US 20160006714A1

(19) **United States**

(12) **Patent Application Publication**

Grigorovitch et al.

(10) **Pub. No.: US 2016/0006714 A1**

(43) **Pub. Date: Jan. 7, 2016**

(54) **PROTECTED MEDIA PIPELINE**

(71) Applicant: **MICROSOFT TECHNOLOGY LICENSING, LLC**, Redmond, WA (US)

(72) Inventors: **Alexandre Grigorovitch**, Redmond, WA (US); **Chadd Knowlton**, Bellevue, WA (US); **Kirt Debique**, Seattle, WA (US); **James Alkove**, Woodinville, WA (US); **Geoffrey T. Dunbar**, Kirkland, WA (US); **Sumedh N. Barde**, Redmond, WA (US)

(21) Appl. No.: **14/852,520**

(22) Filed: **Sep. 12, 2015**

Related U.S. Application Data

(63) Continuation of application No. 11/116,689, filed on Apr. 27, 2005.

(60) Provisional application No. 60/673,979, filed on Apr. 22, 2005.

Publication Classification

(51) **Int. Cl.**

H04L 29/06 (2006.01)

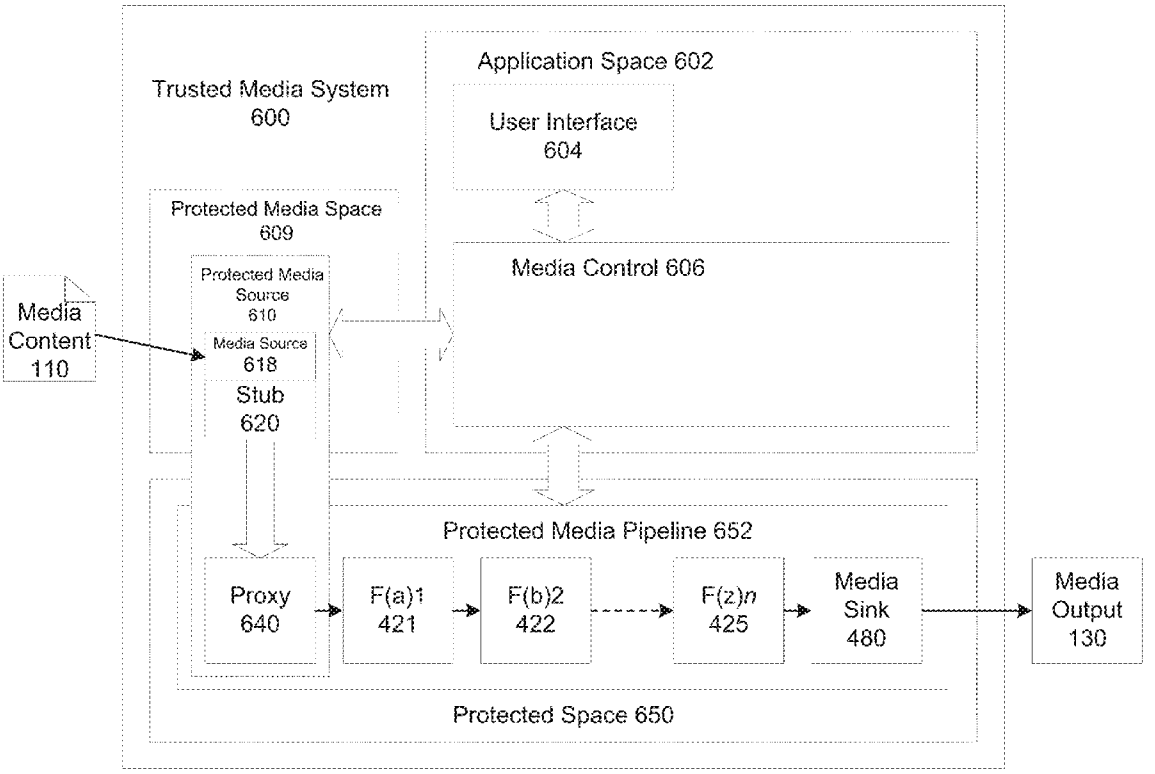
G06F 21/10 (2006.01)

(52) **U.S. Cl.**

CPC ***H04L 63/08*** (2013.01); ***H04L 63/10*** (2013.01); ***G06F 21/10*** (2013.01)

(57) **ABSTRACT**

A system for processing a media content comprising an application space, a media control mechanism operating in the application space, the media control mechanism controlling the operation of the system, a user interface adapted to provide input to the media control mechanism, a protected space distinct from the application space, and a protected media pipeline operating in the protected space, the protected media pipeline coupled to the media control mechanism, the protected media pipeline adapted to access the media content, process the media content, and output the media content.



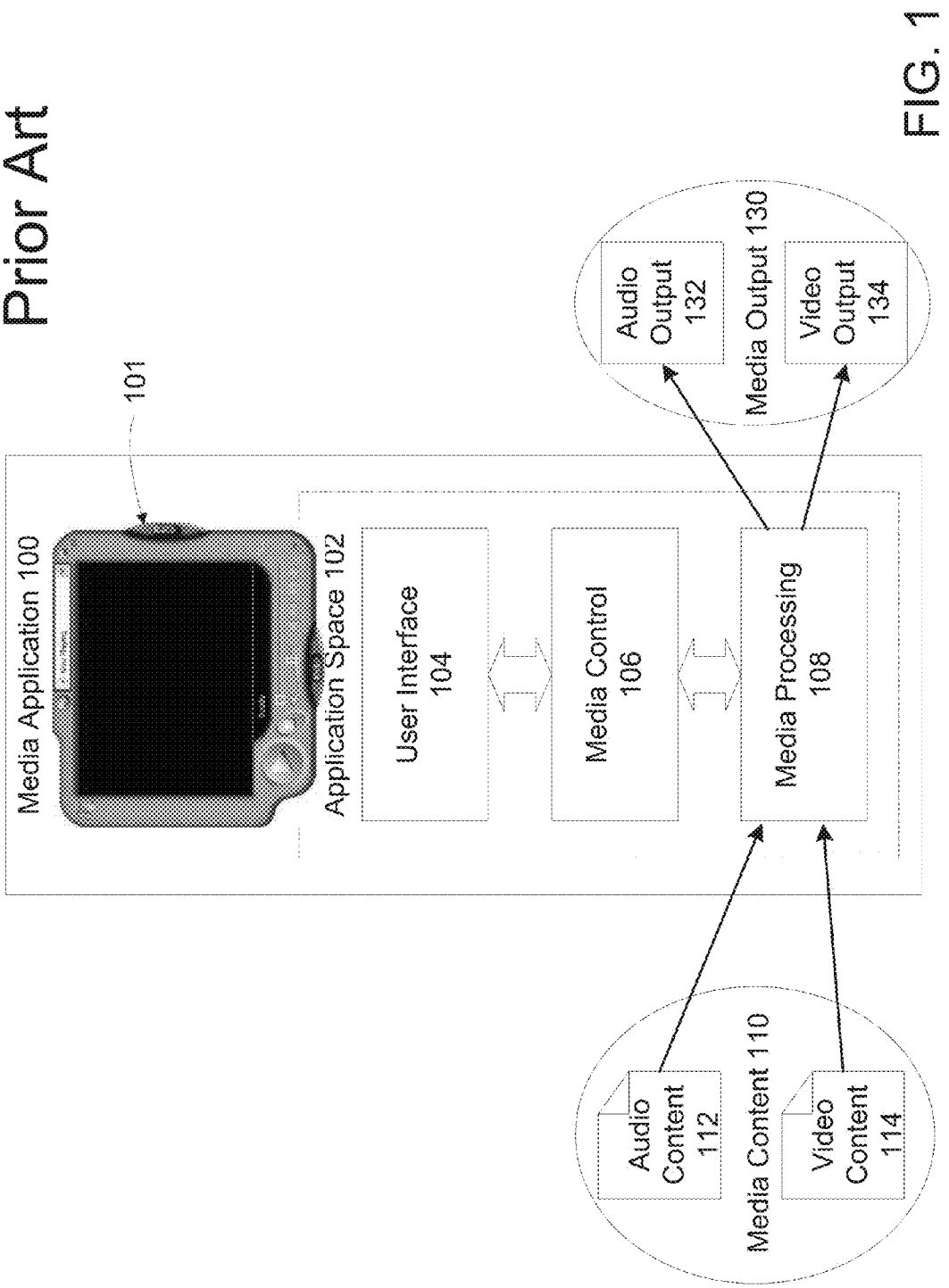

Prior Art
101
Media Application 100
Application Space 102
User Interface 104
Media Control 106
Media Processing 108
Media Content 110
Audio Content 112
Video Content 114
Media Output 130
Audio Output 132
Video Output 134

FIG. 1

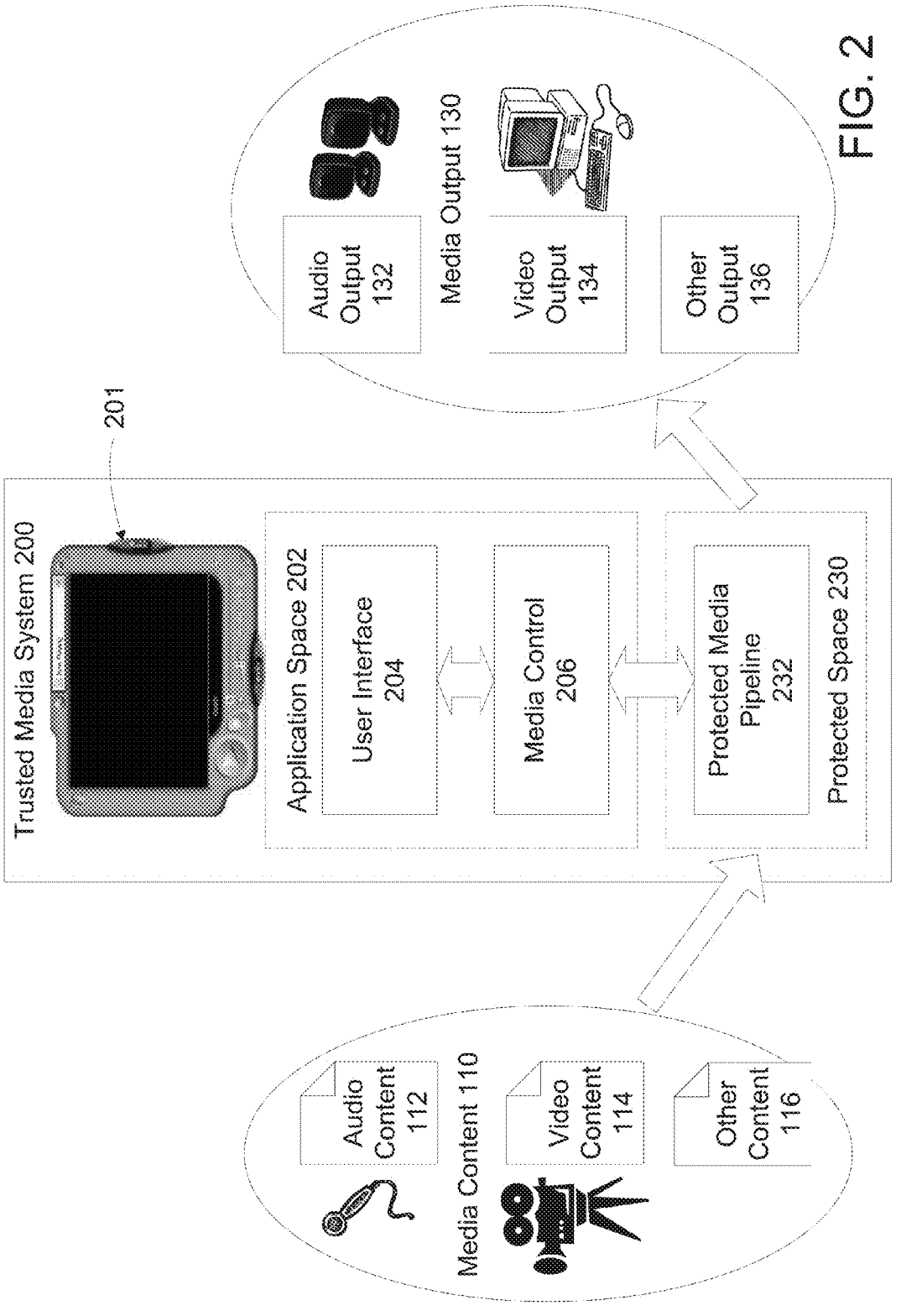

Media Content 110
Audio Content 112
Video Content 114
Other Content 116
Trusted Media System 200
201
Application Space 202
User Interface 204
Media Control 206
Protected Media Pipeline 232
Protected Space 230
Media Output 130
Audio Output 132
Video Output 134
Other Output 136

FIG. 2

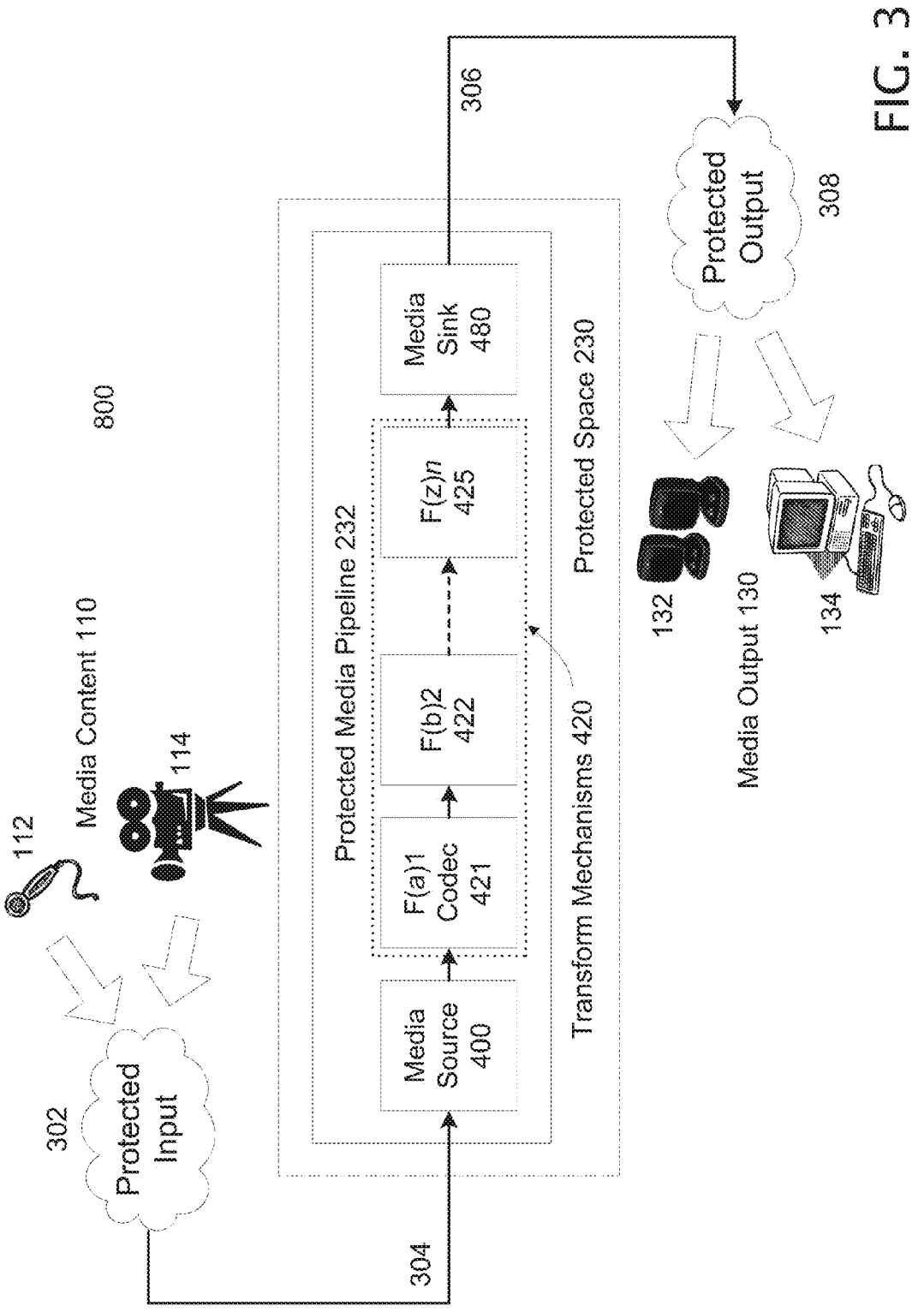

800
Media Content 110
112
114
302
Protected Input
304
Protected Media Pipeline 232
Media Source 400
F(a)1 Codec 421
F(b)2 422
F(z)n 425
Media Sink 480
306
Protected Space 230
Transform Mechanisms 420
Protected Output
308
Media Output 130
132
134

