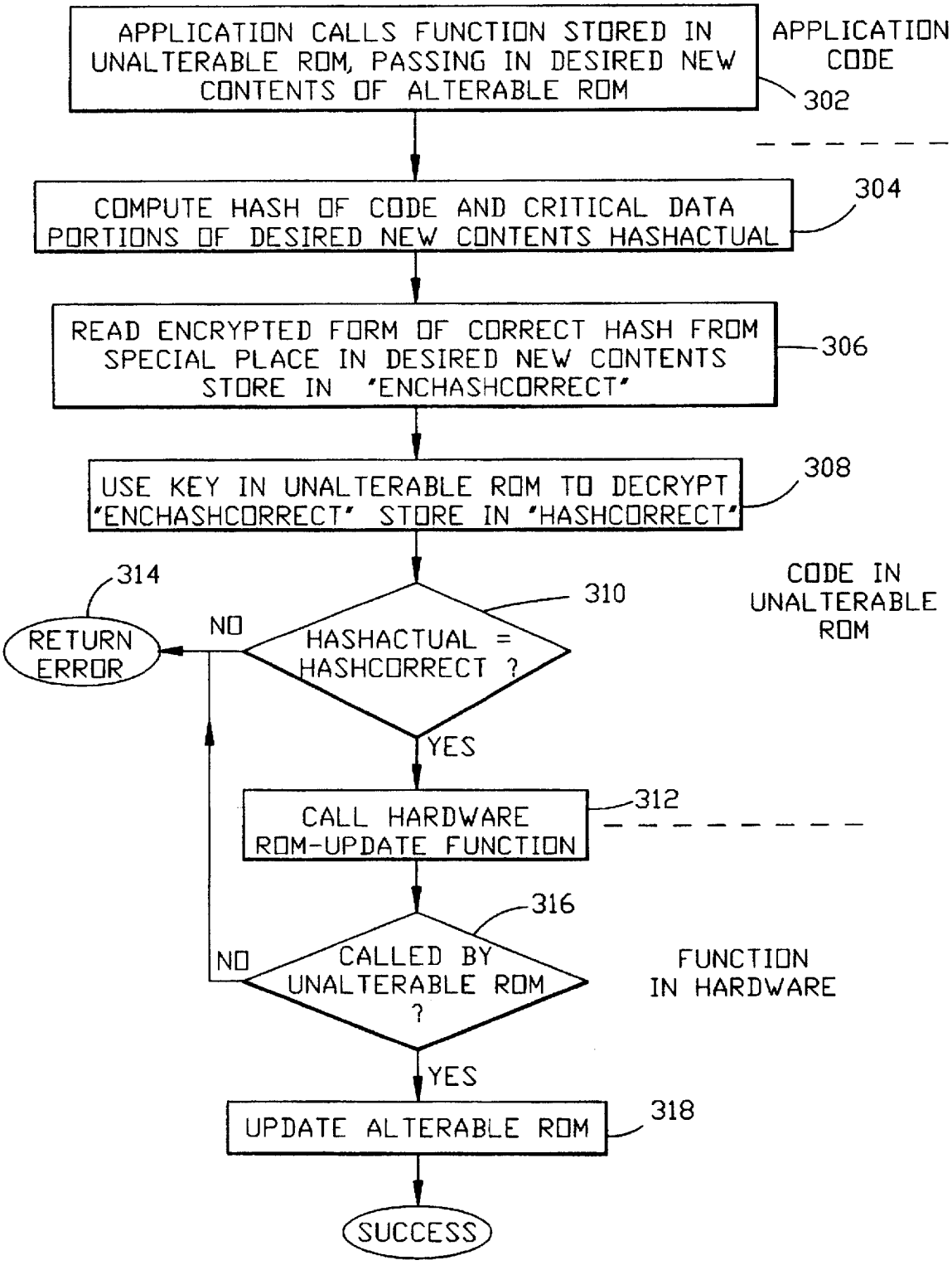

POWER ON
CONTROL-ALT-DEL
HIT RESET BUTTON
202
SPECIAL KEY HELD DOWN ?
YES
218
MINIMAL BIOS SHELL
NO
COMPUTE HASH OF CODE PORTION AND CRITICAL DATA POTIONS OF ALTERABLE ROM STORE IN "HASHACTUAL"
204
READ ENCRYPTED FORM OF CORRECT HASH FROM SPECIAL PLACE IN ALTERABLE ROM, STORE IN "ENCHASHCORREDT"
206
USE KEY IN UNALTERABLE ROM TO DECRYPT "ENCHASHCORRECT", STORE RESULT IN "HASHCORRECT"
210
HASHACTUAL = HASHCORRECT ?
YES
TO STARTUP CODE IN ALTERABLE ROM
212
NO
PRINT WARNING
214
WAIT FOR KEYPRESS
216

FIG.3


APPLICATION CALLS FUNCTION STORED IN UNALTERABLE ROM, PASSING IN DESIRED NEW CONTENTS OF ALTERABLE ROM
APPLICATION CODE
302
COMPUTE HASH OF CODE AND CRITICAL DATA PORTIONS OF DESIRED NEW CONTENTS HASHACTUAL
304
READ ENCRYPTED FORM OF CORRECT HASH FROM SPECIAL PLACE IN DESIRED NEW CONTENTS STORE IN "ENCHASHCORRECT"
306
USE KEY IN UNALTERABLE ROM TO DECRYPT "ENCHASHCORRECT" STORE IN "HASHCORRECT"
308
314
310
CODE IN UNALTERABLE ROM
RETURN ERROR
NO
HASHACTUAL = HASHCORRECT ?
YES
CALL HARDWARE ROM-UPDATE FUNCTION
312
316
NO
CALLED BY UNALTERABLE ROM ?
FUNCTION IN HARDWARE
YES
UPDATE ALTERABLE ROM
318
SUCCESS

SYSTEM AND METHOD FOR PROTECTING INTEGRITY OF ALTERABLE ROM USING DIGITAL SIGNATURES

FIELD OF THE INVENTION

The invention relates to the protection of the integrity of computer system basic input-output systems.

BACKGROUND OF THE INVENTION

Modern general-purpose computers contain programs stored in non-volatile read-only memory (ROM) which are used to "bootstrap" the system when power is turned on, and to provide basic low-level access to the hardware. These programs generally perform various tests for proper functioning of the system hardware at power-on and then locate, load and transfer control to the operating system bootstrap code. They also provide a standard interface (sometimes called the basic Input/Output System, or BIOS) to the functions of the hardware.

While such system ROMs were originally of the permanently "burned-in" variety, which can be changed only by physically replacing a microchip, advances in technology have recently made it possible to utilize alterable, or "FLASH" ROM instead. The advantage of alterable ROM is that its contents can be altered by software, making ROM updates significantly simpler. As alterable ROM technology advances, and as systems become more complex, requiring more frequent ROM updates, the use of FLASH for this purpose is quickly becoming more common.

While software-alterable ROM has definite advantages, it also has dangers; since the ROM is the basic software that controls the startup and low-level operation of the system, if it becomes corrupted (accidently or maliciously), the integrity of the system as a whole can be compromised, and it can be very difficult either to detect the corruption or to repair it.

There are well-known methods of verifying the integrity of the contents of ROMs (FLASH and otherwise) by performing a simple checksum, to ensure that, to a very high probability, no accidental changes have been made to the contents of the ROM. The techniques used to do this verification are typically a simple additive checksum or a cyclic redundancy check: these techniques are designed to be simple and fast, while having a high probability of detecting typical accidental or defect-caused changes to ROM. They are, however, easily "invertible" ; that is, given the current contents of ROM and the current value of the checksum, an attacker desiring to make intentional changes to the ROM without modifying the checksum would be able to do so with little difficulty.

A further feature of many current systems is that they allow the user to access the built-in programs stored in ROM for examining and altering system configuration settings. This typically is accomplished by starting the system from a special diskette, or pressing a combination of keys during system setup. But the configuration programs, and the programs that decide whether or not to pass control to them, are themselves alterable ROM (on machines that have alterable ROM), and therefore could become corrupted.

SUMMARY OF THE INVENTION

The current invention functions in a component of a computing system containing alterable ROM to verify that the alterable ROM has not been changed, or that a proposed update to the alterable ROM is legitimate. This verification is performed by use of a digital signature, the signature

having the characteristic that it is not easily invertible: even an attacker with full knowledge of the code used to verify the digital signature, and with the ability to alter the current contents of the ROM and the current signature, would have to perform a prohibitive amount of computation to generate a new content/signature pair that would pass the test.

The manufacturer, on the other hand, by virtue of having access to a secret piece of data (for example, the private key in an asymmetrical-key cryptosystem), is able to produce signatures for new versions of the contents of alterable ROM very easily.

BRIEF DESCRIPTION OF THE DRAWING

FIG. **1** is a system in accordance with the invention.

FIG. **2** is a flow diagram describing a method for checking the integrity of an alterable ROM, in accordance with one aspect of the invention.

FIG. **3** is a flow diagram describing a method for updating or restoring the contents of an alterable ROM in accordance with a further aspect of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIG. **1** is a block diagram of a system in accordance with the present invention.

The system includes a CPU **100** with optional cache memory **102**, a permanent storage device **104**, such as a hard disk drive, random access memory **106**, an input device such as keyboard **108**, and an output device, such as display **110**. The system components are connected via bus **112**.

The system further comprises an unalterable ROM **114**, which stores various programs used to bootstrap the system at startup and provide basic low level system hardware access. Also provided is an alterable ROM **116**, such as a FLASH ROM, which stores additional bootstrapping and hardware access programs. The programs in the ROMs **114** and **116** together constitute first and second portions of a general bootstrap program.