FIG. 3

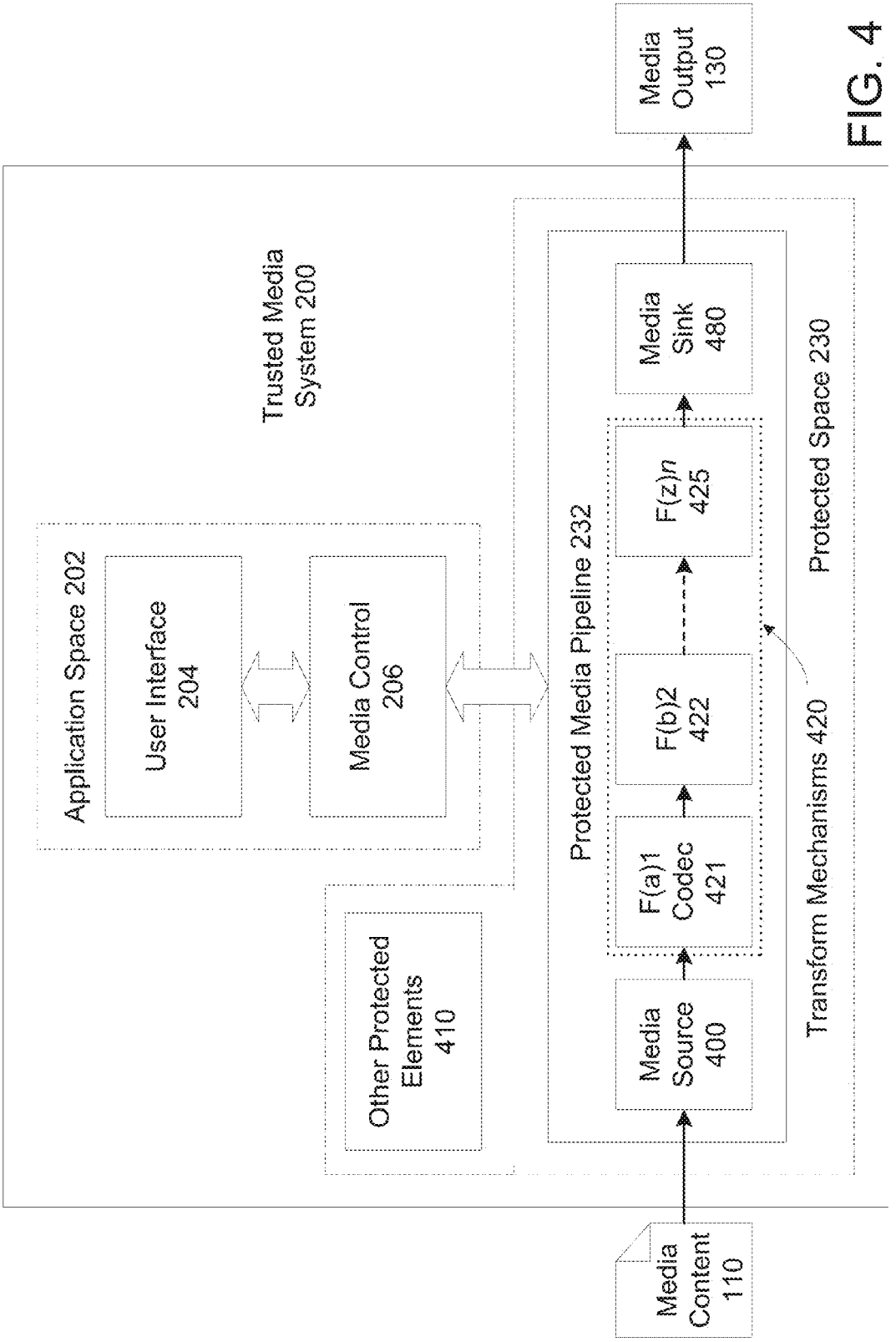

Media Content 110
Media Source 400
F(a)1 Codec 421
F(b)2 422
F(z)n 425
Media Sink 480
Media Output 130
Protected Media Pipeline 232
Transform Mechanisms 420
Protected Space 230
Other Protected Elements 410
Application Space 202
User Interface 204
Media Control 206
Trusted Media System 200

FIG. 4

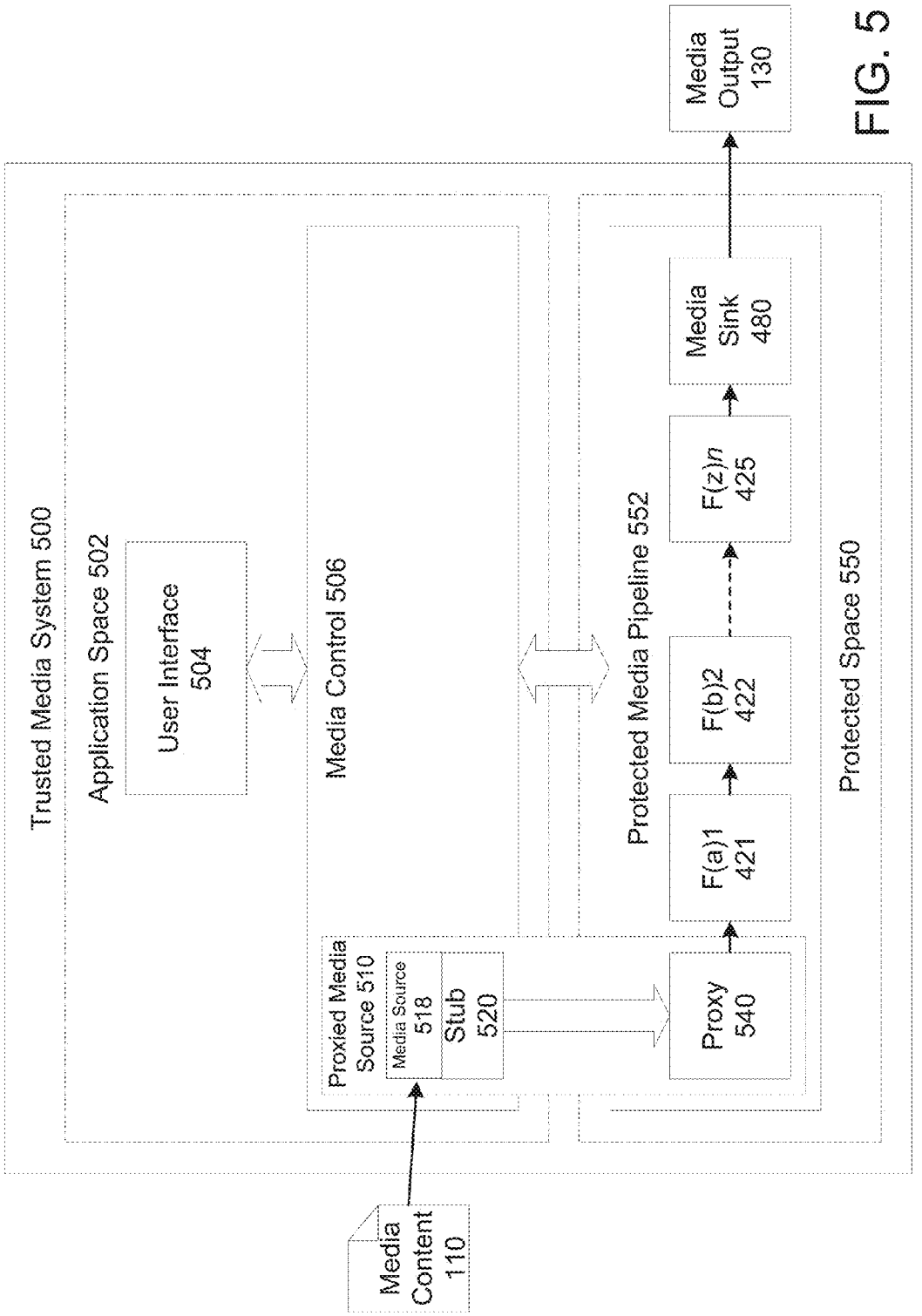

Trusted Media System 500
Application Space 502
User Interface 504
Media Control 506
Proxied Media Source 510
Media Source 518
Stub 520
Protected Media Pipeline 552
Proxy 540
F(a)1 421
F(b)2 422
F(z)n 425
Media Sink 480
Protected Space 550
Media Output 130
Media Content 110

FIG. 5

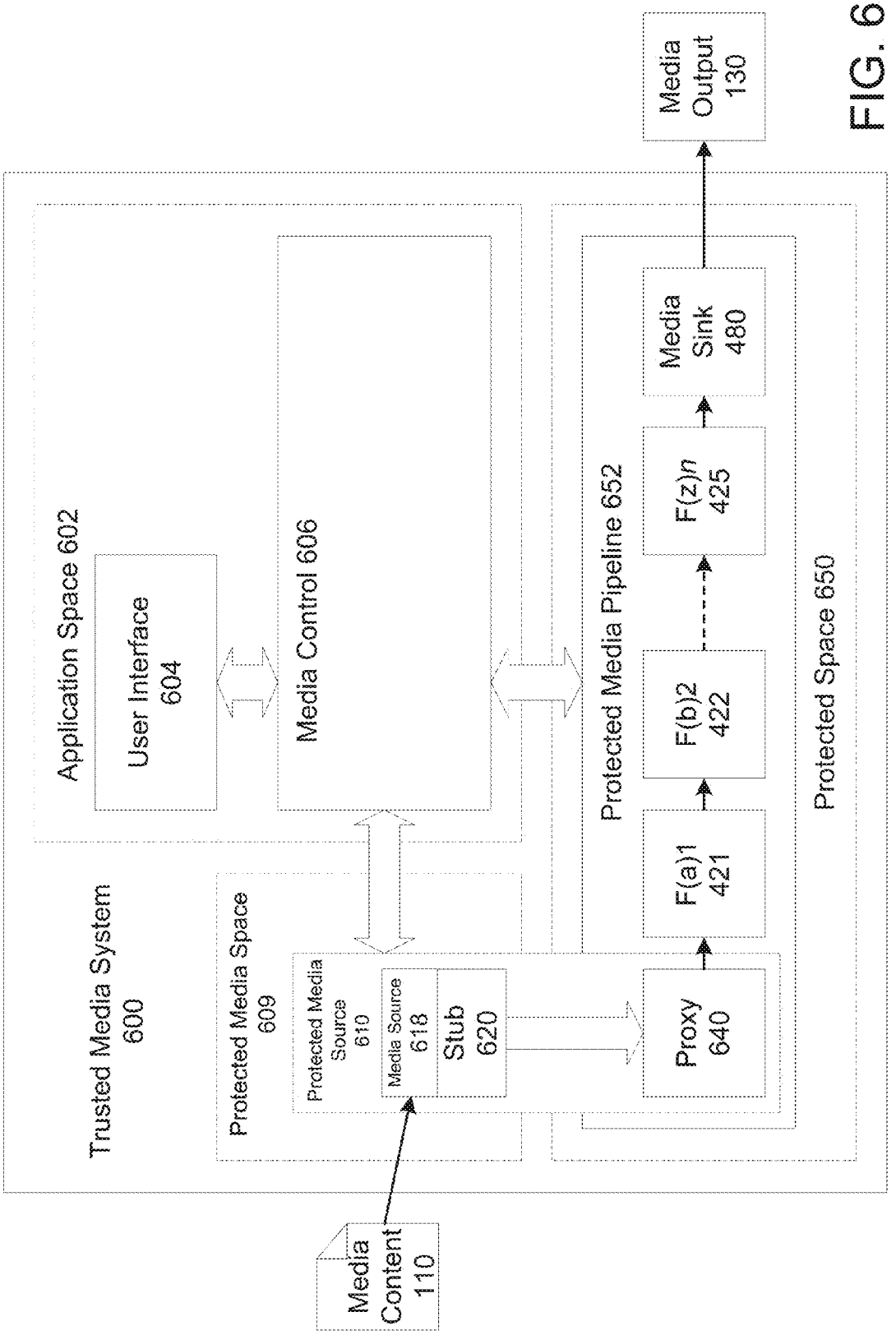

Media Content 110
Trusted Media System 600
Application Space 602
User Interface 604
Media Control 606
Protected Media Space 609
Protected Media Source 610
Media Source 618
Stub 620
Protected Space 650
Protected Media Pipeline 652
Proxy 640
F(a)1 421
F(b)2 422
F(z)n 425
Media Sink 480
Media Output 130

FIG. 6

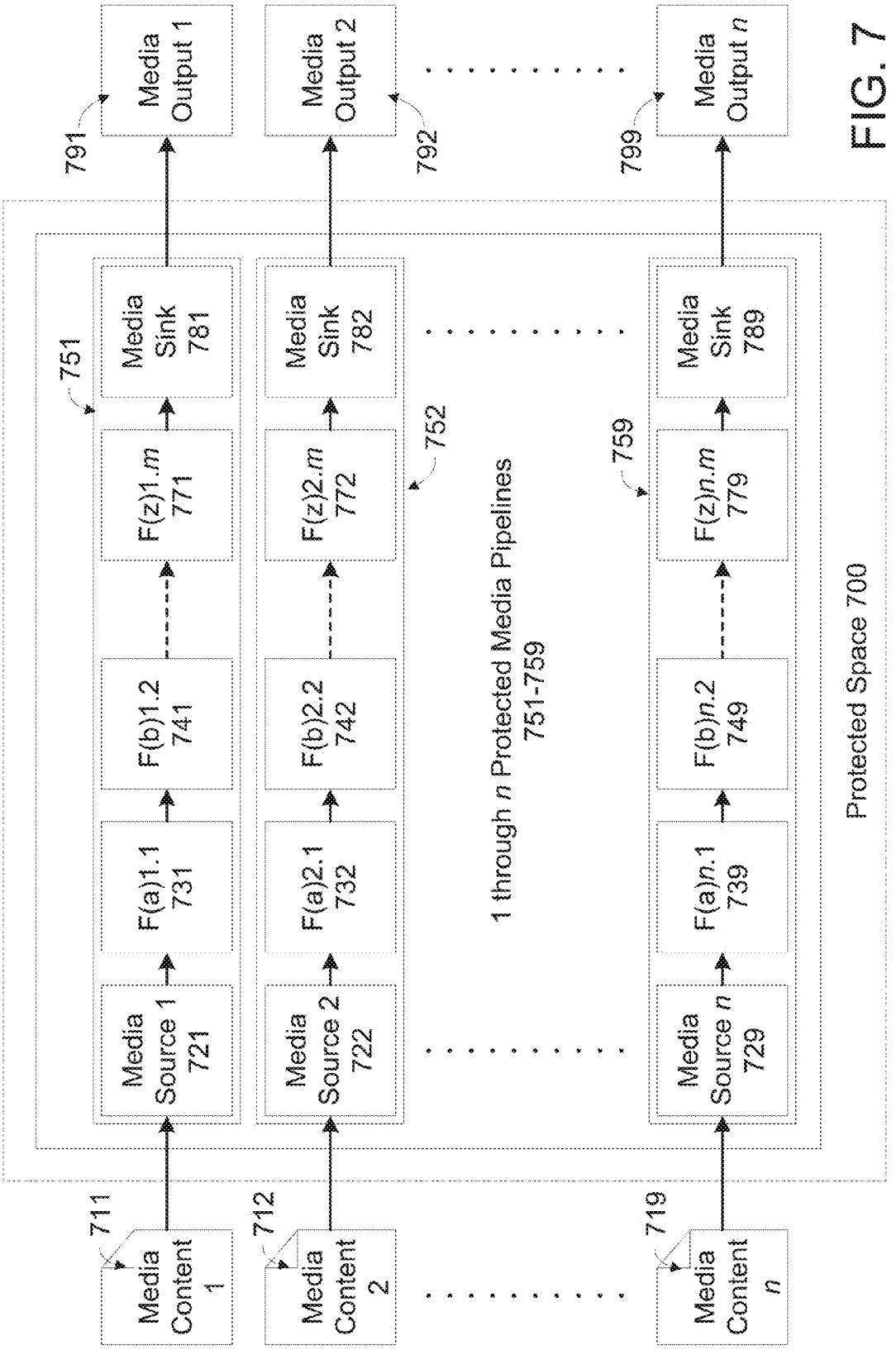

Media Content 1
711
Media Content 2
712
Media Content n
719
Media Source 1 721
Media Source 2 722
Media Source n 729
F(a)1.1 731
F(a)2.1 732
F(a)n.1 739
F(b)1.2 741
F(b)2.2 742
F(b)n.2 749
F(z)1.m 771
F(z)2.m 772
F(z)n.m 779
Media Sink 781
Media Sink 782
Media Sink 789
751
752
759
1 through n Protected Media Pipelines 751-759
Protected Space 700
Media Output 1
Media Output 2
Media Output n
791
792
799