Also provided in accordance with the invention is an alterable ROM corruption detect and repair means **118**. The means **118** can be implemented as software running in unalterable ROM **114**. Means **118** operates as described with respect to FIGS. **2** and **3** to detect unauthorized modifications to the alterable ROM **116**, and also either to restore the alterable ROM to its uncorrupted state, or to make authorized changes to the alterable ROM. Means **118** can either constitute part of unalterable ROM **114**, or reside in a separate hardware or software location in the system.

In one embodiment of the invention, a system bootstrap routine is stored in unalterable ROM **114**, the routine performing, when called, a signature computation on the current contents of the alterable ROM **116** and the current signature (stored in ROM **114** or elsewhere), and then passes control to the bootstrap code in the ROM **116** only if the signature is validated. Defect-caused or malicious changes to the FLASH ROM would therefore prevent the system from starting up correctly at the next power-on. The system could also be configured so that an attempt to update ROM **116** will cause an immediate restart from unalterable ROM **114**, immediately revealing a corrupted update. More complex implementations involve a secure-update module that guarantees (for example by monitoring instruction fetches using methods known to the art) that ROM **116** updates could be done only by code running from the ROM **116** itself, and each version of the alterable ROM could contain

Nintendo - Ancora Exh. 1003

signature-checking code, and reject any attempted update that did not meet a signature test.

A means for correcting alterable ROM **116**, in the event that it becomes corrupted, is also provided in accordance with the invention. The first programs to run when the system is started are entirely in unalterable ROMS; the first thing these programs do is check for a signal from the user that the alterable ROM **116** should be bypassed. If the signal is present, the first programs transfer control to other programs, also stored in unalterable ROM, that allow the user to examine and optionally replace the contents of the alterable ROM **116**; in doing this, they do not run any code from the alterable ROM, and they will work correctly even if the alterable ROM has become corrupted. If the signal from the user is not found, control passes to the normal system startup programs stored in alterable ROM **116**.

In one implementation of this invention, the unalterable ROM **114** is placed at the top of the memory space of a PC-compatible system, such as the one shown in FIG. **1**. When the system is first started, control passes to a small program in the unalterable ROM. This program checks to see if both control keys are being held down; if they are not, it passes control to the normal system-startup entry point stored in the alterable ROM **116**, which is located just below the unalterable ROM in the memory space. If the special key combination is being pressed, the program loads the first sector from the diskette in the A: drive, using a minimal diskette-input routine built into the unalterable ROM, and then passes control to it. The program on the diskette can then use the minimal unalterable diskette-input routine to read in the rest of itself, and then perform whatever user interaction is necessary to examine, verify, and if necessary, replace the contents of the alterable ROM.

In other possible implementations, the unalterable ROM could itself contain all the programming needed to interact with the user to examine and replace the contents of the alterable ROM. Many other signals from the user are also possible besides a specific key-combination; the unalterable ROM could check for a certain byte at a certain offset in the first sector of the diskette in the A: drive, or for the presence of a certain signal in the serial or parallel input port on the system, or for the position of a special switch added to the system for this purpose.

Another possible implementation would involve a section of protected code that would verify the alterable ROM at intervals during normal operation of the computer, by putting the code in a non-interceptable service routine that would get called whenever a timer-tic occurred, or whenever a disk was accessed (for instance), and warn the user if the signature check failed.

In any implementation of this invention, the code doing the signature calculation could ignore any desired part of the alterable ROM being checked, to allow for the saving of variable data in the alterable ROM (only code parts, and data parts that should not be subject to unauthorized update, need be checked.)

In all these cases, the manufacturer of the system would be able to provide correctly-signed alterable ROM updates (by virtue of holding the secret half of the asymmetric keypair used to generate and check digital signatures), but anyone else attempting to install an alterable ROM update would be unable to correctly do so without prohibitive amounts of computation (the exact amount of computation needed, and the speed of the signature check itself, would depend on the digital signature algorithm chosen, many of which are in the art).

A vulnerability of some implementations of this invention is that if the secret key of a key-pair is ever divulged, an attacker would be able to make unauthorized alterable ROM modifications on machines which use that key-pair for verification. To minimize the impact of this, a manufacturer could use a different key-pair for different subsets of the systems protected: depending on the needs of the specific situation, each machine model, or each submodel, could use a different key-pair. For greater security, at a cost of some convenience, the subsets could be made as small as desired. For instance, if the keypair were changed every few thousand machines manufactured, the publication of one secret key would expose only a few thousand machines to unauthorized modification; on the other hand, in order to install a legitimate update, the user would first have to determine which subset his machine's serial number corresponds to, and obtain the update tailored for that subset. In the extreme case, a different key-pair could be used for each machine: this involves more effort and more bookkeeping, but if one secret key is divulged, only one machine is compromised (only that single machine may have its alterable ROM contents altered without detection).

To protect against exposure of a private key, it might also be desirable to have a mechanism for updating the public key used for verification. For instance, a system might be designed so that, if it were determined that the private key protecting it has been exposed, a new public key could be installed by a special update process or by physically replacing a chip.

The methods by which the foregoing functions of the system are accomplished are described in detail with respect to FIGS. **2** and **3**.

During the initial setup of the machines at the manufacturer, the manufacturing process includes the following steps:

- For each group of some number of machines (for example, 40,000), a new public key/private key pair is selected for use in an asymmetric cryptosystem, such as RSA. The private key is stored securely at the manufacturer, and the public key is burned into the unalterable ROM **114**.
- After loading the initial contents of the alterable ROM **116**, the manufacturer computes a cryptographic hash (such as MD5) of the code portions and critical data portions of the contents, encrypts the hash value using the private_key, and stores the encrypted value as a signature at a fixed location in the alterable ROM.
- The reason for using the asymmetric cryptosystem in addition to the cryptographic hash is that then anyone possessing the public key can verify a ROM/hash pair, but only the manufacturer, possessing the private key, can generate a valid pair.

Referring to FIG. **2**, when a protected system is booted, either from power-up or any other form of system reset, control passes to code in the non-alterable ROM. That code performs the following steps (use of the "Escape" key in the following steps is, of course, just one example of a possible means for triggering a boot to the Minimal BIOS):

- If the Escape key on the keyboard is being held down (step **202**), immediately transfer control to the Minimal BIOS, stored in unalterable ROM **114** (the minimal BIOS is a program or set of programs residing in the unalterable ROM, and is described below), without executing any programs in the alterable ROM **218**. Otherwise, continue.
- Compute a cryptographic hash of the code portions and critical data portions of the contents of the alterable

Nintendo - Ancora Exh. 1003

ROM, using the same algorithm (e.g., MD5) that was used at initial alterable ROM load time (step **204**), and store the result as "HashActual".

Retrieve the encrypted form of the original hash value from the alterable ROM and store it as "EncHashCorrect"(step **206**), and decrypt it using the public key stored in unalterable ROM, to obtain the original stored hash value (step **208**). Store the decrypted result as "HashCorrect".

Compare the two hash values HashActual and HashCorrect (step **210**). If they match, pass control to the alterable ROM startup routine (step **212**).

If the two hash values do not match, print a warning message on the display (step **214**), wait for a keypress (step **216**), and transfer control to the Minimal BIOS in the unalterable ROM (step **218**).

When the manufacturer wants to create an update to the bootstrap programs and data in the alterable ROM, it generates an update file for each of the different private keys. To create the update file for a program protected by a given private key, the manufacturer:

Retrieves the private key for that program from secure storage.

Computes the cryptographic hash of the code portions and critical data portions of the alterable ROM contents, encrypts the result with the private key, and stores the encrypted value at the reserved offset in the alterable ROM image (i.e., a software image of the alterable ROM's contents).

Referring now to FIG. **3**, when it is desired to update the contents of the alterable ROM using the FLASH ROM image, it calls a special BIOS function in the unalterable ROM (step **302**), which performs the following steps:

Compute a cryptographic hash of the code portions and critical data portions of the requested new contents of the alterable ROM and store as "HashActual", using the same algorithm as used in the previous steps (e.g., MD5) (step **304**).

Retrieve the encrypted form of the stored hash value (i.e., the encrypted signature) from the appropriate place in the requested new contents of the alterable ROM, store it as "EncHashCorrect"(step **306**), and decrypt it using the public key stored in unalterable ROM to obtain the correct stored hash value (signature) (step **308**). Store as "HashCorrect".

Compare HashActual and HashCorrect (step **310**). If they match, execute the proper instructions to tell the hardware to update the contents of the alterable ROM (step **318**). The hardware, using methods known to the art, will only carry out the request if it came from the correct place in the unalterable ROM (step **316**).

If the two hash values do not match (step **310**) hardware detects that the update request did not come from the proper place in the unalterable ROM (step **316**), the update fails, and the alterable ROM is unchanged. The BIOS, or even the lower-level hardware, could provide the user with a visual or audible warning when this occurs (step **314**).

The Minimal BIOS referred to above can, for example, carry out the following functions (at no time does it execute any code from the alterable ROM):

Perform minimal functional tests of some parts of the system, including the diskette drive, display, and keyboard.

Interact with the user via the display and keyboard, allowing the user to perform a small set of functions, including:

Getting help on the Minimal BIOS functions,

Performing a alterable ROM update from an image file on diskette (using the alterable ROM update process described above),

Loading a more complete shell program from diskette, and passing control to it,

Passing control directly to the alterable ROM, even though it may have failed the integrity check,

Rebooting the system normally.

While the invention has been described in particular with respect to preferred embodiments, it will be recognized by those skilled in the art that modifications to the described embodiments can be effected without departing from the spirit and scope of the invention.

We claim:

1. A system, comprising:

a processor;

a storage device storing an operating system program for execution on the processor;

an alterable read only memory for storing data; and

a corruption detection device for detecting unauthorized changes to data in the alterable read only memory, the corruption detection device reading a signature, encrypted with a private key, that represents a non-corrupted version of data in the alterable read only memory, and further reading, from a secure memory location, a public key for decrypting the signature, the corruption detection device operating to compare the decrypted using public key, signature to a computed signature for detecting an occurrence of an unauthorized change to the data in the alterable read only memory.

2. The system of claim **1**, further comprising: an unalterable read only memory for storing the public key and for storing a first portion of a bootstrap program for controlling the system during a system initialization and subsequently transferring control of the system to the operating system.

3. The system of claim **2**, wherein the data stored in the alterable read only memory comprises a second portion of the bootstrap program.

4. The system of claim **3**, further comprising means for validly altering the second portion of the bootstrap program.

5. The system of claim **4**, wherein the means for validly altering comprises means for:

computing a hash of data constituting a proposed alteration;

reading an encrypted form of a signature representing the data constituting the proposed alteration;

decrypting the signature representing the data constituting the proposed alteration using the public key;

comparing the hash and the decrypted signature and if they match, writing the data constituting the proposed alteration to the second portion of the bootstrap program.

6. The system of claim **5**, wherein the means for validly altering comprises a computer program stored in the unalterable read only memory.

7. The system of claim **3**, wherein the corruption detection device comprises means, responsive to a triggering event, for:

computing a hash of the second portion of the bootstrap program;

reading the encrypted form of the signature and the public key from their storage locations;

decrypting the encrypted form of the signature using the public key; and

Nintendo - Ancora Exh. 1003

comparing the decrypted signature with the hash.

8. The system of claim **7**, further comprising means, if the decrypted signature matches the hash, for transferring control from the first portion of the bootstrapping program to the second portion of the bootstrapping program.

9. The system of claim **8**, wherein the triggering event is a power-up of the system.

10. The system of claim **8**, wherein the triggering event is the actuation of a system reset switch.

11. The system of claim **8**, wherein the triggering event is the actuation of a combination of keys on a keyboard coupled to the processor.

12. The system of claim **1**, wherein the alterable read only memory is a FLASH ROM.

13. The system of claim **1**, further comprising means for actuating the corruption detection device periodically while the system is in operation.

14. The system of claim **1**, further comprising means for replacing the public key with a new authorized public key.

15. A method, comprising the steps of:

storing data in an alterable read only memory of a computer system;

storing in a first memory location in the system an encrypted signature representing a valid copy of the data in the alterable read only memory;

storing in a second memory location in the system a public key to the encrypted signature;

in response to a triggering event, computing a current signature for the data stored in the alterable read only memory, decrypting the signature representing the valid copy using the public key, and comparing the decrypted signature and the current signature to determine the validity of the data stored in the alterable read only memory.

16. The method of claim **15**, wherein the alterable read only memory is a FLASH ROM.

17. The method of claim **15**, wherein the triggering event is a power up of the computer system.

18. The method of claim **15**, wherein the triggering event is the actuation of a system reset switch.

19. The method of claim **15**, wherein the triggering event is the actuation of a combination of keys on a keyboard coupled to the system.

20. The method of claim **15**, further comprising a step of storing a first portion of a system bootstrapping program in an unalterable read only memory; and wherein the data stored in the alterable read only memory comprises a second portion of the system bootstrapping code.

21. The method of claim **20**, wherein the steps of computing a current signature, decrypting the signature and comparing the decrypted signature and the current signature are performed while the system is under the control of the first portion of the system bootstrapping program.

22. The method of claim **21**, further comprising, if the data in the alterable read only memory are valid, passing control of the system from the first portion of the bootstrapping program to the second portion of the bootstrapping program.

23. The method of claim **20**, further comprising a step of replacing data in the alterable read only memory with valid replacement data.

24. The method of claim **23**, further comprising a step of determining whether a group of candidate replacement data is valid replacement data by:

computing a hash of the data constituting the candidate replacement data;

decrypting an encrypted signature representing the data constituting the candidate replacement data;

comparing the hash and the decrypted signature, and if they match, designating the candidate replacement data as valid replacement data.

25. The method of claim **20**, wherein the first and second memory locations are in the unalterable read only memory.

26. The method of claim **15**, wherein the triggering event is executed periodically during operation of the system.

27. A method for operating a data processing system, comprising steps of:

partitioning a bootstrap program between an unalterable read only memory device and an alterable memory device;

storing, in the alterable memory device, private key encrypted validity data representing a portion of the bootstrap program stored in the alterable memory device;

storing, in the unalterable read only memory device, a public key for decrypting the private key encrypted validity data;

in response to a triggering event, executing a portion of the bootstrap program stored in the unalterable read only memory device, the executed portion of the bootstrap program first computing validity data for at least some of the content of the alterable memory device, then using the stored public key to decrypt the private key encrypted validity data, and then comparing the decrypted validity data to the computed validity data; and

transferring control of the bootstrap program from the portion stored in the unalterable read only memory device to the portion stored in the alterable memory device only if the result of the comparison indicates that no unauthorized modifications have been made to the content of the alterable memory device.

28. A method as in claim **27**, wherein the alterable memory device is a FLASH ROM.

* * * * *

APPENDIX B-10

US005835594A

United States Patent [19]

Albrecht et al.

[11] **Patent Number: 5,835,594**

[45] **Date of Patent: Nov. 10, 1998**

[54] **METHODS AND APPARATUS FOR PREVENTING UNAUTHORIZED WRITE ACCESS TO A PROTECTED NON-VOLATILE STORAGE**

[75] Inventors: **Mark Albrecht**, Banks; **Frank Wildgrube**, Hillsboro, both of Oreg.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **598,803**

[22] Filed: **Feb. 9, 1996**

[51] **Int. Cl.**[6] **H04L 9/00**

[52] **U.S. Cl.** **380/23**; 25/4

[58] **Field of Search** 380/3, 4, 23, 24, 380/25, 30

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,278,837	7/1981	Best	178/22.09
5,022,077	6/1991	Bealkowski et al.	380/4
5,144,659	9/1992	Jones	380/4
5,289,540	2/1994	Jones	380/4
5,377,264	12/1994	Lee et al.	380/4
5,421,006	5/1995	Jablon et al.	380/4
5,465,299	11/1995	Matsumoto et al.	380/23
5,479,509	12/1995	Ugon	380/23

OTHER PUBLICATIONS

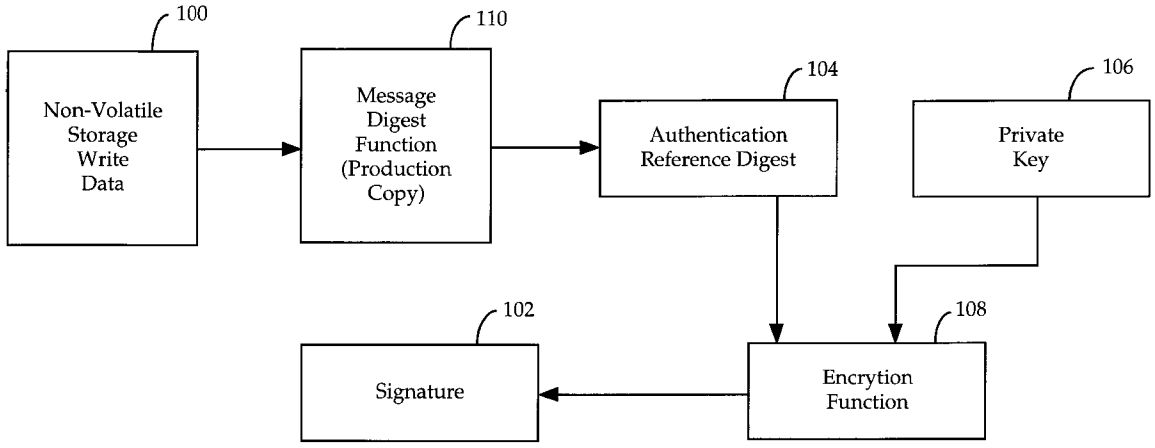
PCT International Search Report for International Application No. PCT/US97/01965, dated Jun. 9, 1997.

Primary Examiner—Thomas H. Tarcza
Assistant Examiner—Carmen D. White
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

An electronic signature is generated in a predetermined manner and attached to a transferable unit of write data, to facilitate authenticating the write data before allowing the write data to be written into a protected non-volatile storage. The write data is authenticated using a collection of secured authentication functions. Additionally, the actual writing of the authenticated write data into the protected non-volatile storage is performed by a secured copy utility.