FIG. 7

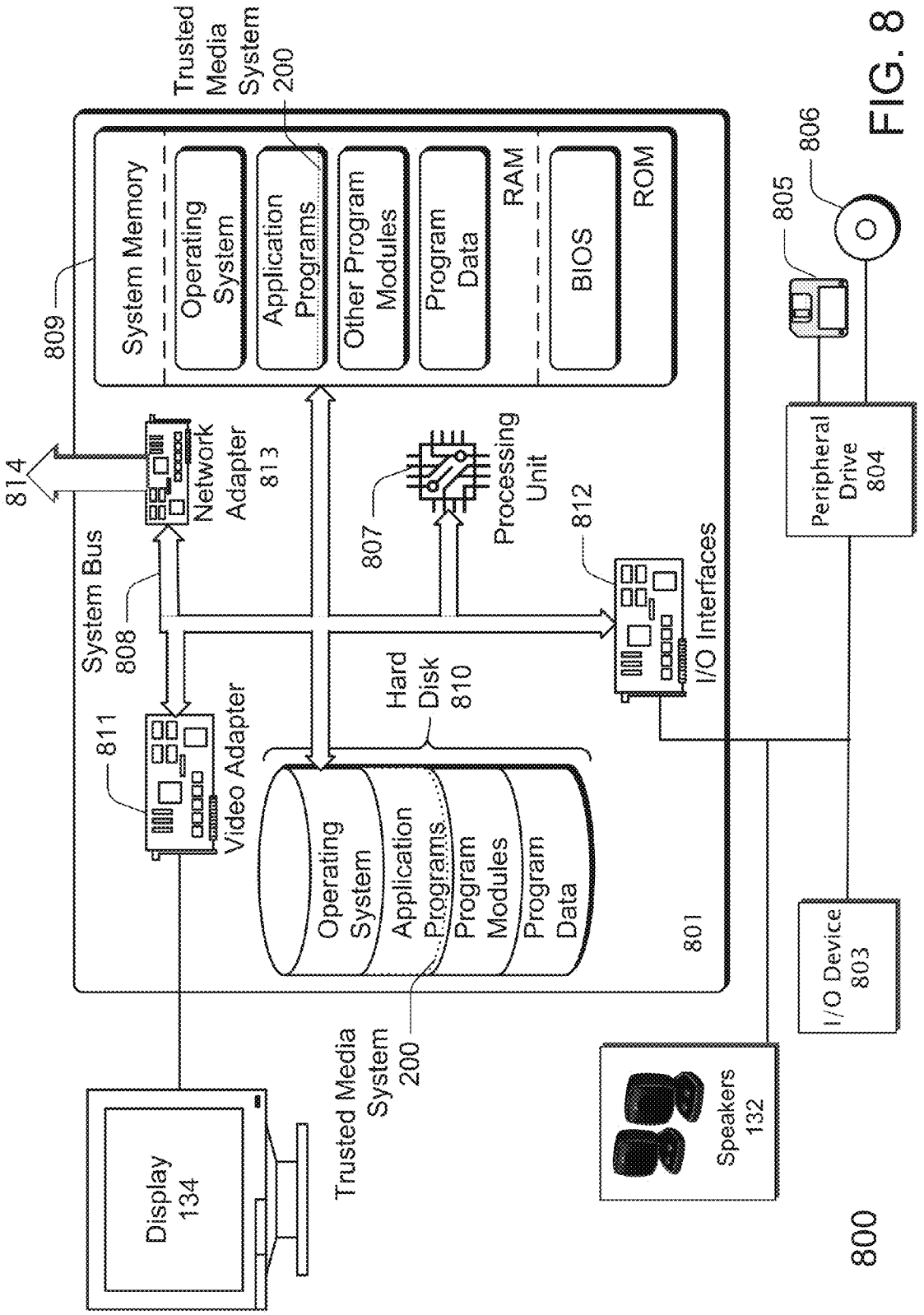
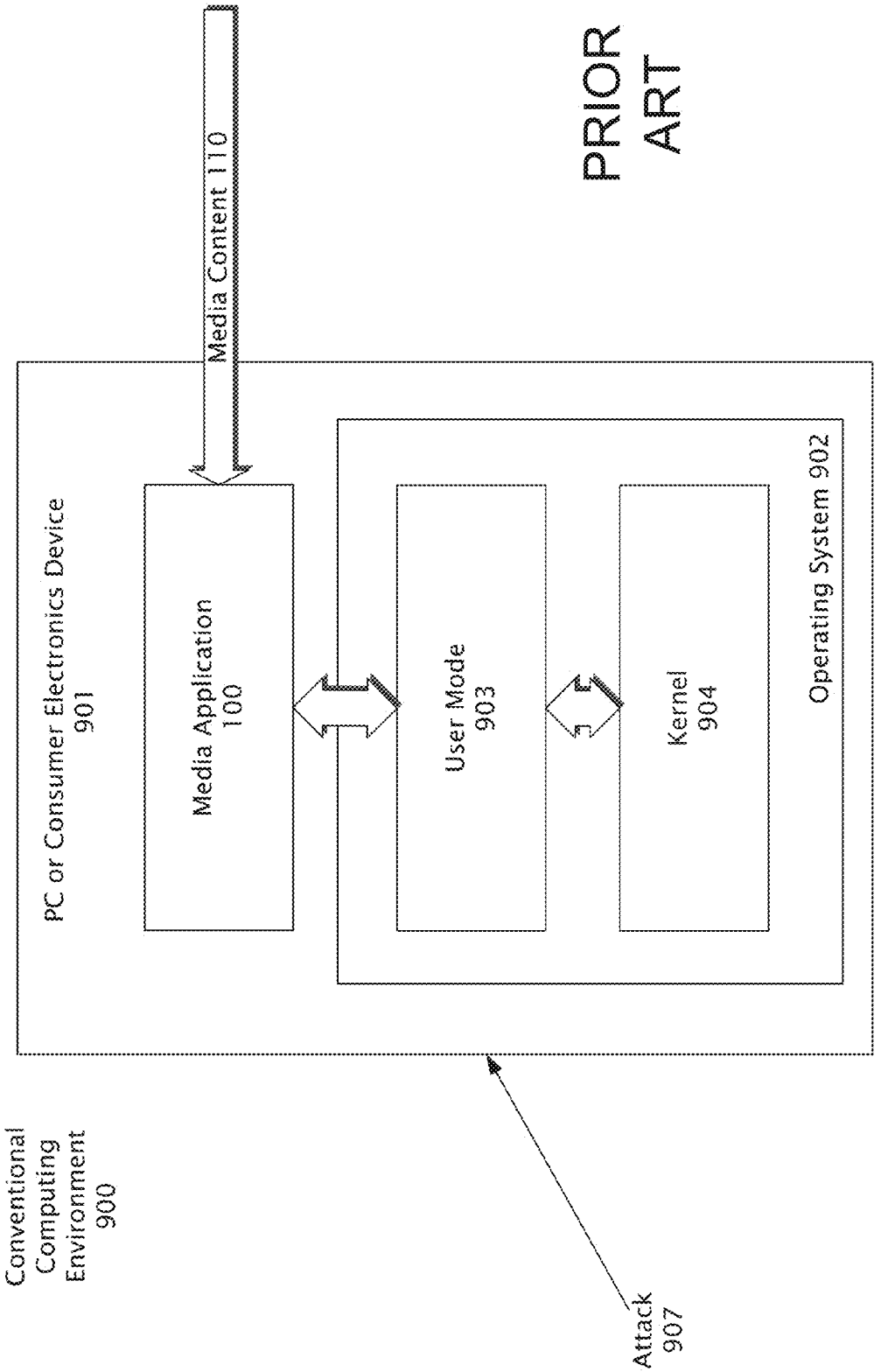

Trusted Media System 200
System Memory
Operating System
Application Programs
Other Program Modules
Program Data
RAM
BIOS
ROM
809
814
Network Adapter 813
System Bus 808
807
Processing Unit 812
I/O Interfaces
Peripheral Drive 804
805
806
811
Video Adapter
Hard Disk 810
Operating System
Application Programs
Program Modules
Program Data
801
Trusted Media System 200
Display 134
Speakers 132
I/O Device 803
800

FIG. 8



PRIOR ART

FIG. 9

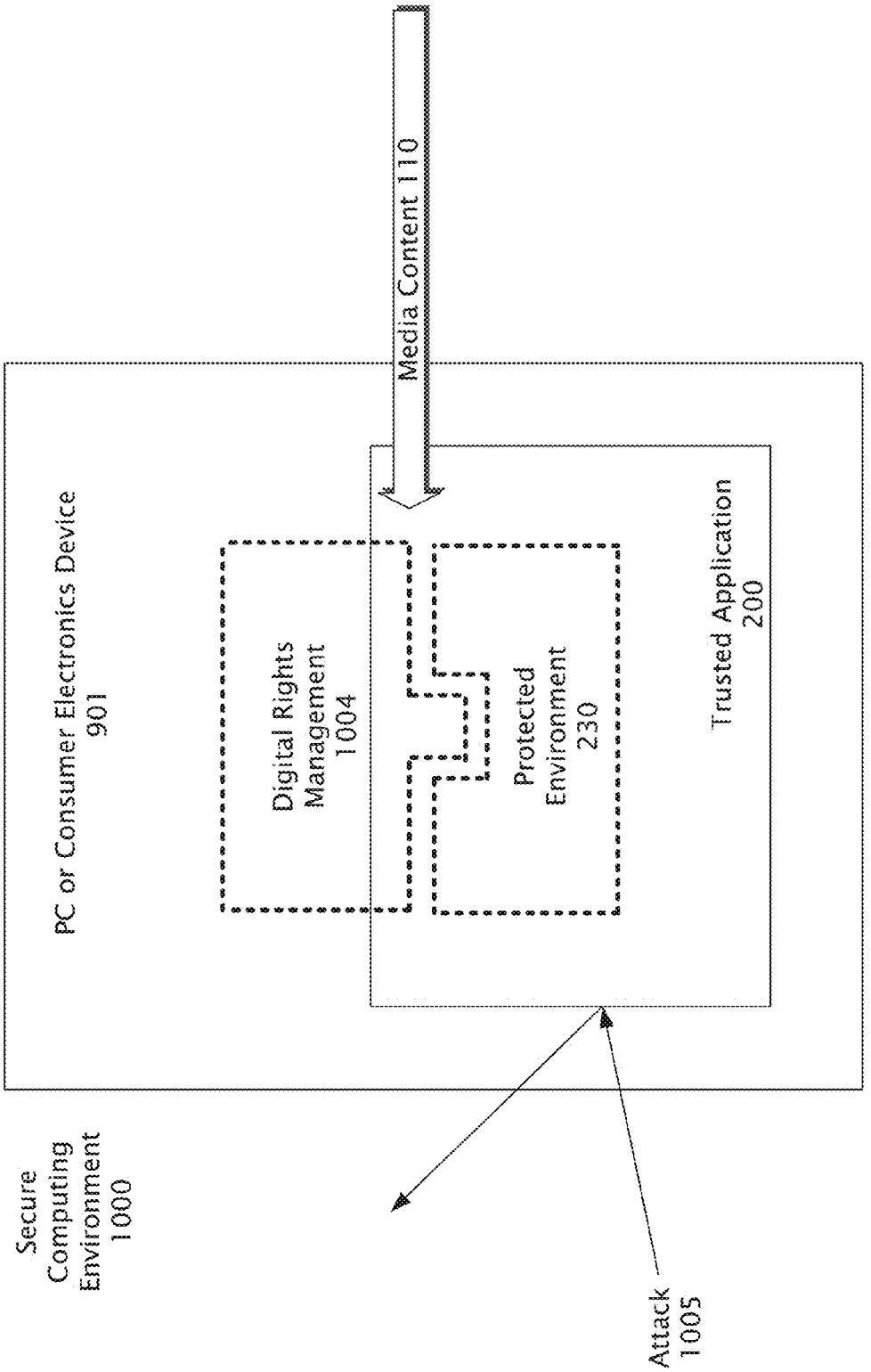

Media Content 110
PC or Consumer Electronics Device
901
Digital Rights
Management
1004
Protected
Environment
230
Trusted Application
200
Secure
Computing
Environment
1000
Attack
1005