23 Claims, 7 Drawing Sheets



Nintendo - Ancora Exh. 1003

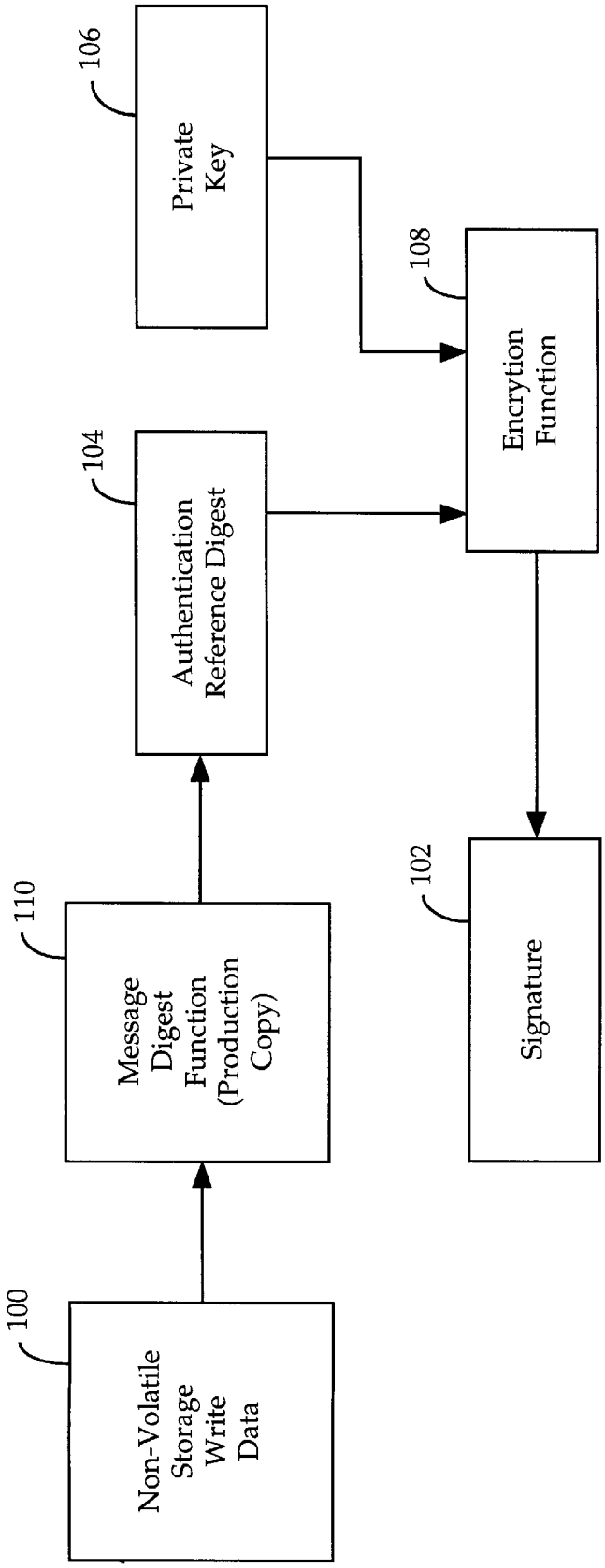

100
Non-Volatile Storage Write Data
110
Message Digest Function (Production Copy)
104
Authentication Reference Digest
106
Private Key
108
Encrytion Function
102
Signature

Figure 1

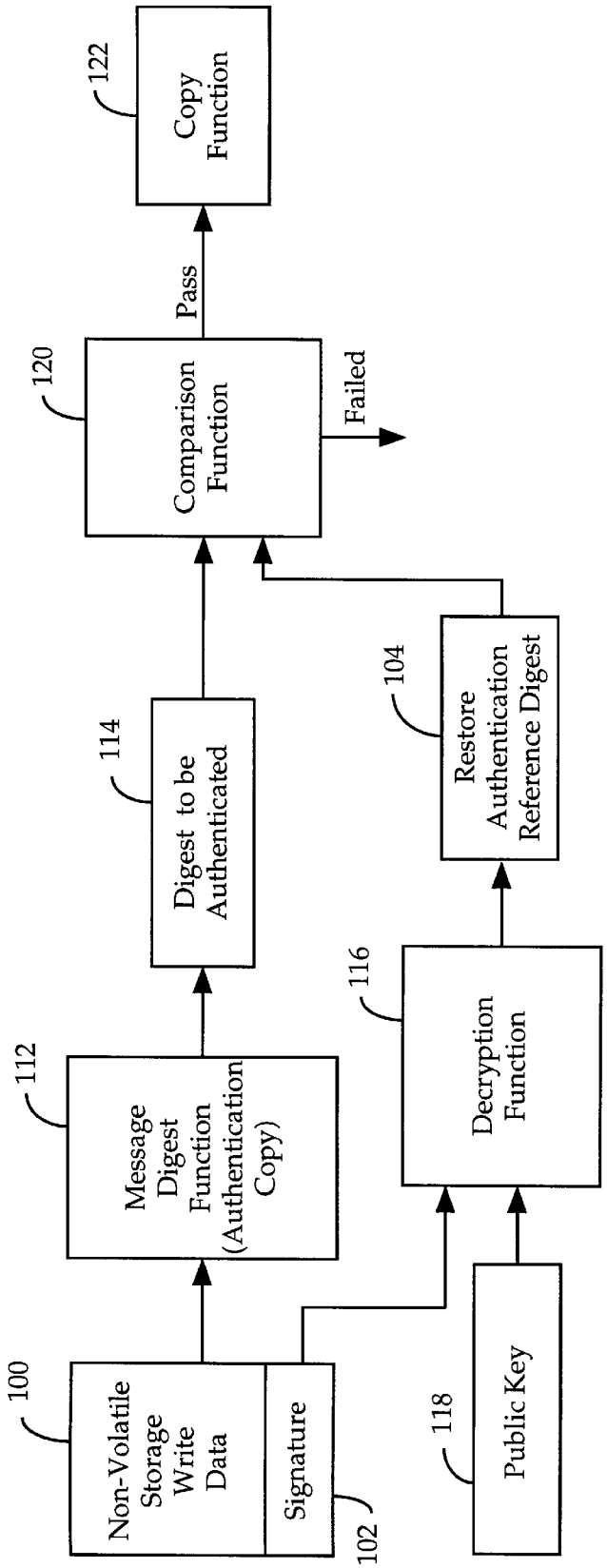

100
Non-Volatile Storage Write Data
Signature
102
112
Message Digest Function (Authentication Copy)
114
Digest to be Authenticated
116
Decryption Function
118
Public Key
104
Restore Authentication Reference Digest
120
Comparison Function
Failed
Pass
122
Copy Function

Figure 2

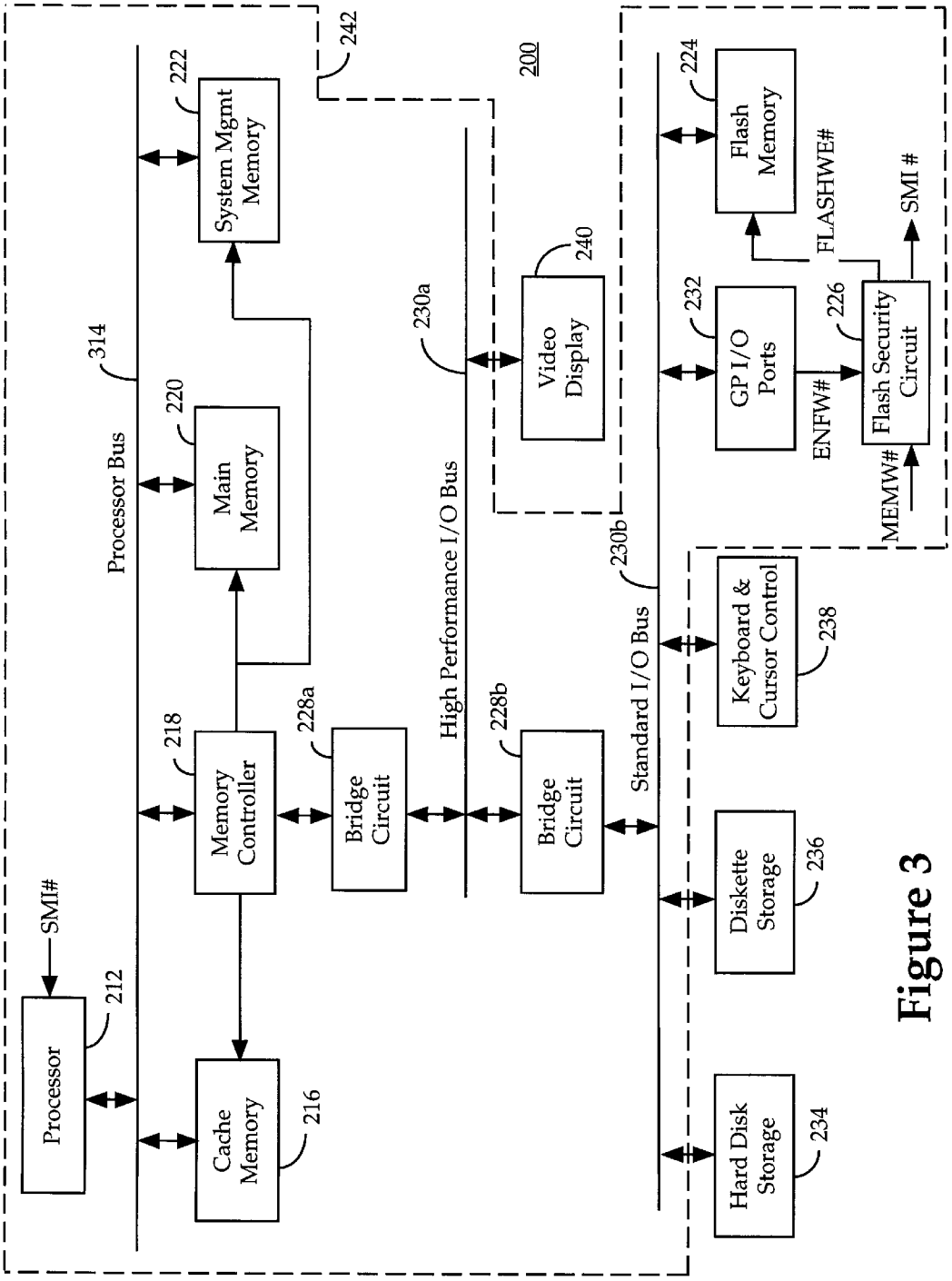
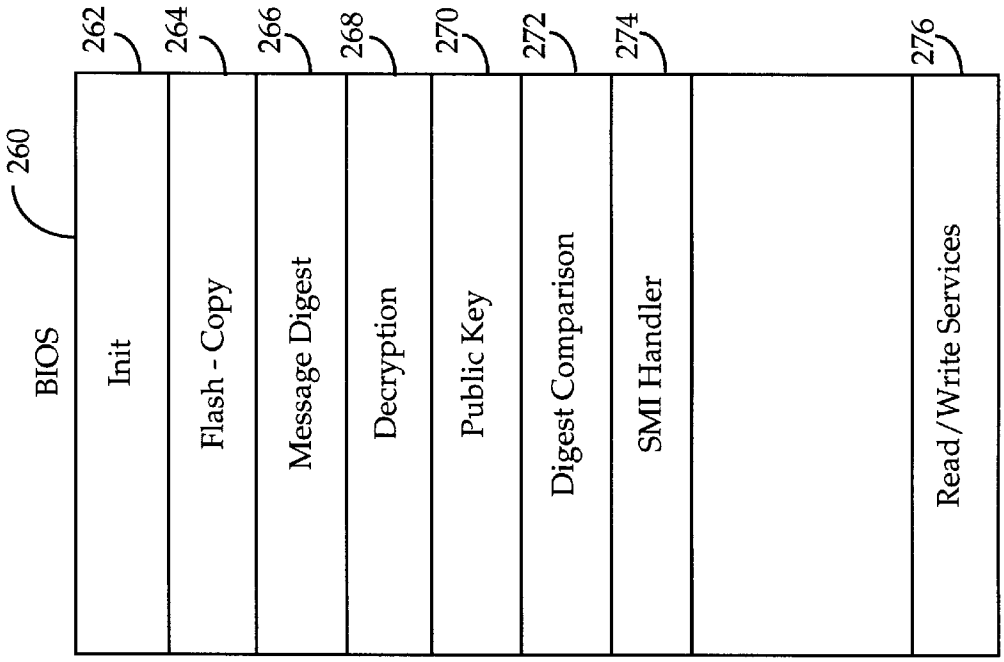

Processor
SMI#
212
Cache Memory
216
Memory Controller
218
Main Memory
220
System Mgmt Memory
222
Processor Bus
314
242
Bridge Circuit
228a
High Performance I/O Bus
230a
Bridge Circuit
228b
Video Display
240
200
Standard I/O Bus
230b
Hard Disk Storage
234
Diskette Storage
236
Keyboard & Cursor Control
238
GP I/O Ports
232
Flash Memory
224
FLASHWE#
ENFW#
Flash Security Circuit
226
MEMW#
SMI #

Figure 3


BIOS
260
Init
262
Flash - Copy
264
Message Digest
266
Decryption
268
Public Key
270
Digest Comparison
272
SMI Handler
274
Read/Write Services
276

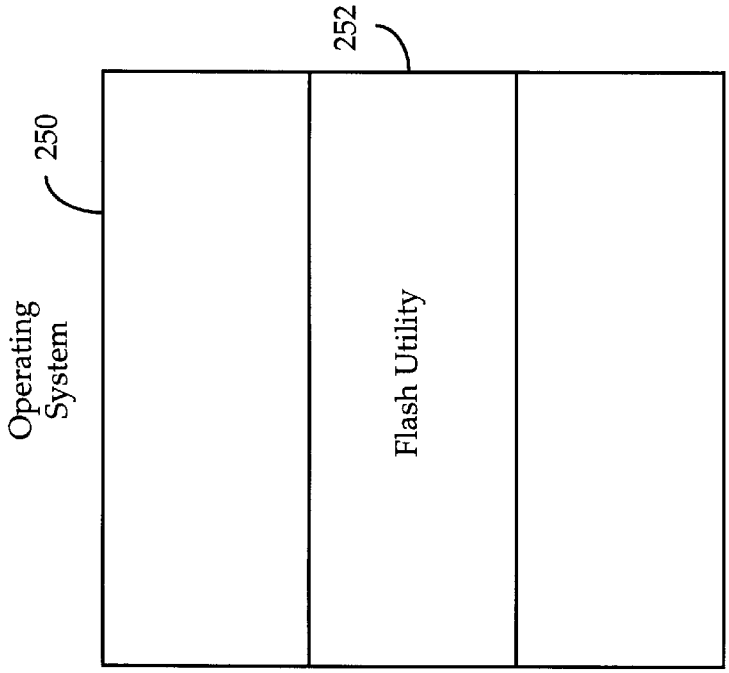

Operating System
250
Flash Utility
252

Figure 4

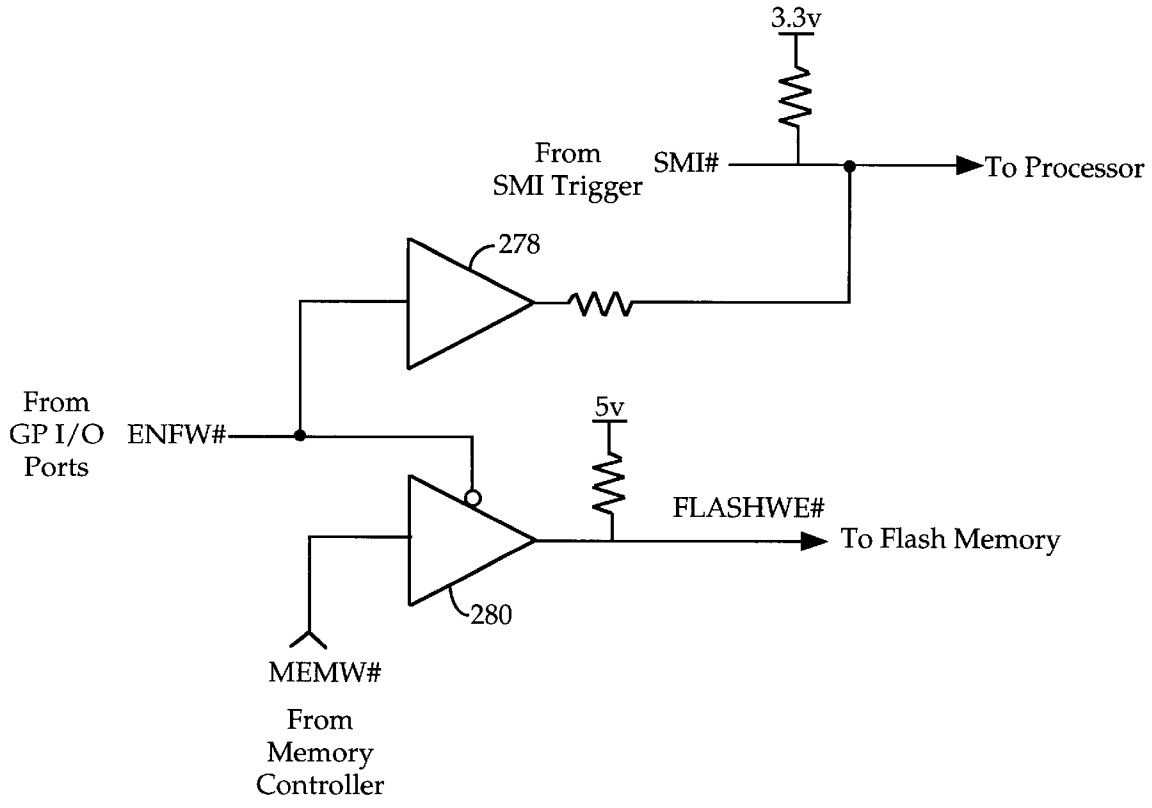

3.3v
From
SMI Trigger
SMI#
To Processor
278
From
GP I/O
Ports
ENFW#
5v
FLASHWE#
To Flash Memory
280
MEMW#
From
Memory
Controller

Figure 5

System Management Mode (Prior Art)