FIG. 10

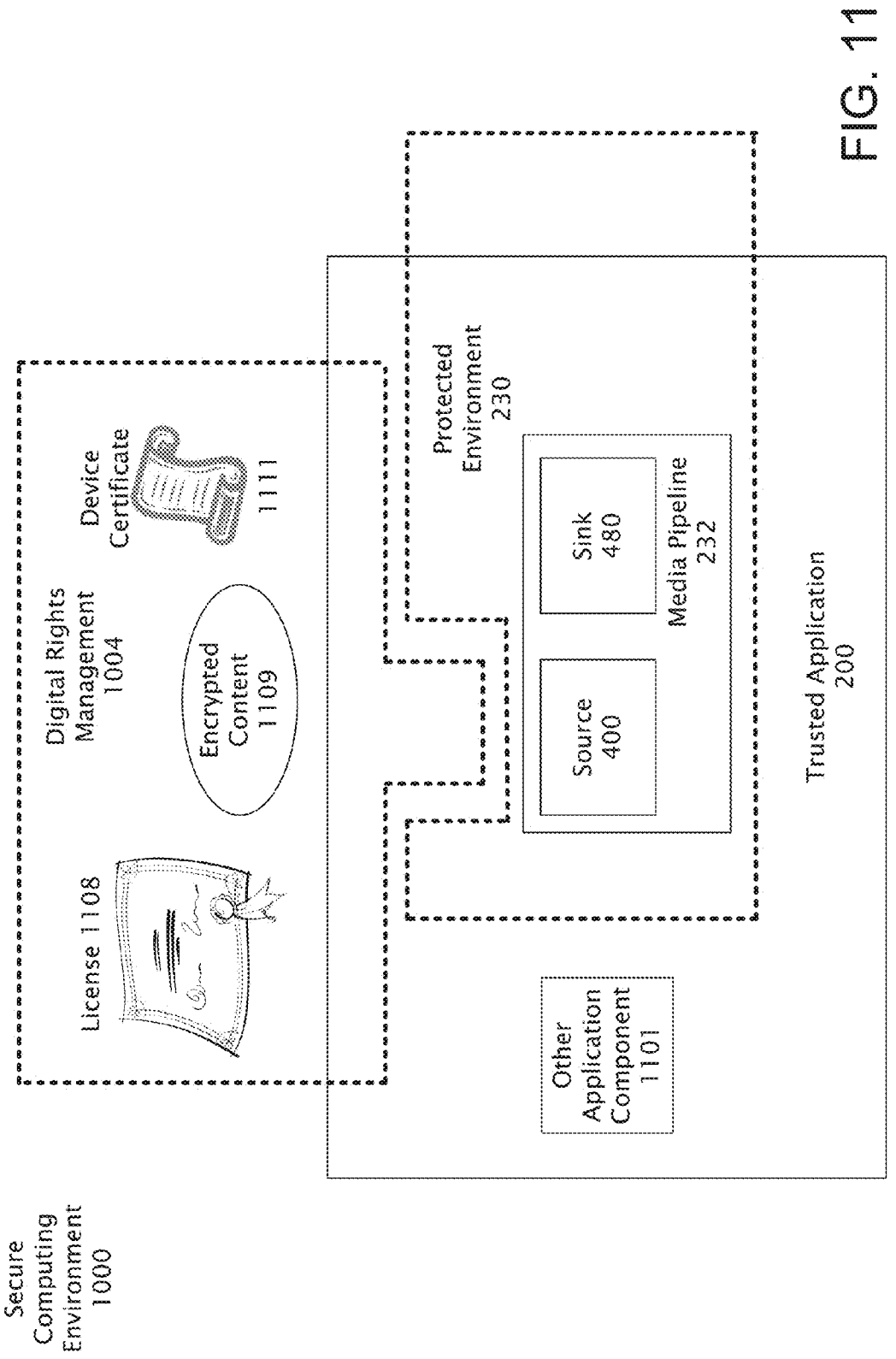

Secure Computing Environment 1000
Trusted Application 200
Protected Environment 230
Media Pipeline 232
Source 400
Sink 480
Other Application Component 1101
License 1108
Digital Rights Management 1004
Encrypted Content 1109
Device Certificate 1111

FIG. 11

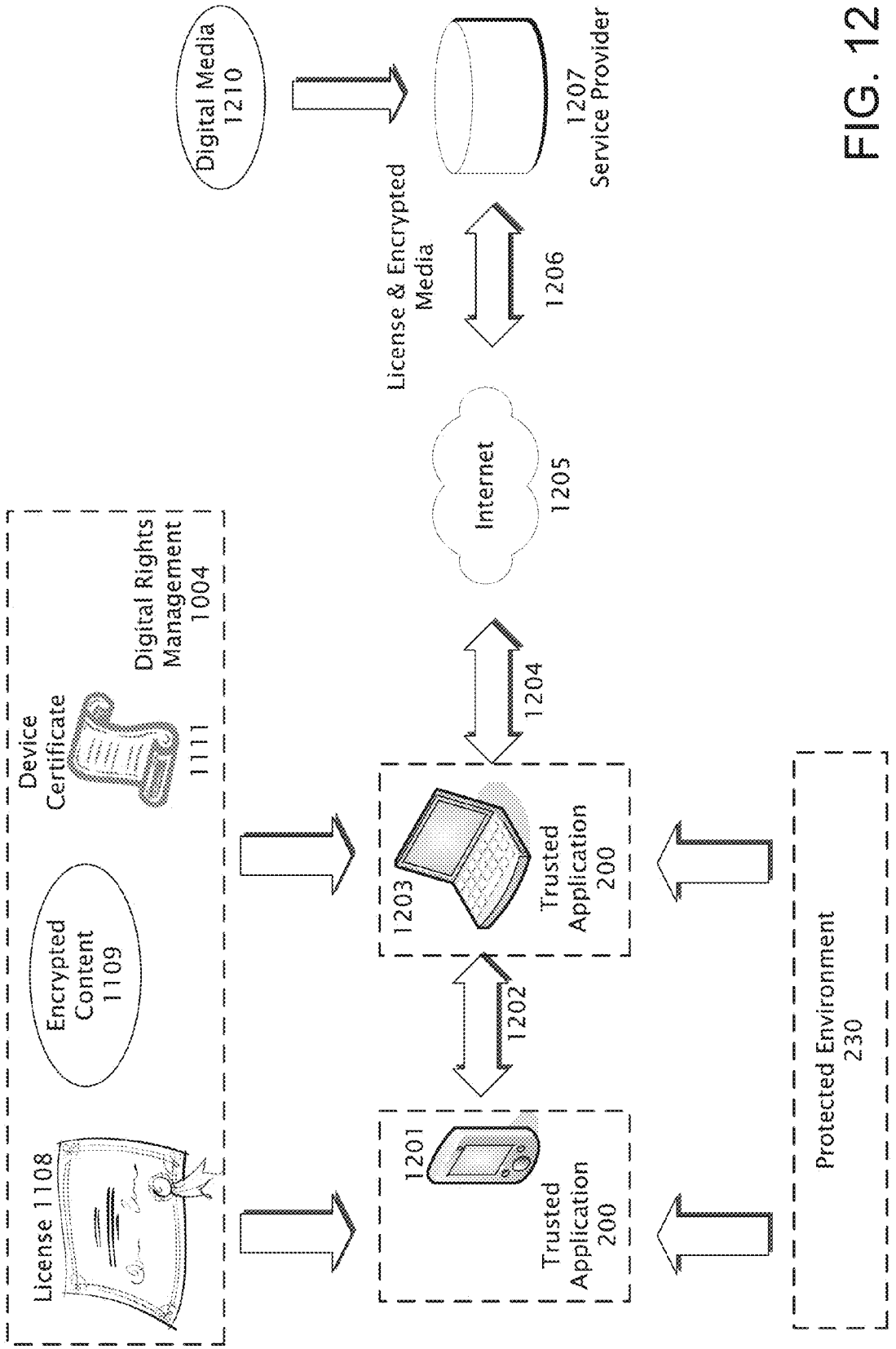

Digital Media
1210
1207
Service Provider
License & Encrypted
Media
1206
Internet
1205
1204
1203
Trusted
Application
200
1202
1201
Trusted
Application
200
Protected Environment
230
License 1108
Encrypted
Content
1109
Device
Certificate
1111
Digital Rights
Management
1004

FIG. 12

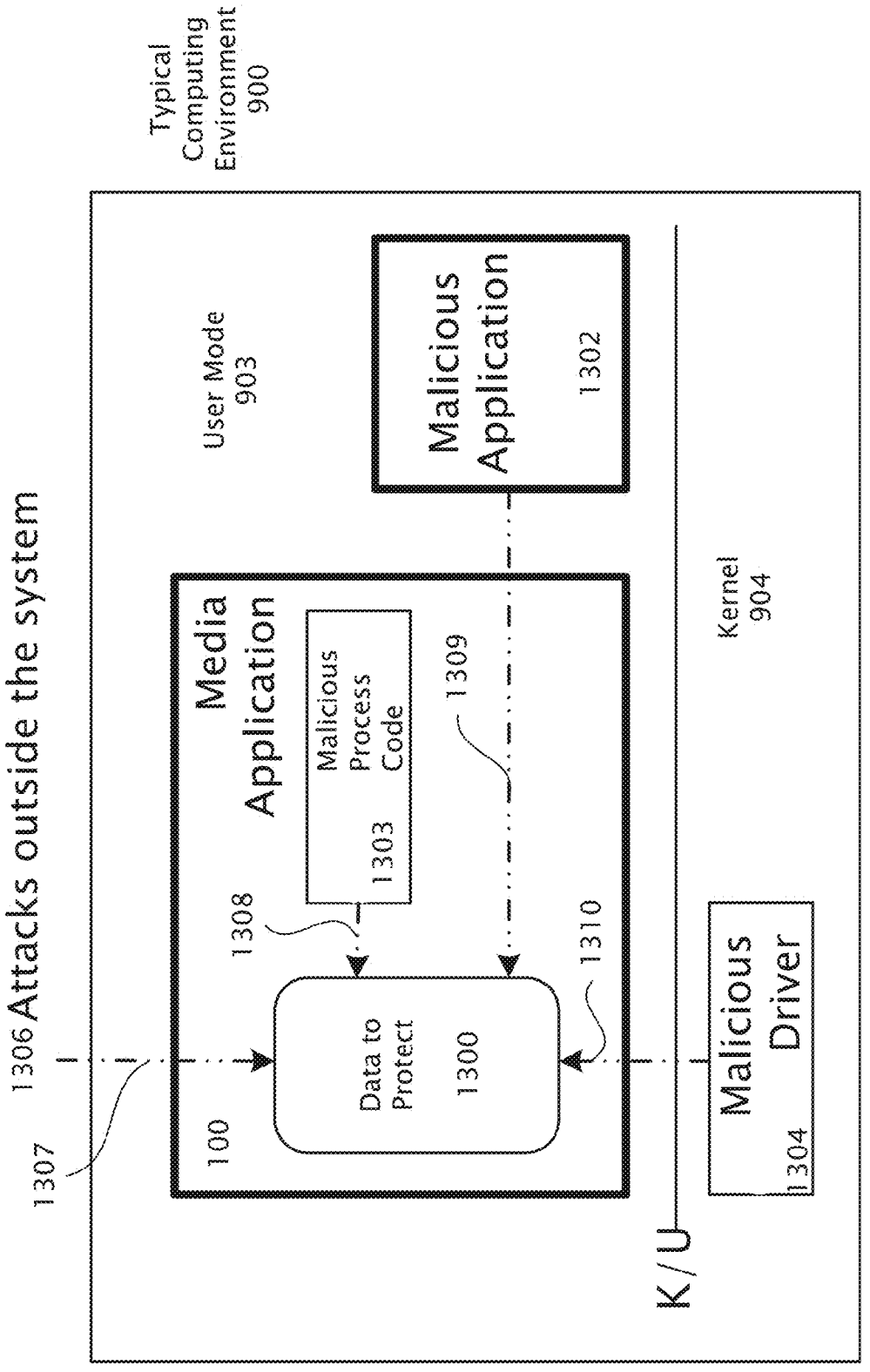

Typical Computing Environment 900
User Mode 903
Kernel 904
Attacks outside the system
1306
1307
Media Application
100
Malicious Process Code
1303
1308
1309
1310
Data to Protect
1300
Malicious Application
1302
Malicious Driver
1304
K/U

FIG. 13

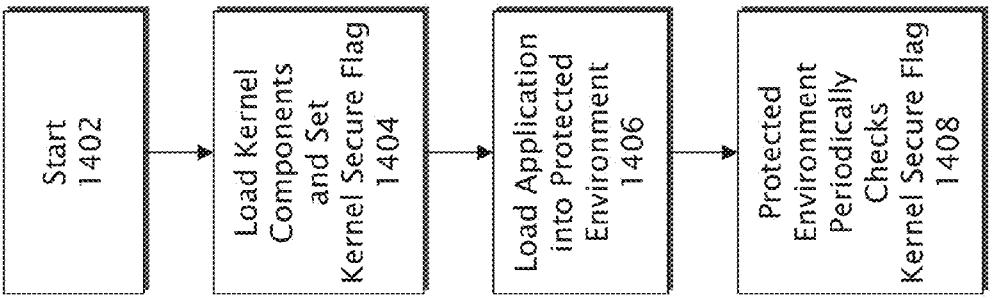

Start
1402
Load Kernel Components and Set Kernel Secure Flag
1404
Load Application into Protected Environment
1406
Protected Environment Periodically Checks Kernel Secure Flag
1408
Creation and Maintenance of Protected Environment for Trusted Application
1400

FIG. 14

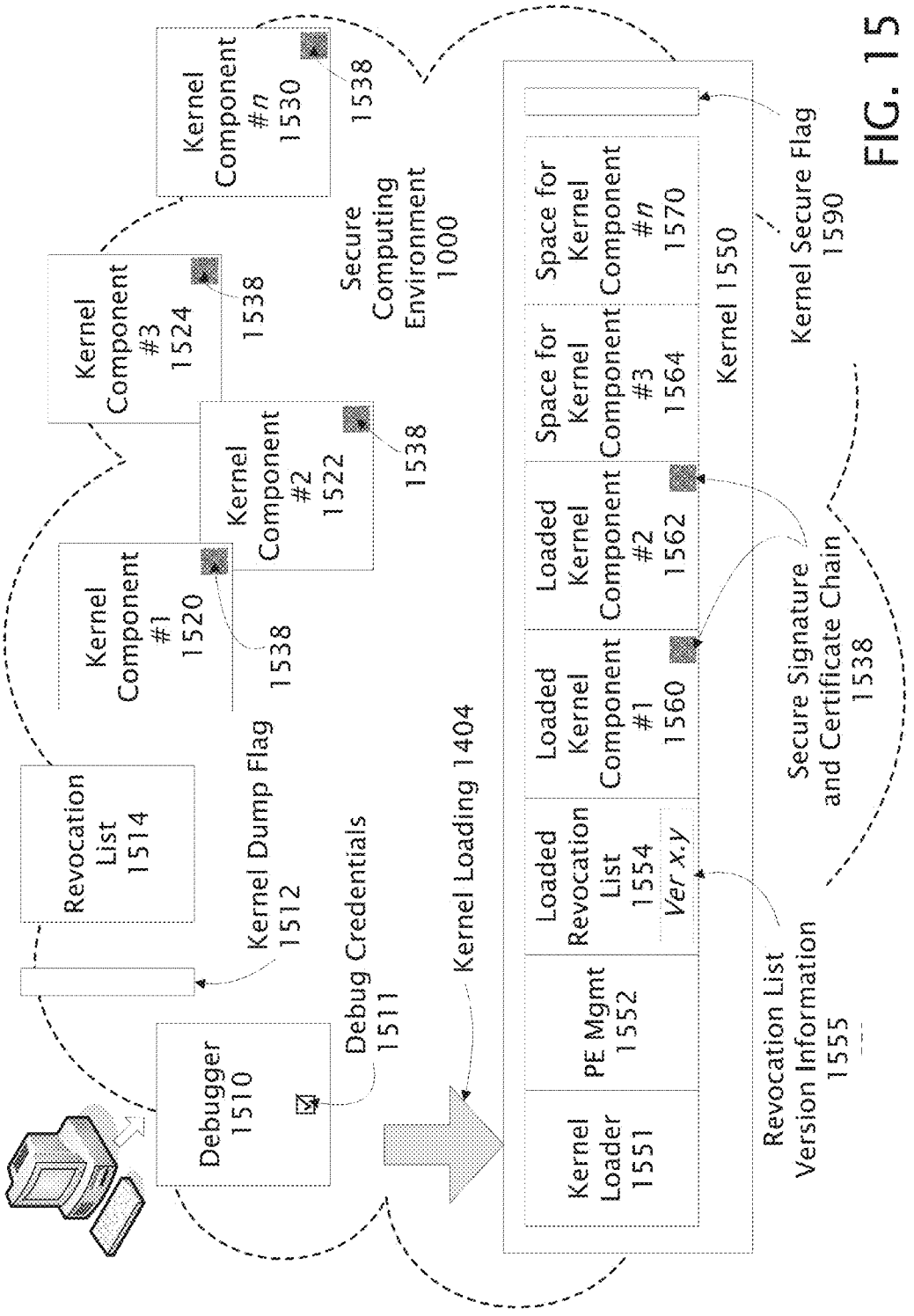

Secure Computing Environment 1000
Kernel Component #n 1530
1538
Kernel Component #3 1524
1538
Kernel Component #2 1522
1538
Kernel Component #1 1520
1538
Revocation List 1514
Kernel Dump Flag 1512
Debug Credentials 1511
Debugger 1510
Kernel Loading 1404
Kernel Loader 1551
PE Mgmt 1552
Loaded Revocation List 1554
Ver x.y
Loaded Kernel Component #1 1560
Loaded Kernel Component #2 1562
Space for Kernel Component #3 1564
Space for Kernel Component #n 1570
Kernel 1550
Kernel Secure Flag 1590
Secure Signature and Certificate Chain 1538
Revocation List Version Information 1555