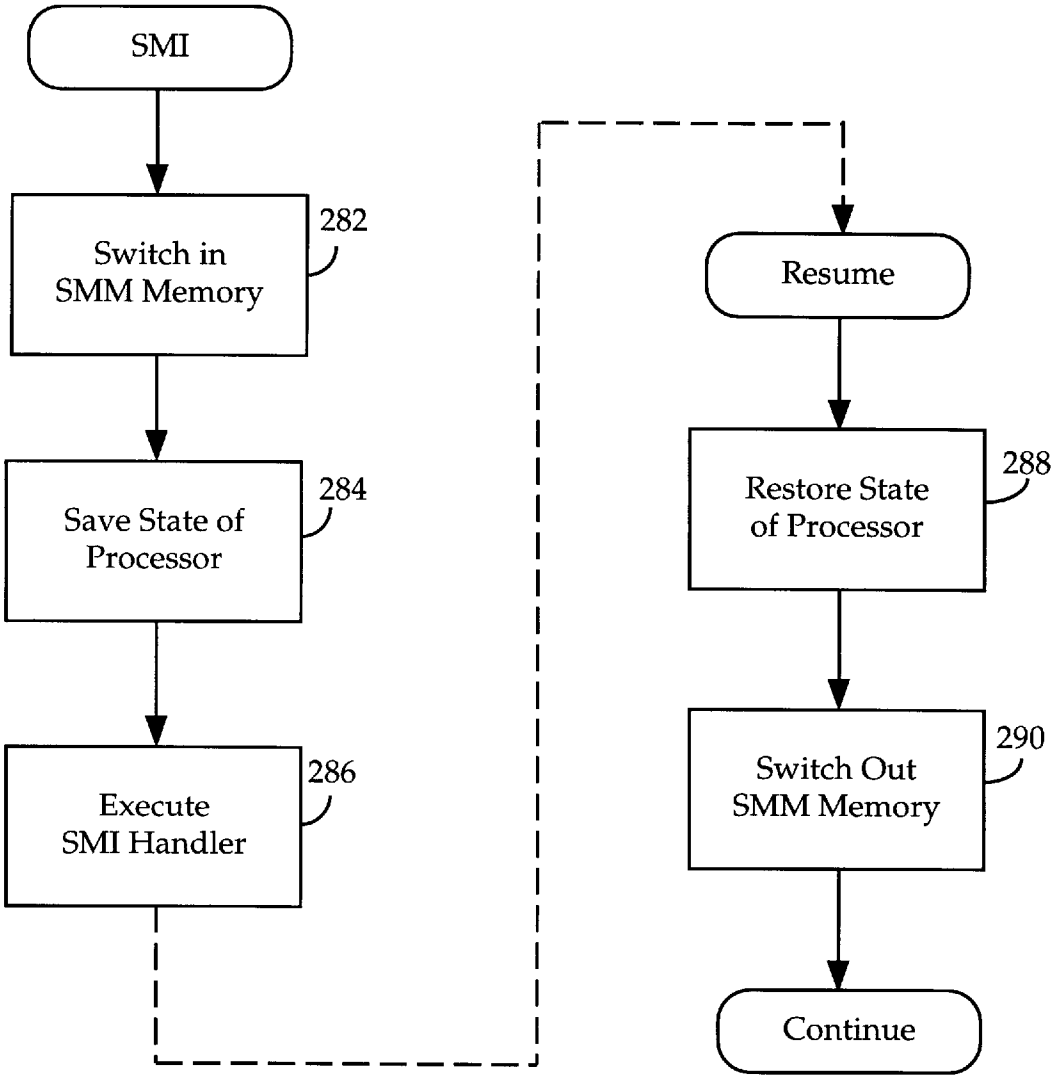


Figure 6

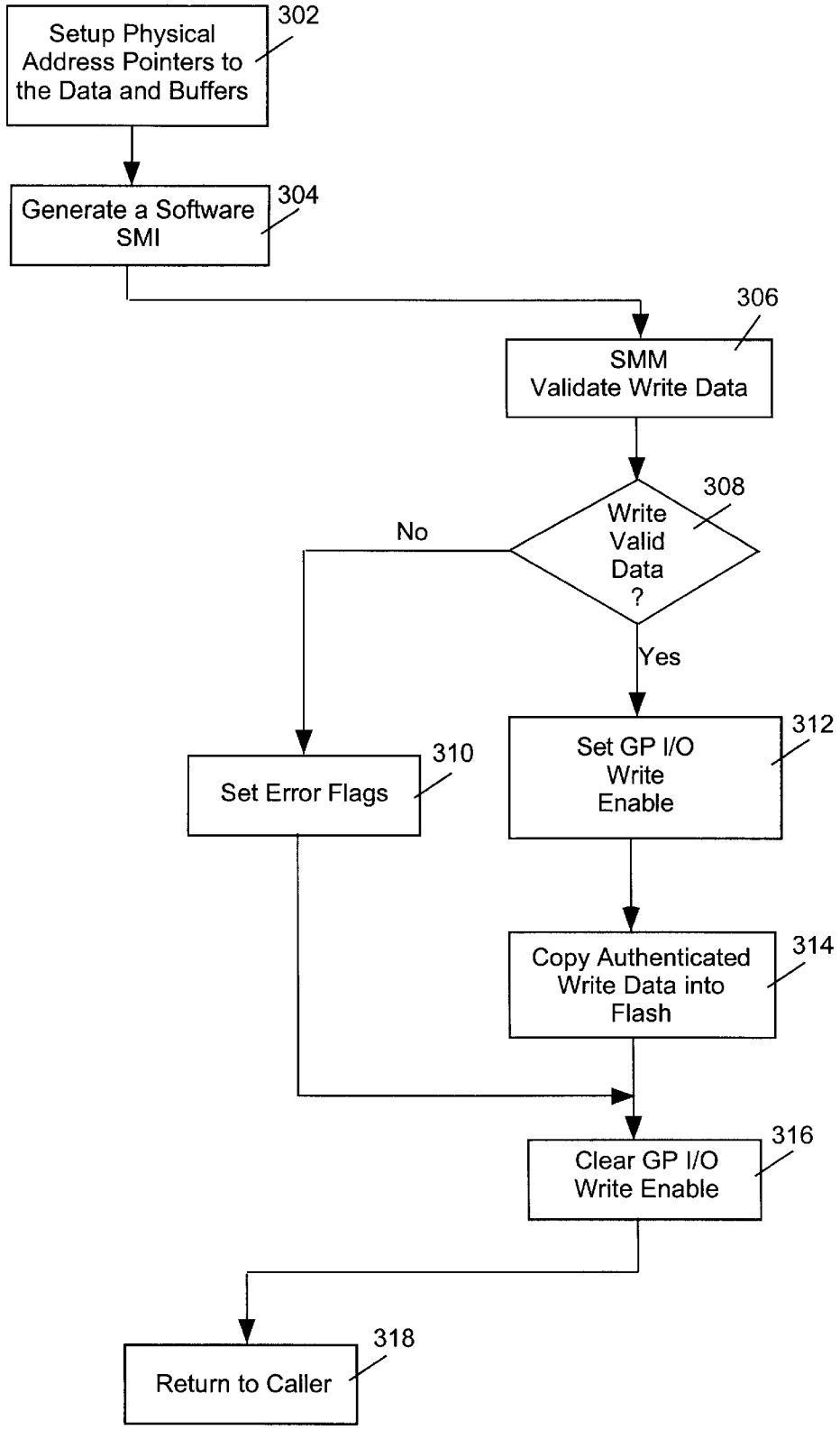

Setup Physical Address Pointers to the Data and Buffers
302
Generate a Software SMI
304
306
SMM Validate Write Data
308
Write Valid Data ?
No
Yes
Set Error Flags
310
Set GP I/O Write Enable
312
Copy Authenticated Write Data into Flash
314
Clear GP I/O Write Enable
316
Return to Caller
318

Figure 7

METHODS AND APPARATUS FOR PREVENTING UNAUTHORIZED WRITE ACCESS TO A PROTECTED NON-VOLATILE STORAGE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of computer systems. More specifically, the present invention relates to data security on computer systems.

2. Background Information

Existing methods of preventing unauthorized write access to nonvolatile storage such as FLASH memory typically rely on "secret" access methods to a write enable circuit. These "secret" access methods to the write enable circuit can be reverse-engineered through the use of standard debugging hardware. Once reverse engineered, a person will be able to produce code that can write to the "protected" non-volatile storage at will. If the code is used in a malicious manner, it can be used to introduce viruses into the "protected" non-volatile storage or even destroy the content of the non-volatile storage.

Thus, it is desirable to have a more robust approach to preventing unauthorized access to non-volatile storage, in particular, an approach that does not rely on the access method not being known. As will be described in more detail below, the present invention achieves these and other desirable results.

SUMMARY OF THE INVENTION

In accordance to the present invention, an electronic signature is generated in a predetermined manner and attached to a transferable unit of write data, to facilitate authenticating the write data before allowing the write data to be written into a protected non-volatile storage. The write data is authenticated using a collection of secured authentication functions. Additionally, the actual writing of the authenticated write data into the protected non-volatile storage is performed by a secured copy utility.

The electronic signature is functionally dependent on the content of the write data, and the predetermined manner of generating the electronic signature is reproducible during write time. In one embodiment, the electronic signature is generated by the creator of the write data, by generating a digest based on the content of the write data using a message digest function, and then encrypting the generated digest with a secret private key using an encryption function.

The collection of secured authentication functions include a secured corresponding copy of the message digest function, and a secured complementary decryption function. During operation, the secured decryption function reconstitutes the original digest by decrypting the electronic signature with a secured complementary public key, while the secured copy of the message digest function generates another digest based on the content of the write data to be authenticated. The two digests are compared using a secured comparison function. If the two digests pass the comparison, the secured copy utility is invoked to copy the authenticated write data into the protected non-volatile storage, otherwise, the write data are rejected.

In one embodiment, the authentication functions are secured by copying them into a normally unavailable system management memory during system initialization. The authentication functions are invoked using a system management interrupt (SMI), which when asserted, automatically maps the system management memory into the normal system memory space. A non-volatile memory write security circuitry is provided to qualify a memory write signal provided to the protected non-volatile storage, and to generate the SMI whenever a write to the protected non-volatile storage is requested.

BRIEF DESCRIPTION OF DRAWINGS

The present invention will be described by way of exemplary embodiments, but not limitations, illustrated in the accompanying drawings in which like references denote similar elements, and in which:

FIGS. **1–2** illustrate the essential elements of the present invention, and their interrelationships with each other;

FIG. **3** illustrates an exemplary computer system incorporated with the teachings of the present invention on securing the authentication functions;

FIG. **4** illustrates the system BIOS, and for one embodiment, the operating system of the exemplary computer system in further detail;

FIG. **5** illustrates the FLASH security circuitry of FIG. **3** in further detail;

FIG. **6** illustrates execution flow of the exemplary computer system under a system management mode; and

FIG. **7** illustrates one embodiment of the execution flow for writing into FLASH memory.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, for purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known features are omitted or simplified in order not to obscure the present invention. Furthermore, for ease of understanding, certain method steps are delineated as separate steps, however, these separately delineated steps should not be construed as necessarily order dependent in their performance.

Referring now to FIGS. **1** and **2**, two block diagrams illustrating the essential elements of the present invention, and their interrelationships to each other are shown. As illustrated, a transferable unit of non-volatile storage write data **100** is provided with an electronic signature **102** to facilitate authenticating write data **100** prior to allowing write data **100** to be written into a non-volatile storage. Preferably, electronic signature **102** is "attached" to write data **100**. Examples of a transferable unit include a file, or a block, whereas examples of non-volatile storage include FLASH memory or erasable programmable read-only-memory (EPROM). Examples of write data is system basic input/output service (BIOS) updates, such as additions, deletions and modifications. For many applications, it is expected that electronic signature **102** is generated and "attached" to write data **100** at the time write data **100** is created.

For the illustrated embodiment, electronic signature **102** is generated by encrypting a reference digest **104** with a secret private key **106** using an encryption function **108**. The reference digest **104** is generated using a message digest function **110**. In other words, the content of reference digest **104** is functionally dependent on the content of write data **100**. Accordingly, the content of electronic signature **102** is also functionally dependent on the content of write data **100**.

Nintendo - Ancora Exh. 1003

At write time, a secured corresponding copy of message digest function **112** generates a "new" digest **114** in real time. At the same time, a secured complementary decryption function **116** reconstitutes original reference digest **104** by decrypting electronic signature **102** using secured complementary public key **118**. The two digests **104** and **114** are provided to a secured comparison function **120** to determine if they are identical. The two digests **104** and **114** are identical if write data **100** is authentic, since both digests **104** and **114** are functionally dependent on the contents of write data **100**, generated by copies of the same message digest function **110** and **112**, and the encryption were decrypted in a complementary manner. If the two digests **104** and **114** compared successfully, a secured copy function **122** is notified to perform the actual writing into the protected non-volatile storage, otherwise the write data is rejected.