FIG. 15

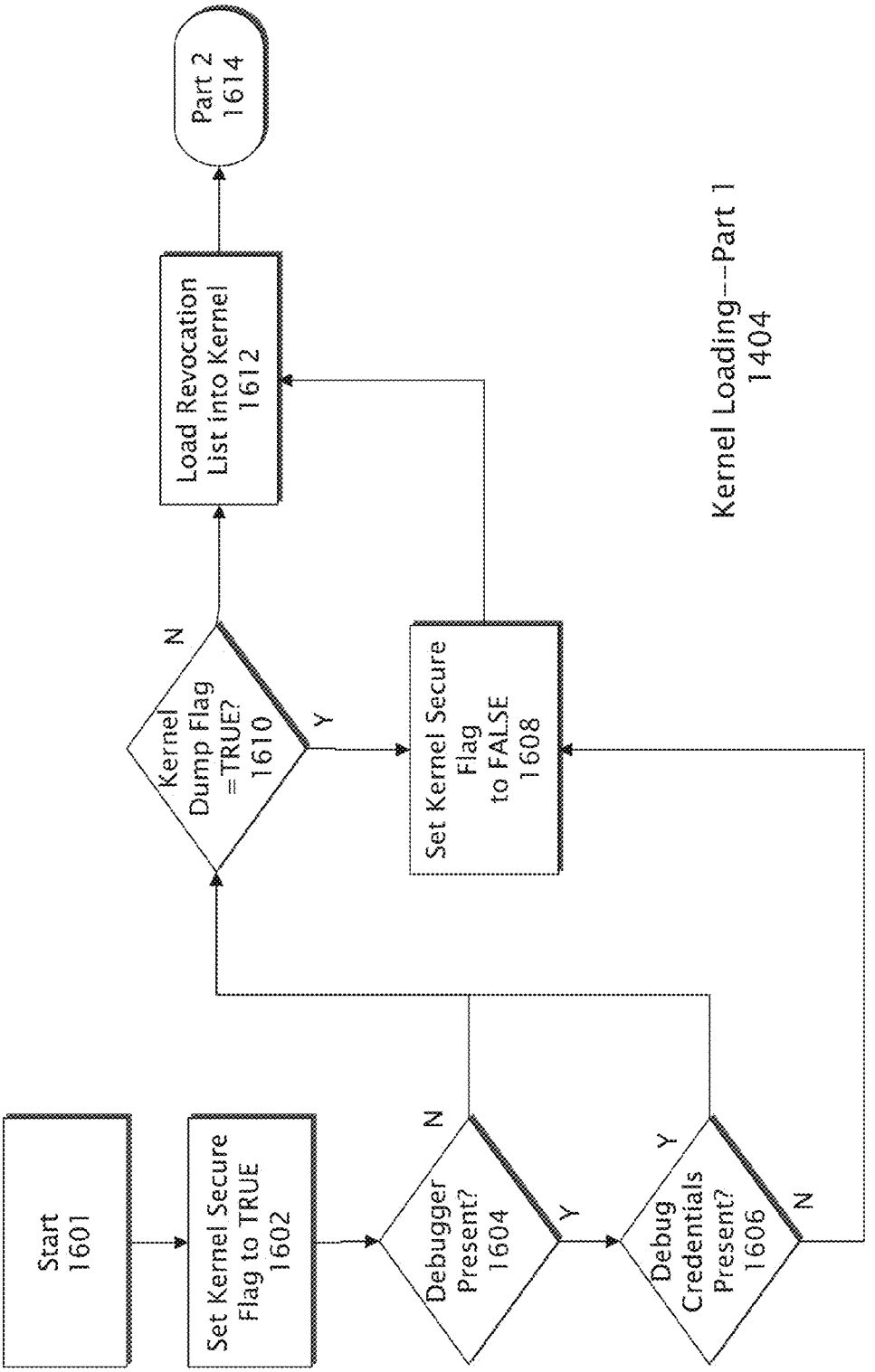

Kernel Loading--Part 1
1404
Start
1601
Set Kernel Secure Flag to TRUE
1602
Debugger Present?
1604
N
Y
Debug Credentials Present?
1606
Y
N
Kernel Dump Flag =TRUE?
1610
N
Y
Set Kernel Secure Flag to FALSE
1608
Load Revocation List into Kernel
1612
Part 2
1614

FIG. 16

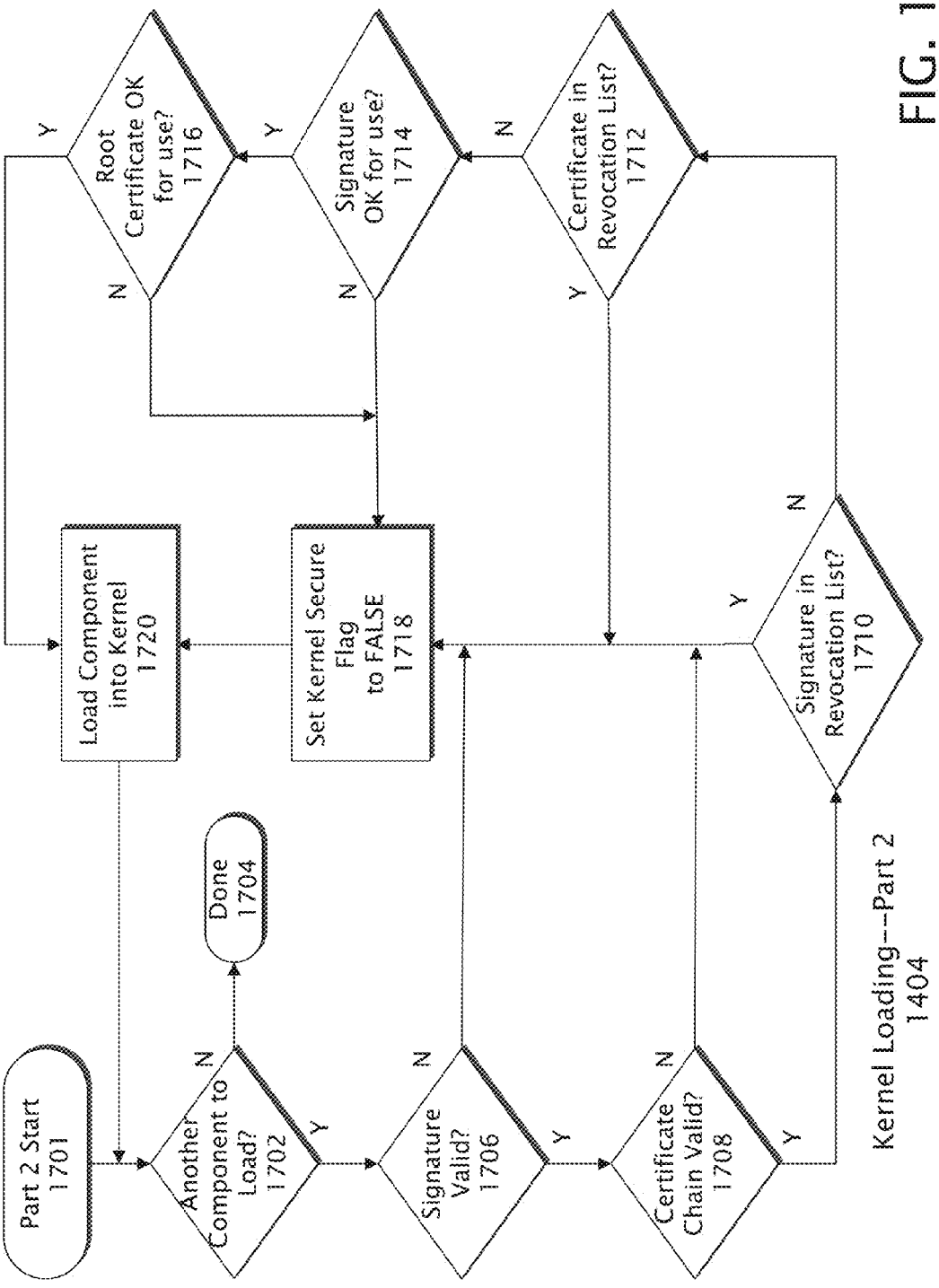

Part 2 Start 1701
Another Component to Load? 1702
Y
N
Done 1704
Signature Valid? 1706
Y
N
Certificate Chain Valid? 1708
Y
N
Signature in Revocation List? 1710
Y
N
Certificate in Revocation List? 1712
Y
N
Signature OK for use? 1714
Y
N
Root Certificate OK for use? 1716
Y
N
Set Kernel Secure Flag to FALSE 1718
Load Component into Kernel 1720
Kernel Loading--Part 2 1404

FIG. 17

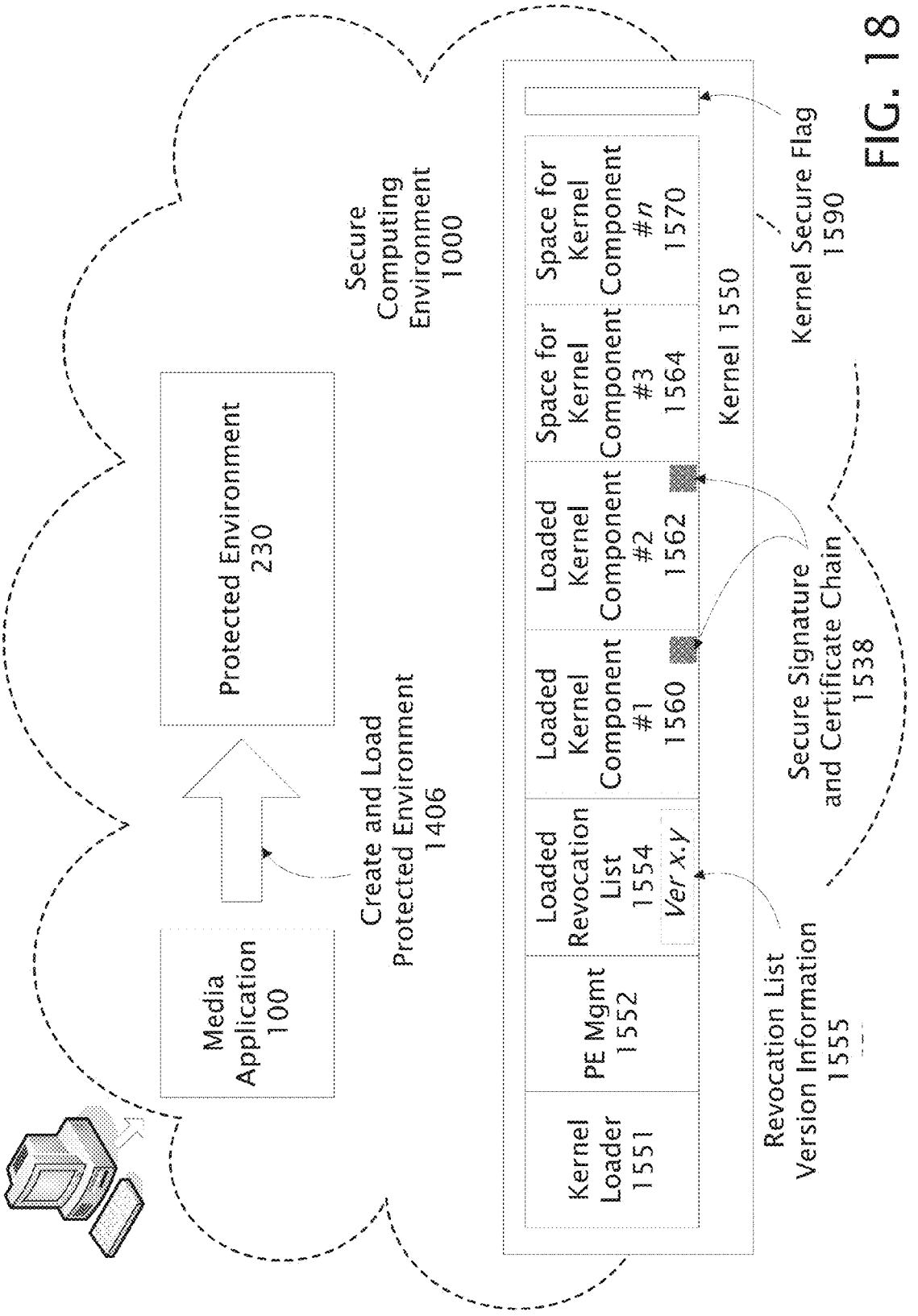

Secure Computing Environment 1000
Protected Environment 230
Media Application 100
Create and Load Protected Environment 1406
Kernel 1550
Kernel Loader 1551
PE Mgmt 1552
Loaded Revocation List 1554
Ver x.y
Loaded Kernel Component #1 1560
Loaded Kernel Component #2 1562
Space for Kernel Component #3 1564
Space for Kernel Component #n 1570
Kernel Secure Flag 1590
Secure Signature and Certificate Chain 1538
Revocation List Version Information 1555

FIG. 18

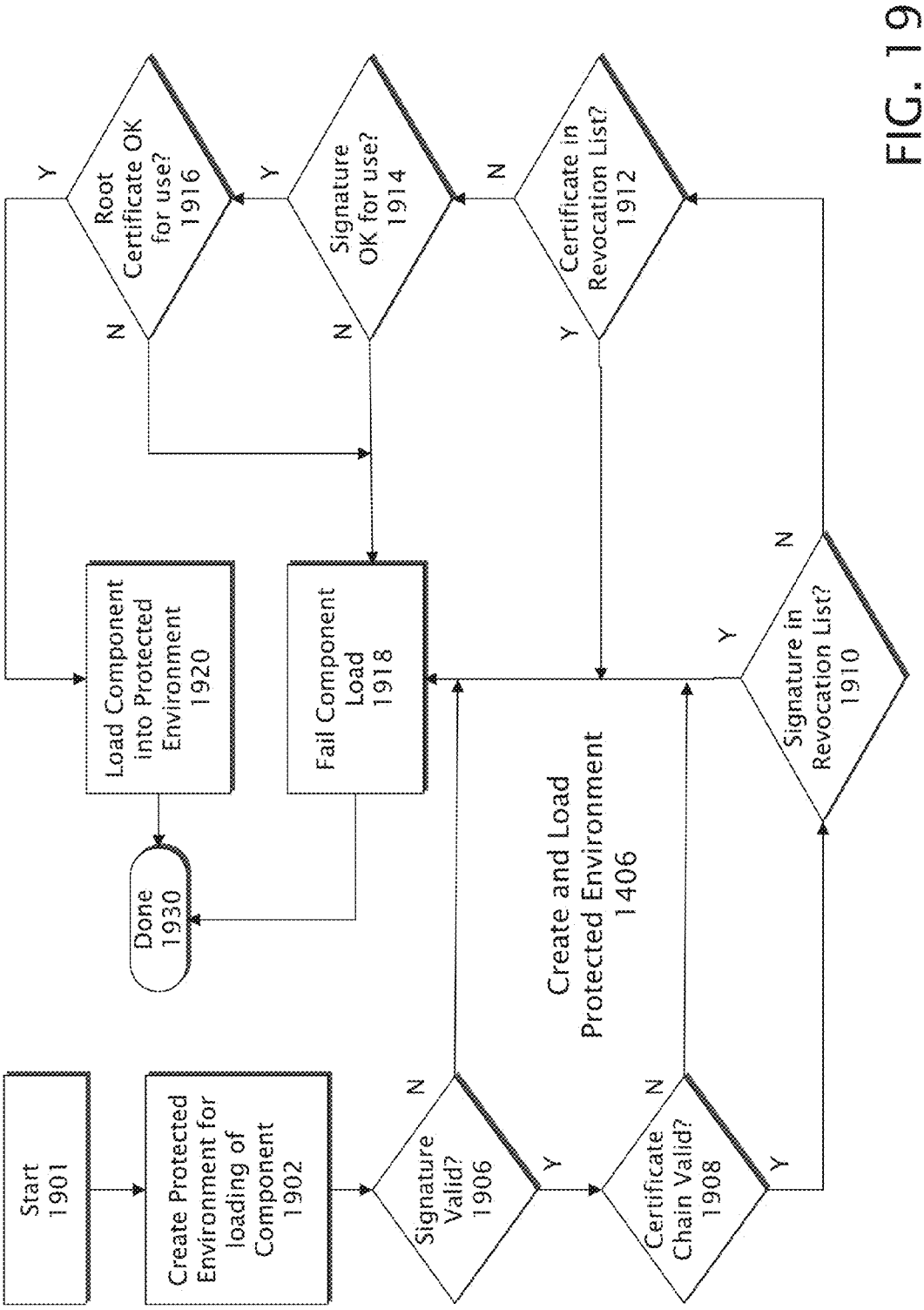

Start 1901
Create Protected Environment for loading of Component 1902
Signature Valid? 1906
Certificate Chain Valid? 1908
Signature in Revocation List? 1910
Certificate in Revocation List? 1912
Signature OK for use? 1914
Root Certificate OK for use? 1916
Fail Component Load 1918
Load Component into Protected Environment 1920
Done 1930
Create and Load Protected Environment 1406
Y
N

FIG. 19

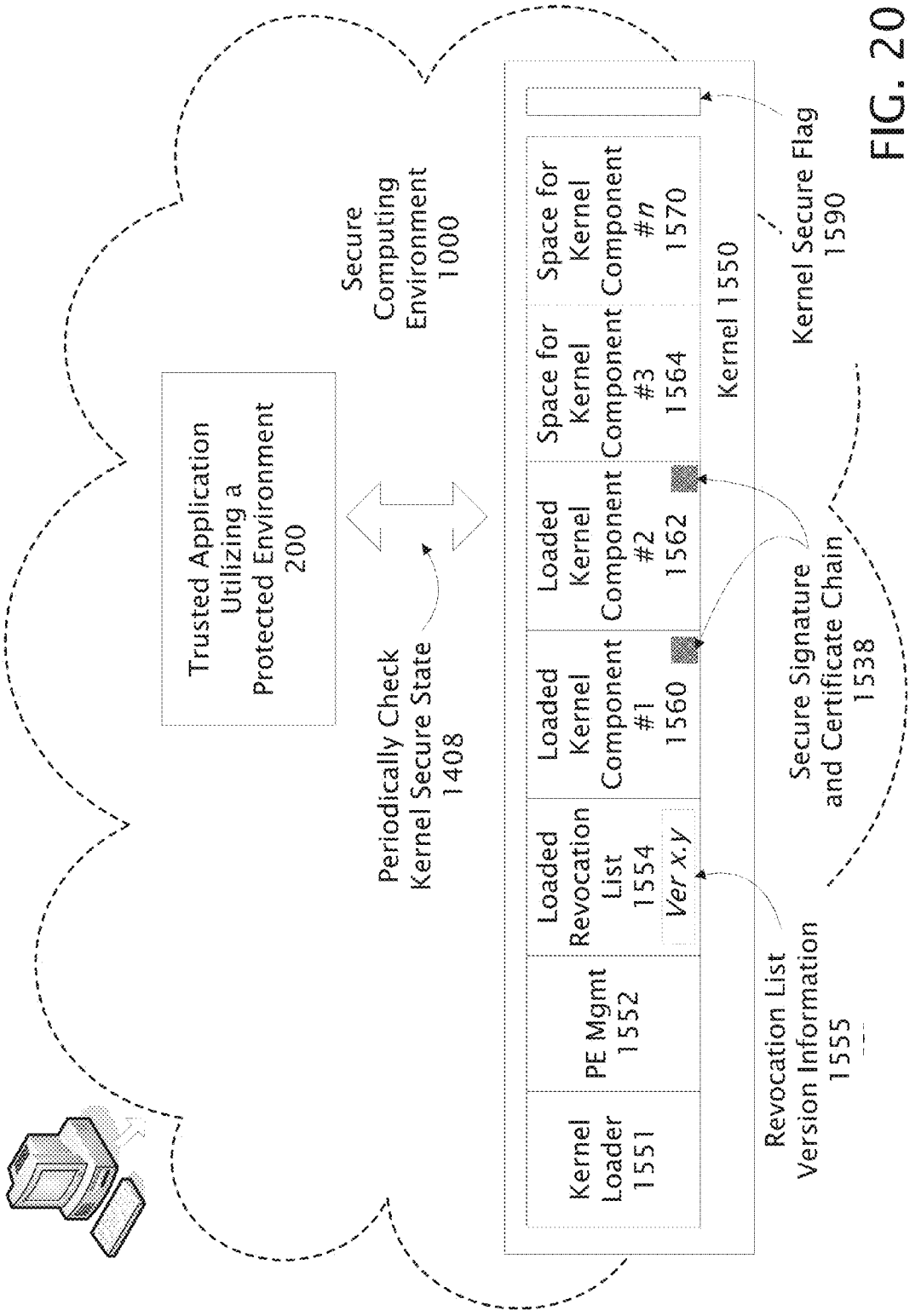

Secure Computing Environment 1000
Trusted Application Utilizing a Protected Environment 200
Periodically Check Kernel Secure State 1408
Kernel 1550
Kernel Loader 1551
PE Mgmt 1552
Loaded Revocation List 1554
Ver x.y
Loaded Kernel Component #1 1560
Loaded Kernel Component #2 1562
Space for Kernel Component #3 1564
Space for Kernel Component #n 1570
Kernel Secure Flag 1590
Secure Signature and Certificate Chain 1538
Revocation List Version Information 1555

FIG. 20


Protected Environment 232
PE Mgmt 1552
IsKernelSecure(MinRLVer) 2120
Return(SecureFlag = FALSE) 2122
Return(SecureFlag = FALSE) 2124
Return(SecureFlag = FALSE) 2126
Return(SecureFlag = TRUE) 2128
If calling process <> secure then...
Else If Kernel Secure Flag == FALSE then...
Else If loaded RLVer < MinRLVer then...
Else...
Check Kernel Secure State 1408

FIG. 21

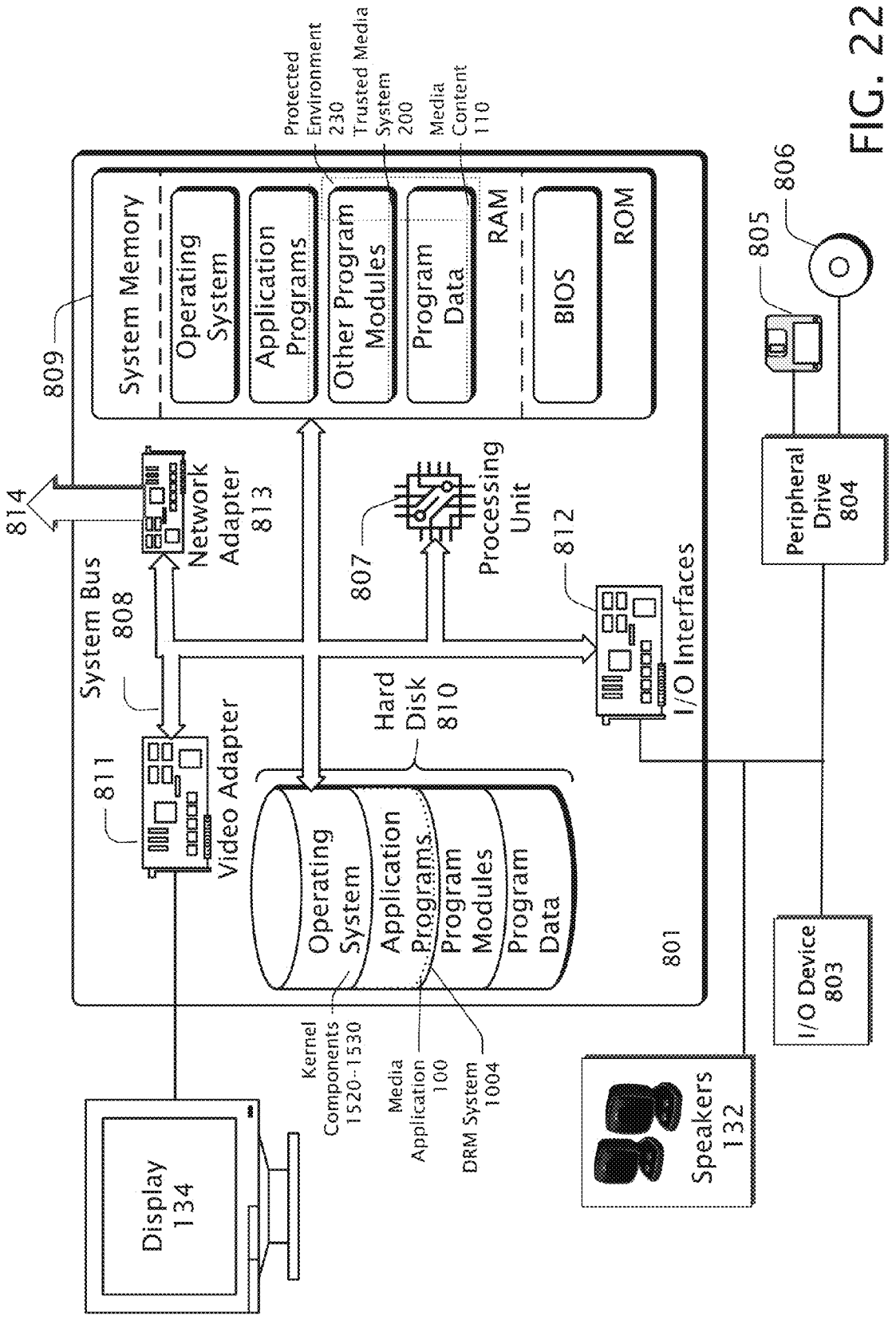

Protected Environment 230
Trusted Media System 200
Media Content 110
809
System Memory
Operating System
Application Programs
Other Program Modules
Program Data
RAM
BIOS
ROM
806
805
814
Network Adapter 813
807
Processing Unit
812
Peripheral Drive 804
System Bus 808
I/O Interfaces
811
Video Adapter
Hard Disk 810
Operating System
Application Programs
Program Modules
Program Data
801
Kernel Components 1520-1530
Media Application 100
DRM System 1004
I/O Device 803
Display 134
Speakers 132

FIG. 22

PROTECTED MEDIA PIPELINE

CROSS-REFERENCE TO RELATED APPLICATION(S)

[0001] This application is a Continuation of and claims benefit from U.S. patent application Ser. No. 11/116,689 that was filed on Apr. 27, 2005, and that is a Non-Provisional of U.S. Provisional Patent Application No. 60/673,979 that was filed on Apr. 22, 2005, each of which is incorporated herein by reference in its entirety.

DESCRIPTION OF THE DRAWINGS

[0002] The present description will be better understood from the following detailed description read in light of the accompanying drawings, wherein:

[0003] FIG. **1** is a block diagram showing an example of a typical prior art media player or application designed to operate on an exemplary personal computer.

[0004] FIG. **2** is a block diagram showing an example of a trusted media system comprising an application space and a distinct protected space.

[0005] FIG. **3** is a block diagram showing exemplary components comprising an end-to-end system for protecting media content and other data from initial input to final output of a computing environment.

[0006] FIG. **4** is a block diagram showing exemplary components comprising a protected media pipeline operating in a protected space as part of a trusted media system.

[0007] FIG. **5** is a block diagram showing an alternate example of a protected media pipeline having a proxied media source as part of a trusted media system.

[0008] FIG. **6** is a block diagram showing an example of a further alternative example of a trusted media system.

[0009] FIG. **7** is a block diagram showing a plurality of protected media pipelines.

[0010] FIG. **8** is a block diagram showing an exemplary computing environment in which the software applications, systems and methods described in this application may be implemented.

[0011] FIG. **9** is a block diagram showing a conventional media application processing media content operating in a conventional computing environment with an indication of an attack against the system.

[0012] FIG. **10** is a block diagram showing a trusted application processing media content and utilizing a protected environment or protected space that tends to be resistant to attack.

[0013] FIG. **11** is a block diagram showing exemplary components of a trusted application that may be included in the protected environment.

[0014] FIG. **12** is a block diagram showing a system for downloading digital media content from a service provider that utilizes an exemplary trusted application utilizing a protected environment.

[0015] FIG. **13** is a block diagram showing exemplary attack vectors that may be exploited by a user or mechanism attempting to access media content or other data typically present in a computing environment in an unauthorized manner.

[0016] FIG. **14** is a flow diagram showing the process for creating and maintaining a protected environment that tends to limit unauthorized access to media content and other data.

[0017] FIG. **15** is a block diagram showing exemplary kernel components and other components utilized in creating an exemplary secure computing environment.

[0018] FIG. **16** and FIG. **17** are flow diagrams showing an exemplary process for loading kernel components to create an exemplary secure computing environment.

[0019] FIG. **18** is a block diagram showing a secure computing environment loading an application into an exemplary protected environment to form a trusted application that may be resistant to attack.

[0020] FIG. **19** is a flow diagram showing an exemplary process for creating a protected environment and loading an application into the protected environment.

[0021] FIG. **20** is a block diagram showing an exemplary trusted application utilizing an exemplary protected environment periodically checking the security state of the secure computing environment.

[0022] FIG. **21** is a flow diagram showing an exemplary process for periodically checking the security state of the secure computing environment.

[0023] FIG. **22** is a block diagram showing an exemplary computing environment including a representation of a protected environment, a trusted media system, and other related elements.

[0024] Like reference numerals are used to designate like elements in the accompanying drawings.

DETAILED DESCRIPTION

[0025] The detailed description provided below in connection with the appended drawings is intended as a description of the present examples and is not intended to represent the only forms in which the present examples may be constructed or utilized. The description sets forth the functions of the examples and the sequence of steps for constructing and operating the examples. However, the same or equivalent functions and sequences may be accomplished by different examples.

[0026] Although the present examples are described and illustrated herein as being implemented in a computer system, the system described is provided as an example and not a limitation. As those skilled in the art will appreciate, the present examples are suitable for application in a variety of different types of electronic systems.

Introduction

[0027] Digital media content is widely used in the form of CDs, DVDs and downloadable files. Various devices are able to process this media content including personal computers running various media player applications and the like, CD and DVD players, MP3 players and other general-purpose and/or dedicated electronic devices designed to process digital media content.

[0028] Because media content often comes in the form of a for-sale consumer products and the like, producers and providers may be anxious to protect their media content from unauthorized access, duplication, use, etc. Therefore, media content is often encrypted and/or otherwise secured. Some form of encryption key and/or other access mechanism may be provided for use with the media so that it can be accessed when and how appropriate. This key or mechanism may be used by a media application or the like to gain access to the protected media for processing, playing, rendering, etc.

[0029] Once the key or other mechanism has been used to decrypt or otherwise access media content within a system the media content may be vulnerable in its unprotected form. It may be possible to attack the system and/or media application so as to gain access to the unprotected media content. This may lead to the unauthorized access, use, duplication, distribution, etc. of the media content.

[0030] To avoid unauthorized access, a system that rightfully accesses the media content should be capable of protecting the media content. This protection should extend from the time the key or the like is obtained, used to access the media content, throughout any processing performed on the content, until the content is appropriately rendered in its authorized form. For example, a particular meeting may be recorded and encrypted using an access key with the intent of making the recording available to authorized personnel. Later, the recording is made available to an authorized individual via a media application on a PC. The media application uses the key to decrypt and access the media content, process it and play it for the listener. But if the media application itself has been compromised, or the application and/or content is attacked, the unencrypted media may no longer be protected.