Encryption and decryption functions **108** and **116** may implement any one of a number of private/public key encryption/decryption techniques known in the art. Similarly, message digest function **110**/**112** may also implement any one of a number of message digest techniques known in the art. For further information on private/public key encryption/decryption techniques, see e.g. Hellman et al., Public Key Cryptographic Apparatus and Method, U.S. Pat. No. 4,218,582, and Rivest et al., Cryptographic Communications System and Method, U.S. Pat. No. 4,405,829; and for further information on message digest, see e.g. Method for Identifying Subscribers and for Generating and Verifying Electronic Signatures in a Data Exchange System, U.S. Pat. No. 4,995,082, and Rivest, The MD5 Message Digest Algorithm, Request For Comment (RFC) 1321, April 1992.

Creation of electronic signature **102** and associating it with write data **100** as described above, may be practiced in any number of computer systems known in the art, provided they are equipped to store and execute message digest function **110** and encryption function **108**. It is anticipated that for most applications, creation of electronic signature **102** will be practiced on the same computer system where write data **100** is created. For example, for the above mentioned system BIOS update application, it is anticipated that the system BIOS updates and electronic signature **102** will be generated and associated at the same time and on the same computer system.

FIG. **3** illustrates an exemplary computer system **200** incorporated with the teachings of the present invention on authenticating write data before allowing the write data to be written into a protected non-volatile storage. Exemplary computer system **200** includes processor **212**, processor bus **214**, cache memory **216**, memory controller **218**, and a plurality of other memory units **220**–**224** coupled to each other as shown. Other memory units **220**–**224** include main memory **220**, system management memory **222**, and FLASH memory **224**. In accordance to the present invention, exemplary computer system **200** includes in particular FLASH security circuitry **226**. Additionally, computer system **200** includes bridge circuits **228***a*–**228***b*, high performance and standard (input/output) I/O buses **230***a*–**230***b*, general purpose I/O (GPIO) ports **232**, hard and diskette storages **234**–**236**, keyboard and cursor control device **238**, and display **240**, coupled to each other and the above enumerated elements as shown.

For the illustrated embodiment, buses **214**, **230***a* and **230***b* are disposed on motherboard **242**. Elements **212**, **216**–**226**, **228***a*–**228***b* and **232** are either removably interconnected to motherboard **242** via sockets (not shown) or "soldered" onto motherboard **242**, whereas elements **234**–**238** are coupled to motherboard **42** through cables and connectors (not shown).

Processor **212** performs the conventional function of executing code. Processor **212** is equipped to execute code in multiple modes including a system management mode (SMM). Processor **212** is also equipped to respond to a wide variety of interrupts including a system management interrupt (SMI), which places processor **212** in SMM. Memory controller **218** and volatile memory units **216**, **220** and **222** perform the conventional functions of controlling memory access, and providing execution time storage respectively. In particular, for each write access to memory, memory controller **218** generates a MEMW# signal for the addressed memory unit. Memory controller **218** normally does not map system management memory **222** as part of the normal system memory space. System management memory **222** is mapped into the system memory space, when processor **212** enters SMM. Furthermore, except for system initialization, processor mode transition, and execution in SMM, system management memory **222** is write disabled.

FLASH memory **224** performs its conventional function of providing non-volatile storage respectively. In particular, FLASH memory **224** stores system BIOS. During system initialization, the bulk of the system BIOS that are not security sensitive are loaded into main memory **220**, whereas the remaining system BIOS (including in particular the write data authentication functions) that are security sensitive are loaded into system management memory **224**. Flash security circuit **226** protects FLASH memory **224** from unauthorized write accesses, by keeping FLASH memory **224** write disabled, and generating an SMI to invoke the secured system BIOS write data authentication functions in system management memory **222** to authenticate the write data, whenever it enables FLASH memory **224** for a write access. General purpose I/O ports **232** also perform their conventional functions for providing I/O ports to a variety of peripherals. In particular, one of the I/O ports is used to notify FLASH security circuit **226** of a write request to FLASH memory **224**. The write request is denoted by writing to a corresponding register of the I/O port using a standard I/O instruction of exemplary computer system **200**.

Hard disk storage **234** also performs the conventional function of providing non-volatile storage. In particular, hard disk storage **234** stores operating system of exemplary computer system **200**. During system initialization, operating system is loaded into main memory **220**. All other elements perform their conventional function known in the art. Except for the particularized functions and/or requirements, all enumerated elements are intended to represent a broad category of these elements found in computer systems.

FIG. **4** illustrates system BIOS and operating system of exemplary computer system **200** in further detail. As shown, system BIOS **260** includes init function **262**, FLASH copy utility **264**, message digest function **266**, decryption function **268**, public key **270**, digest comparison function **272**, SMI handler **274** and read/write service **276**, whereas, for some embodiments, operating system **250** includes FLASH utility **252**.

Init function **262** initializes system BIOS **260** during system initialization, including loading FLASH copy utility **264**, message digest function **266**, decryption function **268**, public key **270**, digest comparison function **272**, and SMI handler **274** into system management memory **222**. As described earlier, system management memory **222** is normally not mapped into system management space, unless a SMI is triggered placing processor **212** in SMM, and system management memory **222** is write disabled except for

Nintendo - Ancora Exh. 1003

initialization, processor mode transition, and execution in SMM. Accordingly, these system BIOS functions are secured from malicious modification.

SMI handler **274** services SMIs, invoking other functions (including the write data authentication functions) as necessary, depending on the cause of a particular SMI. As will be described in more detail below, SMI handler **274** is given control upon entry into SMM. As described earlier, message digest **266** generates a digest in real time for the write data of a FLASH write request, in accordance to the content of the write data, and decryption function **268** decrypts the electronic signature "attached" to the write data of the FLASH write request using public key **270**, to reconstitute the FLASH write data's original digest. Digest comparison function **272** compares the two digests, and finally FLASH copy utility **264** performs the actual writing of the authenticated data into FLASH memory **224**. Message digest function **266**, decryption function **268**, digest comparison function **272**, and FLASH copy utility **264** are invoked in due course by SMI handler **274** upon determining that a SMI is triggered by FLASH security circuitry **226**.

Read/Write services **276** provides read and write services to I/O devices. Read/Write services **276** are among the bulk of the BIOS functions that are loaded into main memory **220** during system start up.

For some embodiments, FLASH utility **252** is included to perform various FLASH related functions including in particular copying of FLASH write data from an external source medium to a buffer in main memory **220**, and then copying the FLASH write data from the buffer into FLASH memory **224** by way of read/write services **276**, which invokes message digest function **266**, decryption function **268**, etc., to validate the FLASH write data, and if validated, FLASH copy utility **264** to perform the actual writing, to be described more fully below. Examples of such FLASH write data are system BIOS additions, deletions, and modifications described earlier, and an example of an external source medium is a diskette.

FIG. **5** illustrates FLASH security circuit **226** in further detail. As shown, FLASH security circuit **226** includes first and second drivers **278** and **280**. The input (ENFW#) of first driver **278** is provided by one of the I/O ports of GPIO ports **232**, whereas the output of first driver **278** is coupled to a signal line coupling a SMI trigger mechanism to processor **212**. Thus, whenever, GPIO ports **232** sets ENFW# active to enable write access, in response to a FLASH write request, first driver **278** causes a SMI to be triggered for processor **212**.

The inputs (ENFW# and MEMW#) of second driver **280** are provided by the same I/O port of general purpose I/O ports **232** and memory controller **218** respectively, whereas the output (FLASHWE#) of second driver **280** is provided to FLASH memory **224**. FLASHWE# is tri-stated. FLASHWE# becomes active, when both MEMW# and ENFW# are active. In other words, the write signal (MEMW#) from memory controller **218** is qualified by ENFW#, which at the same time through first driver **278** would cause a SMI to be triggered. Thus, the secured authentication functions stored in system management memory **222** would be invoked to authenticate the write data before allowing them to be written into FLASH memory **224**.

FIG. **6** illustrates execution flow of the exemplary computer system in SMM. As shown, upon detection of an SMI, processor **212** directs memory controller **218** to switch in and map system management memory **222** as part of the system memory space, and in response, memory controller **218** performs the requested switching and mapping accordingly, step **282**. Next, processor **212** saves the processor state into system management memory **222**, step **284**. Upon saving the processor state, processor **212** transfers execution control to pre-stored SMI handler **274**, step **286**.

SMI handler **274** then determines the cause of the SMI and services the SMI accordingly, invoking other routines such as the authentication functions as necessary. Upon servicing the SMI, SMI handler **274** executes a Resume instruction to transfer execution control back to the interrupted programs. In response, processor **212** restores the saved processor state from system management memory **222**, step **288**. Furthermore, processor **212** directs memory controller **218** to unmap system management memory **222** from the system memory space and switch out system management memory **222**. In response, memory controller **218** performs the requested unmapping and switching accordingly, step **290**.

As a result, the SMI is serviced in a manner that is transparent to the executing operating system, subsystems as well as applications. In other words, an SMI is a transparent system service interrupt.

FIG. **7** illustrates one embodiment of the execution flow for writing data into FLASH memory **224**. As shown, in response to a write request from an application, such as FLASH utility **252** described earlier, read/write services **276** set up the physical address pointers to the write data, step **302**. Next, for the illustrated embodiment, read/write services **276** generate a software SMI to enter SMM and to provide the SMI handler with the physical address pointers of the write data, step **304**. A software SMI is used and preferred at this point in time as opposed to the designated GPIO port **232** because FLASH memory would remain disabled during the authentication process.

Upon entry into SMM, as described earlier, SMI handler **274** is given control. Upon ascertaining the reason for the SMI, SMI handler **274** invokes message digest **266** and decryption function **268** to authenticate the write data identified by the physical address pointers, step **306**. If the write data fails the authentication process, step **308**, SMI handler **274** sets the appropriate error flags, step **310**, clears the designated GPIO port, step **316**, and exits SMM. Upon given control again, read/write services **276** returns to the caller, after performing the necessary "clean ups".