[0031] One approach may be to construct a system for accessing, processing and rendering the media content within a protected environment that is designed to prevent unauthorized access to the media content. The example provided here describes a process and system for protecting media content from unauthorized access. Protection may be afforded by a protected media pipeline, among other mechanisms, which processes some, or all, of a media within a protected environment or protected space. A protected media pipeline may be composed of several elements.

[0032] A media source that may be part of the protected media pipeline accesses the media content, passes it through a set of transform functions or processes (decoders, effects, etc.) and then to a media sink which renders the processed media to a media output(s) (video rendering process, audio rendering process, etc). As an example, rendering may be as simple as sending audio signals to a set of headphones or it may be sending protected content in a secure manner to yet another process, system or mechanism external to the protected media pipeline.

[0033] A protected media pipeline may be constructed as a set or chain of media processing mechanisms operating in a secure or protected environment. In a PC, a protected media pipeline can be thought of as a software process that operates in a secure environment which protects the media content from unauthorized access while the content is being accessed, played and/or otherwise processed by the media system. When media content is being processed by an electronic device, a protected media pipeline can be thought of as a set of media processing mechanisms operating within a secure environment such that the media being processed is resistant to unauthorized access. The mechanism for providing this resistance may be purely physical in nature, such as a sealed case or lack of access points to the media content.

[0034] There may be two major aspects to constructing a trusted media system with a protected media pipeline. First, a trusted media system may be designed and constructed in such a way that it acknowledges and adheres to any access rules of the media content by ensuring that no actions are taken with the content above and beyond those allowed. Various mechanisms known to those skilled in this technology area may be used to address this first point. These mechanisms may include using encryption/decryption, key exchanges, passwords, licenses, interaction with a digital rights management system, and the like. Further, this may be as simple as storing the media content on/in a device such that it is resistant to physical, electronic or other methods of accessing and using the media content, except as intended.

[0035] Second, the trusted media system may be designed and constructed such that the media content being processed is secure from malicious attacks and/or unauthorized access and use. Processing the media content via a protected media pipeline operating in a protected environment or protected space addresses this second point. So in short, a protected media pipeline operating in a protected space refers to a media processing environment that resists unauthorized access to the media content being processed.

[0036] FIG. **1** is a block diagram showing an example of a typical prior art media player or application **100** designed to operate on an exemplary personal computer (FIG. **8**, **800**). Equivalently, media players may operate on other devices with similar processing capabilities such as consumer electronic devices and the like. Other media applications may include, but are not limited to, media processors, media manipulators, media analyzers, or media formatters. A media application may be a software application program that provides a way of playing media such as audio and video by a digital processor such as a CPU (FIG. **8**, **807**) or the like. A media application may include a user interface or graphic **101** that may indicate the media being played and provides various user controls. Controls may be accessed through activation with a computer pointing device such as a mouse or by conventional buttons or the like. Such a media application may be thought of as a software application program operating in an application space **102** that is provided by the PC's computing environment (FIG. **8**, **801**) or operating system.

[0037] Another example of a media player may be a hardware device comprising a memory capable of storing media content and various button, switches, displays and controls and the like to allow a user to control the device, select the media to be played, control volume, download media content, etc.

[0038] The media player **100** may be comprised of mechanisms **104**, **106** and **108**. These mechanisms may operate in the application space **102**. For a software media player, an application space **102** may be a space created in system memory (FIG. **8**, **809**) on a PC (FIG. **8**, **800**) where various software components or processes can be loaded and executed. For a hardware media player an application space **102** may be a printed circuit board and an electronic module containing the electronic elements that perform the processing and functions of the media player **100**. The media player application **100** may include other spaces and mechanisms which may provide additional capabilities or features that may or may not be directly related to the processing of media. For example, a second media player playing a music selection may operate in a media application at the same time as a media player playing a newscast.

[0039] The application space **102** may include a user interface process **104** coupled to a media control process **106** which in turn is coupled to a media processing process **108**. Typically these processes enable the media application **100** to couple to a source of media content **110**, process the media content **110** and render it via media output **130**. The media

content **110** may or may not be encrypted or otherwise protected as part of an overall security and access control scheme.

[0040] For example, when activated the media application **100** may access audio content **112** and video content **114** typically available on a DVD ROM, an on-line source, or the like. The media content **110** may be played via media processing **108** which renders the content as audio output **132** and/or video output **134**. Audio and video may typically be rendered on the speakers and/or display of a PC (FIG. **8**, **800**). This system is only one example of common media applications and environments that enable audio and video and the like to be processed, played and/or provided to other processes or systems. Another example of a media application would be a consumer electronic device such as an electronic juke box or the like. Yet another example would be a dedicated electronic device, with or without software and/or firmware.

[0041] Application space **102** may contain various processes and, in this example, includes the user interface process **104**, the media control process **106**, the media processing process **108**, or their equivalents, used to coordinate and control the overall operation of the media application **100** and its processes. Typically, to prepare the media content **110**, the user interface process **104** may provide an interface **101** for interaction between the user and the application. The media control process **106** or its equivalents may provide the overall management and control of the internal operations of the media application **100**. The media processing process **108** may perform the processing of the media content **110** making it possible to render the media content via the media output **130**, or perform whatever other media processing it may have been designed to perform.

[0042] The processes described above may not be secure against unauthorized access to the media content **110**. Processing the media content **110** via such a system may expose it to unauthorized access. Such an unprotected application may enable users and/or attackers, with varying degrees of effort, to access and make use of the media content **110** in an unauthorized manner. For example, unauthorized access may enable the unauthorized sharing, copying, modifying, and/or distributing of media content **110**.

Exemplary Trusted Media System

[0043] FIG. **2** is a block diagram showing an example of a trusted media system **200** comprising an application space **202** and a distinct protected space **230**. In this exemplary embodiment of a media player the system comprises a protected media pipeline **232** operating within a protected space **230** in addition to user interface **204** and media control **206** mechanisms operating in the application space **202**.

[0044] The protected space **230** typically provides a protected environment for media content **110** processing, the protected space **230** resisting unauthorized access to the media content **110** during processing. Media content **110** is typically protected by various built-in security schemes to deliver it un-tampered—with to a user, such as encryption and the like. However, once the media content **110** is decrypted or the like for processing, additional mechanisms to protect it from unauthorized access are required. A protected media pipeline **232** operating in a protected space **230**.

[0045] Application space **202** may be contain various mechanisms including, but not limited to, a user interface mechanism **204** and a media control mechanism **206**, or their equivalents, which are coupled to the protected media pipeline **232** operating within the protected space **230**. Typically the user interface process **204** may provide an interface **201** or set of controls for interaction between the user and the system. The media control process **206** may provide the overall management and control of the internal operations of the trusted media system **200**. The protected media pipeline **232** operating in the protected space **230** may perform the processing of the media content **110** and render the content via the media output **130**, or perform whatever other media processing the media system **200** is designed to perform.

[0046] One or more protected spaces **230** may be provided as an extension of a computing environment (FIG. **8**, **801**) and typically possess a heightened level of security and access control. A protected space **230** may also include mechanisms to ensure that any mechanism operating inside it, such as a protected media pipeline **232**, along with any media content being processed within the protected space **230**, are used and accessed appropriately. In some embodiments the access and use privileges may be indicated by a media content license and/or a digital rights management system. Alternatively, mechanisms such as password protection, encryption and the like may provide access control.

[0047] FIG. **3** is a block diagram showing exemplary components comprising an end-to-end system for protecting media content **110** and other data from initial input **302** to final output **308** of a computing environment **800**. Such a system tends to protect media **110** or other data from the point of entry into a computing environment **800** to its final output **130** in addition to providing protection during processing within a protected media pipeline **232** and/or other processing components. Such end-to-end protection may be provided via three major components-protected input **302**, a protected space **230** for processing and protected output **308**.

[0048] Protected input **302** may be implement in hardware and/or software and may limit unauthorized access to media content **110** and/or other data as it is initially received onto the system **800** from some source such as a storage device, network connection, physical memory device and the like. The protected input **302** may be coupled to a protected media pipeline **232** via a secure connection **304**. The secure connection **304** allows transfer of the media content **110** between the protected input **302** and the protected media pipeline **232** and/or other processing components and may be implemented using mechanisms such that it is tamper resistant.

[0049] Protected output **306** may be implemented in hardware and/or software and may limit unauthorized access to media content **110** as it is transferred from a protected media pipeline **232** or other processing to the output of the computing environment **800** which may be speakers, video displays, storage media, network connections and the like. The protected output **308** may be coupled to a protected media pipeline **232** via a secure connection **306**. The secure connection **306** allows transfer of the media content **110**, which may be in a processed form, between the protected media pipeline **232** and the protected output **308** and may be implemented using mechanisms such that it is tamper resistant.

[0050] Tamper resistance as used here includes limiting unauthorized access, resisting attack and otherwise protecting media content and/or other data from being compromised.

[0051] A protected space may also be referred to as a protected environment. Protected spaces or environments and their creation and maintenance are described beginning with the description of FIG. **9** below.

Protected Media Pipeline

[0052] FIG. **4** is a block diagram showing exemplary components comprising a protected media pipeline **232** operating in a protected space **230** as part of a trusted media system **200**. The components **400**, **421**, **422**, **425**, and **480** form a protected media pipeline **232** operating in a protected space **230**. Of these components, the transforms mechanisms **420** process the media content to prepare it for output. The protected space **230** may also contain other protected elements **410** of the trusted media system **200**.

[0053] The protected media pipeline **232** typically performs the function of accessing and processing protected media content **110** and producing a protected output in the format determined by the trusted media system **200**. Unprotected media content may also be processed in a protected media pipeline **232**. Further, unprotected media pipelines may be constructed and operate in the application space **202** or other spaces. However, an unprotected media pipeline operating in the application space **202** would not benefit from a protected environment **230** which limits unauthorized access to the media content. For processing some types of media content, such as unprotected or unencrypted media content, an unprotected pipeline may be acceptable. In some embodiments there may be a plurality of media content having different security levels (some protected and some unprotected), processed through one or more pipelines each adapted to provide the desired level of protection.

[0054] In the protected media pipeline **232** a media source **400** may be coupled to a series of transform functions or mechanisms **420**. A first transform function F(a)**1** **421** may be coupled to a second transform function F(b)**2** **422** which in turn may be coupled to any number of additional transform functions represented by F(z)n **425**. The output of the set of transform functions **420** may be coupled to a media sink **480**. There are typically one or more transform functions in a protected media pipeline **232**, the specific function of each transform depending on the media content **110** and the processing that the trusted media system **200** is designed to perform.

[0055] The example shown illustrates transform mechanisms that may be connected in series forming a transform chain. In alternative embodiments of a protected media pipeline **232**, two or more of the transform mechanisms may be coupled in parallel and/or two or more media pipelines may be coupled at some point in each pipeline's transform chain forming a single pipeline from that point forward. Further, each transform may have a single input or a plurality of inputs and they may have a single output or a plurality of outputs.

[0056] The media source **400** may access media content **110** via hardware and/or appropriate driver software or the like. For example, using a PC for processing music stored on a CD, the media source **400** couples to CD ROM driver software which controls the CD ROM drive hardware (FIG. **8**, **804**) to read audio data from a CD ROM disk (FIG. **8**, **806**). The media source **400** is a mechanism used in the construction of a media pipeline to access and receive the media content **110** and make it available to the remaining mechanisms of the media pipeline. Alternatively, a media source **400** may couple with a semiconductor memory in a consumer electronic device to access music stored on the device. Equivalent media sources may provide access to one or more types of media content, including video, digital recordings, and the like.

[0057] The media transforms **420**, represented by F(a)**1**, F(b)**2** and F(z)n, (**421**, **422** and **425** respectively) perform specific operations on the media content provided by the media source **400** and may each perform different operations. There are typically at least one media transform in a media pipeline. The media transforms **421**, **422** and **425** prepare and/or process the media content **110** for rendering via the media output **130** and/or for further processing. The specific transformations performed may include operations such as encryption and/or decryption of media content, image enhancement of video content, silence detection in audio content, decompression, compression, volume normalization, and the like. Transforms may process media content **110** automatically or be controlled by a user via virtual or physical handles provided through a user interface **204**. The specific transforms provided in a pipeline depend on the media content **110** to be processed and the function the trusted media system **200** has constructed the pipeline to perform. In a simple media system or application the processing may be as minimal as decoding an audio media and controlling the volume of the media accessed from a semiconductor memory and played on a headset. In a more complex media system or application a wide variety of processing and media manipulation are possible.

[0058] In a trusted media system **200** designed to process encrypted media content one of the transform mechanisms, typically the first transform F(a)**1** **421**, may be a codec which decodes the media content such that it may be further processed. In alternative examples, decryption and/or decompression operations may be performed by distinct mechanisms and one or both operations may be eliminated depending on the format of media content being processed.

[0059] When operating on a PC, the media sink **480** may couple the processed or transformed media content **110** to the media output **130** via the media I/O hardware (FIG. **8**, **812**) controlled by appropriate driver programs. For example, in the case of audio data, the media sink **480** may couple to an available sound driver program which couples audio data that has been transformed to audio output hardware such as an amplifier and/or speakers (FIG. **2**, **132**). When operating on a consumer electronic device, the media sink **480** may be coupled, for example, to an audio amplifier which in turn couples to speakers or a headset through a connector on the device's case.

[0060] By constructing a pipeline that performs the sourcing, transform and sinking functions within a protected space **230**, unauthorized access to the media content **110** may be restricted in a manner that conforms to the wishes of the media content provider/owner. Thus, this approach tends to provide a secure processing environment such that a media content provider may trust that their media content **110** will not be compromised while being processed.

[0061] The output of the protected media pipeline **232** may be coupled to the input of a media output **130**. Alternatively the output of a protected media pipeline **232** may couple to the input of another protected media pipeline or some other process. This coupling may be implemented such that it is tamper resistant and restricts unauthorized access to any data or media content flowing from one pipeline to another or to some other process. The remainder of the elements illustrated in FIG. **4** operate as previously described for FIG. **2**.

[0062] FIG. **5** is a block diagram showing an alternate example of a protected media pipeline **552** having a proxied media source **510** as part of a trusted media system **500**. The

proxied media source **510** includes a media source portion **518** and a stub portion **520** that may operate in an unprotected application space **502**, and a proxy portion **540** that may operate in a protected space **550**. The proxied media source **510** may allow media content **110** to be transferred from the application space **502** via the media source **518** and the stub **520** to the protected space **550** via the proxy **540** by using remote procedure calls or the like.

[0063] When used in a PC environment (FIG. **8**, **800**), the proxied media source **510** architecture described here may simplify the creation of the media source modules by third-party software makers or content providers. Such a simplification may be provided by splitting the proxied media source **510** such that media application writers may only need to implement the media source portion **518**. The stub portion **520** and proxy portion **540** may be provided as an element of the protected environment **550**.

[0064] Further, the use of a proxied media source **510** may support mixing protected and unprotected media content **110** by allowing protected media content to be directed from a media source **518** to a first stub operating as part of a protected media pipeline while the unprotected media content may be directed from the media source **518** to processing modules operating within the unprotected application space **502** or other unprotected space via a second stub portion also operating within the unprotected application space **502** or some other unprotected space.

[0065] Similar to the proxied media source **510**, the media sink **480** may also be proxied and split into stub and proxy portions. The stub portion may operate in the protected space **650** and may encrypt data prior to forwarding it to the proxy portion operating in an application space **202** or some other space. The remainder of the elements in FIG. **5** operate as previously described for FIG. **4**.

[0066] FIG. **6** is a block diagram showing an example of a further alternative example of a trusted media system **600**. In this embodiment the trusted media system **600** includes a protected media source **610** constructed to include a media source portion **618** and a stub portion **620** which operate in a protected media space **609**, and a proxy portion **640** which operates in a protected space **650**. The two protected regions **609** and **650** are coupled by the protected media source **610** with data being passed from the media source portion **618** via the stub portion **620** operating in the protected media space **609** to the proxy portion **640** operating in the protected space **650**. The protected media source **610** may allow media content **110** to be transferred from the protected media space **609** to the protected pipeline space **650** using remote procedure calls or the like. The protected media source **610** architecture described here may simplify the creation of the media source by third-parties or content providers and result in more stable and secure protected media applications **600**. The remaining elements of FIG. **6** operate as previously described for FIG. **4** and FIG. **5**.