On the other hand, if at step **308**, the write data passes the authentication process, SMI handler **274** enables write to FLASH memory **224**, by setting the designated GPIO port **232**, step **312**. Once enabled, the authenticated write data are copied into FLASH memory **224**, step **314**. After all authenticated write data have been copied, as described earlier, SMI handler **274** clears the designated GPIO port **232**, and exits SMM. Upon given control again, read/write services **276** returns to the caller, after performing the necessary "clean ups".

As described earlier, when SMI handler **274** enables write to FLASH memory **224** by way of the designated GPIO port, in addition to enabling FLASH memory **224** for write, a SMI is triggered. However, since this "new" SMI is triggered while the system is in SMM, the "new" SMI is discarded. The reason why the "new" SMI is triggered is because for the illustrated embodiment, the designated GPIO port **232** may be set outside SMM. The "automatic" SMI will ensure that the write data will be authenticated in the event that happens, preventing any possibility of bypassing the authentication process.

Nintendo - Ancora Exh. 1003

Thus, methods and apparatus for preventing unauthorized access to a protected non-volatile memory have been described. While the method and apparatus of the present invention has been described in terms of the above illustrated embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described. The present invention can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of restrictive on the present invention.

What is claimed is:

1. In a computer system comprising a non-volatile storage having stored therein data content, a computer implemented method for protecting the non-volatile storage from unauthorized write access that would result in unauthorized modification of the stored data content, the method comprising the steps of:

a) pre-storing a plurality of associated authentication functions in the non-volatile storage, reading the plurality of associated authentication functions from the non-volatile storage during computer system initialization, and securing the plurality of associated authentication functions on the computer system, the associated authentication functions operative to authenticate write data of a write access to the non-volatile storage using an electronic signature the content of which being functionally dependent on the content of the write data; and

b) selectively invoking the associated authentication functions to authenticate the write data of subsequent write accesses to the non-volatile storage during operation, allowing only authenticated write data to be written into the non-volatile storage.

2. The computer implemented method as set forth in claim **1**, wherein step (a) comprises securing the authentication functions in a secured portion of memory of the computer system.

3. The computer implemented method as set forth in claim **1**, wherein the associated authentication functions of step (a) are implemented as a plurality of system basic input/output services (BIOS) of the computer system; and wherein step (a) comprises securing the associated authentication functions by copying the plurality of system BIOS implementing the associated authentication functions into system management memory of the computer system during system initialization, wherein the system management memory is not mapped into a normal system memory space of the computer system unless the computer system is executing in a system management mode, and wherein the system management memory is write protected except for system initialization and system execution mode transition.

4. The computer implemented method as set forth in claim **1**, wherein the associated electronic signature is generated by encrypting a first digest with a secret private key, the first digest being generated based on the content of the write data of the write access; and step (b) comprises

(b.1) providing read accessibility to the secured associated authentication functions;

(b.2) invoking a secured decryption function of the secured associated authentication functions reconstitute the first digest by decrypting the associated electronic signature using a secured public key complementary to the secret private key,

(b.3) invoking a secured message digest function of the secured associated authentication functions to generate a second digest based on the content of the write data of the write access, and

(b.4) invoking a secured digest comparison function of the secured associated authentication functions to determine if the write data of the write access is authentic by comparing the first and second digests.

5. The computer implemented method as set forth in claim **4**, wherein step (b) further comprises step (b.5) conditionally invoking a secured copy utility of the secured associated authentication functions to copy the write data into the protected non-volatile storage if the first and second digests compared successfully in step (b.4).

6. A computer system comprising:

(a) a non-volatile storage having stored therein data content;

(b) a plurality of authentication functions associated with the data content and stored in the non-volatile storage, operative to authenticate write data of a write access to the non-volatile storage during operation, the associated authentication functions operative to authenticate the write data using an electronic signature which is functionally dependent on the content of the write data;

(c) a secured memory unit operative to store and secure the plurality of associated authentication functions read from the non-volatile storage during system initialization of the computer system; and

(d) a processor coupled to the non-volatile storage and the secured memory unit operative to selectively invoke the associated authentication functions during operation of the computer system to authenticate the write data of subsequent write accesses to the non-volatile storage, protecting the non-volatile storage from unauthorized write access that would result in unauthorized modification of the stored data content.

7. The computer system as set forth in claim **6**, wherein the plurality of authentication functions include

a decryption function for reconstituting a first digest by decrypting the electronic signature with a public key, the electronic signature being generated by encrypting the first digest with a secret private key in a complementary manner,

a message digest function for generating a second digest based on the content of the write data of the write access in the same manner the first digest was generated, and

a digest comparison function for determining whether the write data of the write access is authentic by comparing the first and second digests.

8. The computer system as set forth in claim **7**, wherein the decryption function, the message digest function and the digest comparison function are implemented as a plurality of system basic input/output services (BIOS) of the computer system, which are copied into the secured memory unit during system initialization, wherein the secured memory unit is not mapped into a normal system memory space of the computer system unless the processor is executing in system management mode, and wherein the secured memory unit is write protected except for system initialization and processor execution mode transition.

9. The computer system as set forth in claim **8**, wherein

the non-volatile storage is a FLASH memory storage unit for storing system BIOS;

the decryption function, the message digest function, the digest encryption function and the public key are pre-stored in the FLASH memory storage unit;

the computer system further includes main memory coupled to the processor; and

Nintendo - Ancora Exh. 1003

the write data of the write access are system BIOS updates staged in a buffer in the main memory.

10. The computer system as set forth in claim **9**, wherein the computer system further includes a memory controller coupled to the processor, the main memory, the secured memory unit and the FLASH memory for controlling memory access;

a FLASH security circuit coupled to the memory controller and the FLASH memory for qualifying a write signal provided by the memory controller to the FLASH memory for the write access, and for generating an interrupt to place the processor in the system management mode.

11. The computer system as set forth in claim **10**, wherein the computer system further includes an I/O port coupled to the processor and the FLASH security circuit for notifying the FLASH security circuit of the write access.

12. The computer system as set forth in claim **7**, wherein the plurality of authentication functions further include a copy function for conditionally copying the write data of the write access into the non-volatile storage if the digest comparison function successfully compares the first and second digests.

13. A computer system motherboard comprising:

(a) a non-volatile memory storage unit; and

(b) system basic input/output services (BIOS) including a plurality of associated authentication functions stored in the non-volatile memory storage unit, wherein the plurality of associated authentication functions are read from the non-volatile storage and retained in secured memory upon initialization of a computer system integrated with the computer system motherboard, the plurality of associated authentication functions operative to authenticate write accesses to update the system BIOS using an electronic signature associated with the system BIOS updates, the content of the electronic signature being functionally dependent on the content of the system BIOS updates.

14. The computer system motherboard as set forth in claim **13**, wherein the computer system motherboard further includes

(c) main memory for staging the system BIOS updates in a buffer.

15. The computer system motherboard as set forth in claim **14**, wherein the computer system motherboard further includes

(d) system management memory for storing and securing the plurality of associated authentication functions during operation of the computer system, the plurality of associated authentication functions being copied into the system management memory during system initialization, wherein the system management memory is not mapped into a normal system memory space of the computer system unless the computer system is executing in a system management mode, and wherein the system management memory is write protected except for system initialization and system execution mode transition.

16. The computer system motherboard as set forth in claim **15**, wherein the computer system motherboard further comprises

(e) a processor coupled to the non-volatile memory storage and the system management memory for invoking the authentication functions during operation of the computer system in system management mode to authenticate the system BIOS updates, and to allow only authenticated system BIOS updates to be written from the buffer of main memory into the non-volatile memory storage unit.

17. The computer system motherboard as set forth in claim **16**, wherein the computer system motherboard further comprises:

(f) a memory controller coupled to the processor, the main memory, the system management memory and the non-volatile memory storage unit for controlling memory access;

(g) a non-volatile memory access security circuit coupled to the memory controller and the non-volatile memory storage unit for qualifying a write signal provided by the memory controller to the non-volatile memory storage unit for a write access initiated to write the system BIOS updates into the non-volatile memory storage unit, and for generating an interrupt to place the computer system in the system management mode.

18. The computer system motherboard as set forth in claim **17**, wherein the computer system motherboard further includes an I/O port coupled to the processor and the non-volatile memory access security circuit for notifying the nonvolatile memory security circuit of the write access.

19. The computer system motherboard as set forth in claim **13**, wherein the plurality of authentication functions include

a decryption function for reconstituting a first digest by decrypting the electronic signature with a public key, the electronic signature being generated by encrypting the first digest with a secret private key in a complementary manner,

a message digest function for generating a second digest based on the content of the system BIOS updates in the same manner the first digest was generated, and

a digest comparison function for determining whether the system BIOS updates are authentic by comparing the first and second digests.

20. The computer system as set forth in claim **19**, wherein the plurality of authentication functions further include a copy function for conditionally copying the system BIOS updates into the non-volatile memory storage unit if the digest comparison function successfully compares the first and second digests.

21. The computer implemented method of claim **4**, wherein unsecuring the secured associated authentication functions (b.1) comprises issuing a system management interrupt (SMI) placing the computer system into SMM, wherein the system management memory is mapped to the normal memory space from which the associated authentication functions are selectively invoked to authenticate received data.

22. The computer system of claim **6**, wherein the processor enters system management mode upon the receipt of a system management interrupt (SMI), whereafter, the processor selectively invokes the associated authentication functions to authenticate received data.

23. The computer system motherboard of claim **15**, wherein the computer system enters system management mode upon receipt of a system management interrupt (SMI), whereafter, the computer system selectively invokes the associated authentication functions to authenticate received data.

* * * * *

Nintendo - Ancora Exh. 1003