[0067] FIG. **7** is a block diagram showing a plurality of protected media pipelines **751**-**759**. The protected media pipelines **751**, **752**, **759** operate in a protected space **700**. Alternatively each protected media pipeline may operate in its own protected space or various numbers of pipelines may be grouped into one or more protected spaces in any combination. A trusted media system may provide several such protected media pipelines.

[0068] An example of such a system may be a trusted media system playing a DVD with its audio content in Dolby digital 5.1 format. In this example there may be six different audio pipelines, one for each of the audio channels, in addition to a video pipeline for the video portion of the DVD. All of the protected media pipelines may operate in the same protected space as shown or, alternatively, the protected media pipelines may be grouped in groups of one or more with each group operating in its own distinct protected space.

[0069] In alternative embodiments of a protected media pipeline **232**, two or more of the sources, transform mechanisms and/or sinks may be coupled in parallel and/or two or more media pipelines may be coupled at some point in each pipeline forming a single pipeline from that point forward. Alternatively a single pipeline may split into two pipelines. Further, sources, transforms and/or sinks may have a single input or a plurality of inputs and/or they may have a single output or a plurality of outputs. The remaining elements of FIG. **7** operate as previously described for FIG. **4**.

[0070] FIG. **8** is a block diagram showing an exemplary computing environment **800** in which the software applications, systems and methods described in this application may be implemented. Exemplary personal computer **800** is only one example of a computing system or device that may process media content (FIG. **4**, **110**) and is not intended to limit the examples described in this application to this particular computing environment or device type.

[0071] The computing environment can be implemented with numerous other general purpose or special purpose computing system configurations. Examples of well known computing systems may include, but are not limited to, personal computers **800**, hand-held or laptop devices, microprocessor-based systems, multiprocessor systems, set top boxes, programmable consumer electronics, gaming consoles, consumer electronic devices, cellular telephones, PDAs, and the like.

[0072] The PC **800** includes a general-purpose computing system in the form of a computing device **801**. The components of computing device **801** may include one or more processors (including CPUs, GPUs, microprocessors and the like) **807**, a system memory **809**, and a system bus **808** that couples the various system components. Processor **807** processes various computer executable instructions to control the operation of computing device **801** and to communicate with other electronic and computing devices (not shown) via various communications connections such as a network connection **814** an the like. The system bus **808** represents any number of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures.

[0073] The system memory **809** includes computer readable media in the form of volatile memory, such as random access memory (RAM), and/or non-volatile memory, such as read only memory (ROM). A basic input/output system (BIOS) may be stored in ROM. RAM typically contains data and/or program modules that are immediately accessible to and/or presently operated on by one or more of the processors **807**. A trusted media system **200** may be contained in system memory **809**.

[0074] Mass storage devices **804** and **810** may be coupled to the computing device **801** or incorporated into the computing device by coupling to the system bus. Such mass storage devices **804** and **810** may include a magnetic disk drive which reads from and/or writes to a removable, non volatile magnetic disk (e.g., a "floppy disk") **805**, or an optical disk drive

that reads from and/or writes to a removable, non-volatile optical disk such as a CD ROM, DVD ROM or the like **806**. Computer readable media **805** and **806** typically embody computer readable instructions, data structures, program modules and the like supplied on floppy disks, CDs, DVDs, portable memory sticks and the like.

[0075] Any number of program modules may be stored on the hard disk **810**, other mass storage devices **804**, and system memory **809** (limited by available space), including by way of example, an operating system(s), one or more application programs, other program modules, and program data. Each of such operating system, application program, other program modules and program data (or some combination thereof) may include an embodiment of the systems and methods described herein. For example, a trusted media system **200** may be stored on mass storage devices **804** and **810** and/or in system memory **809**.

[0076] A display device **134** may be coupled to the system bus **808** via an interface, such as a video adapter **811**. A user can interface with computing device **800** via any number of different input devices **803** such as a keyboard, pointing device, joystick, game pad, serial port, and/or the like. These and other input devices may be coupled to the processors **807** via input/output interfaces **812** that may be coupled to the system bus **808**, and may be coupled by other interface and bus structures, such as a parallel port, game port, and/or a universal serial bus (USB).

[0077] Computing device **800** may operate in a networked environment using communications connections to one or more remote computers and/or devices through one or more local area networks (LANs), wide area networks (WANs), the Internet, optical links and/or the like. The computing device **800** may be coupled to one or more networks via network adapter **813** or alternatively by a modem, DSL, ISDN interface and/or the like.

[0078] Communications connection **814** is an example of communications media. Communications media typically embody computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communications media include wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, radio frequency, infrared, and other wireless media.

[0079] Those skilled in the art will realize that storage devices utilized to store computer-readable program instructions can be distributed across a network. For example a remote computer or device may store an example of the system described as software. A local or terminal computer or device may access the remote computer or device and download a part or all of the software to run the program. Alternatively the local computer may download pieces of the software as needed, or distributively process the software by executing some of the software instructions at the local terminal and some at remote computers or devices.

[0080] Those skilled in the art will also realize that by utilizing conventional techniques known to those skilled in the art that all, or a portion, of the software instructions may be carried out by a dedicated electronic circuit such as a digital signal processor ("DSP"), programmable logic array ("PLA"), or the like. The term electronic apparatus as used herein includes computing devices, consumer electronic devices including any software and/or firmware and the like, and electronic devices or circuits containing no software and/or firmware and the like.

[0081] The term computer readable medium may include system memory, hard disks, mass storage devices and their associated media, communications media, and the like.

Protected Environment

[0082] FIG. **9** is a block diagram showing a conventional media application **100** processing media content **110** operating in a conventional computing environment **900** with an indication of an attack **907** against the system **901**. A conventional computing environment **900** may be provided by a personal computer ("PC") or consumer electronics ("CE") device **901** that may include operating system ("OS") **902**. Typical operating systems often partition their operation into a user mode **903**, and a kernel mode **904**. User mode **903** and kernel mode **904** may be used by one or more application programs **100**. An application program **100** may be used to process media content **110** that may be transferred to the device **901** via some mechanism, such as a CD ROM drive, Internet connection or the like. An example of content **110** would be media files that may be used to reproduce audio and video information.

[0083] The computing environment **900** may typically include an operating system ("OS") **902** that facilitates operation of the application **100**, in conjunction with the one or more central processing units ("CPU"). Many operating systems **902** may allow multiple users to have access to the operation of the CPU. Multiple users may have ranges of access privileges typically ranging from those of a typical user to those of an administrator. Administrators typically have a range of access privileges to applications **100** running on the system, the user mode **903** and the kernel **904**. Such a computing environment **900** may be susceptible to various types of attacks **907**. Attacks may include not only outsiders seeking to gain access to the device **901** and the content **110** on it, but also attackers having administrative rights to the device **901** or other types of users having whatever access rights granted them.

[0084] FIG. **10** is a block diagram showing a trusted application **200** processing media content **110** and utilizing a protected environment or protected space **230** that tends to be resistant to attack **1005**. The term "trusted application", as used here, may be defined as an application that utilizes processes operating in a protected environment such that they tend to be resistant to attack **1005** and limit unauthorized access to any media content **110** or other data being processed. Thus, components or elements of an application operating in a protected environment are typically considered "trusted" as they tend to limit unauthorized access and tend to be resistant to attack. Such an application **200** may be considered a trusted application itself or it may utilize another trusted application to protect a portion of its processes and/or data.

[0085] For example, a trusted media player **200** may be designed to play media content **110** that is typically licensed only for use such that the media content **110** cannot be accessed in an unauthorized manner. Such a trusted application **200** may not operate and/or process the media content **110** unless the computing environment **1000** can provide the required level of security, such as by providing a protected environment **230** resistant to attack **1005**.

[0086] As used herein, the term "process" may be defined as an instance of a program (including executable code, machine instructions, variables, data, state information, etc.), residing and/or operating in a kernel space, user space and/or any other space of an operating system and/or computing environment.

[0087] A digital rights management system **1004** or the like may be utilized with the protected environment **230**. The use of a digital rights management system **1004** is merely provided as an example and may not be utilized with a protected environment or a secure computing environment. Typically a digital rights management system utilizes tamper-resistant software ("TRS") which tends to be expensive to produce and may negatively impact computing performance. Utilizing a trusted application **200** may minimize the amount of TRS functionality required to provide enhanced protection.

[0088] Various mechanisms known to those skilled in this technology area may be utilized in place of, in addition to, or in conjunction with a typical digital rights management system. These mechanisms may include, but are not limited to, encryption/decryption, key exchanges, passwords, licenses, and the like. Thus, digital right management as used herein may be a mechanism as simple as decrypting an encrypted media, utilizing a password to access data, or other tamper-resistant mechanisms. The mechanisms to perform these tasks may be very simple and entirely contained within the trusted application **200** or may be accessed via interfaces that communicate with complex systems otherwise distinct from the trusted application **200**.

[0089] FIG. **11** is a block diagram showing exemplary components of a trusted application **200** that may be included in the protected environment **230**. A trusted application **200** will typically utilize a protected environment **230** for at least a portion of its subcomponents **232**, **400**, **480**. Other components **1101** of the trusted application may not utilize a protected environment. Components **232**, **400** and **480** involved in the processing of media content or data that may call for an enhanced level of protection from attack or unauthorized access may operate within a protected environment **230**. A protected environment **230** may be utilized by a single trusted application **200** or, possibly, by a plurality of trusted applications. Alternatively, a trusted application **200** may utilize a plurality of protected environments. A trusted application **200** may also couple to and/or utilize a digital rights management system **1004**.

[0090] In the example shown, source **400** and sink **480** are shown as part of a media pipeline **232** operating in the protected environment **230**. A protected environment **230** tends to ensure that, once protected and/or encrypted content **1109** has been received and decrypted, the trusted application **200** and its components prevent unauthorized access to the content **1109**.

[0091] Digital rights management **1004** may provide a further avenue of protection for the trusted application **200** and the content **1109** it processes. Through a system of licenses **1108**, device certificates **1111**, and other security mechanisms a content provider is typically able to have confidence that encrypted content **1109** has been delivered to the properly authorized device and that the content **1109** is used as intended.

[0092] FIG. **12** is a block diagram showing a system for downloading digital media content **1210** from a service provider **1207** to an exemplary trusted application **200** utilizing a protected environment **230**. In the example shown the trusted application **200** is shown being employed in two places **1201**, **1203**. The trusted application **200** may be used in a CE device **1201** or a PC **1203**. Digital media **1210** may be downloaded via a service provider **1207** and the Internet **1205** for use by the trusted application **200**. Alternatively, digital media may be made available to the trusted application via other mechanisms such as a network, a CD or DVD disk, or other storage media. Further, the digital media **1210** may be provided in an encrypted form **1109** requiring a system of decryption keys, licenses, certificates and/or the like which may take the form of a digital rights management system **1004**. The data or media content **1210** provided to the trusted application may or may not be protected, i.e., encrypted or the like.

[0093] In one example, a trusted application **200** may utilize a digital rights management ("DRM") system **1004** or the like along with a protected environment **230**. In this case, the trusted application **200** is typically designed to acknowledge, and adhere to, the content's usage policies by limiting usage of the content to that authorized by the content provider via the policies. Implementing this may involve executing code which typically interrogates content licenses and subsequently makes decisions about whether or not a requested action can be taken on a piece of content. This functionality may be provided, at least in part, by a digital rights management system **1004**. An example of a Digital Rights Management system is provided in U.S. patent application Ser. No. 09/290,363, filed Apr. 12, 1999, U.S. patent application Ser. Nos. 10/185,527, 10/185,278, and 10/185,511, each of which filed on Jun. 28, 2002, and incorporated herein by reference in its entirety.

[0094] Building a trusted application **200** that may be utilized in the CE device **1201** or the PC **1203** may include making sure the trusted application **200** which decrypts and processes the content **1109** may be "secure" from malicious attacks. Thus, a protected environment **230** typically refers to an environment that may not be easy to attack.

[0095] As shown, the trusted applications **200** operate in a consumer electronics device **1201**, which can be periodically synced to a PC **1203** that also provides a trusted application. The PC **1203** is in turn coupled **1204** to the internet **1205**. The internet connection allows digital media **1210** to be provided by a service provider **1207**. The service provider **1207** may transmit licenses and encrypted media **1206** over the internet **1205** to trusted application **200**. Once encrypted media is delivered and decrypted it may be susceptible to various forms of attack.

[0096] A protected computing environment tends to provide an environment that limit hackers from gaining access to unauthorized content. A hacker may include hackers acting as a systems administrator. A systems administrator typically has full control of virtually all of the processes being executed on a computer, but this access may not be desirable. For example, if a system user has been granted a license to use a media file it should not be acceptable for a system administrator different from the user to be able to access the media file. A protected environment tends to contribute to the creation of a process in which code that decrypts and processes content can operate without giving hackers access to the decrypted content. A protected environment may also limit unauthorized access to users of privilege, such as administrators, and/or any other user, who may otherwise gain unauthorized access to protected content. Protection may include

securing typical user mode (FIG. **9**, **903**) processes and kernel mode (FIG. **9**, **904**) processes and any data they may be processing.

[0097] Processes operating in the kernel may be susceptible to attack. For example, in the kernel of a typical operating system objects are created, including processes, which may allow unlimited access by an administrator. Thus, an administrator, typically with full access privileges, may access virtually all processes.

[0098] Protected content may include policy or similar information indicating the authorized use of the content. Such policy may be enforced via a DRM system or other mechanism. Typically, access to the protected content is granted through the DRM system or other security mechanism, which may enforce policy. However, a system administrator, with full access to the system, may alter the state of the DRM system or mechanism to disregard the content policy.

[0099] A protected environment tends to provide a protected space that restricts unauthorized access to media content being processed therein, even for high-privilege users such as an administrator. When a protected environment is used in conjunction with a system of digital rights management or the like, a trusted application may be created in which a content provider may feel that adequate security is provided to protect digital media from unauthorized access and may also protect the content's policy from be tampered with along with any other data, keys or protection mechanisms that may be associated with the media content.

[0100] Current operating system ("OS") architectures typically present numerous possible attack vectors that could compromise a media application and any digital media content being processed. For purposes of this example, attacks that may occur in an OS are grouped into two types of attacks, which are kernel mode attacks and user mode attacks.

[0101] The first type of attack is the kernel mode attack. Kernel mode is typically considered to be the trusted base of the operating system. The core of the operating system, most system and peripheral drivers operate in kernel mode. Typically any piece of code running in the kernel is susceptible to intrusion by any other piece of code running in the kernel, which tends not to be the case for user mode. Also, code running in kernel mode typically has access to substantially all user mode processes. A CPU may also provide privilege levels for various code types. Kernel mode code is typically assigned the highest level of privilege by such a CPU, typically giving it full access to the system.

[0102] The second type of attack is the user mode attack. Code that runs in user mode may or may not be considered trusted code by the system depending on the level of privilege it has been assigned. This level of privilege may be determined by the user context or account in which it is operating. User mode code running in the context of an administrator account may have full access to the other code running on the system. In addition, code that runs in user mode may be partitioned to prevent one user from accessing another's processes.

[0103] These attacks may be further broken down into specific attack vectors. The protected environment is typically designed to protect against unauthorized access that may otherwise be obtained via one or more of these attack vectors. The protected environment may protect against attack vectors that may include: process creation, malicious user mode applications, loading malicious code into a process, malicious kernel code, invalid trust authorities, and external attack vectors.

[0104] Process creation is a possible attack vector. An operating system typically includes a "create process" mechanism that allows a parent process to create a child process being created. A malicious parent process may, by modifying the create process code or by altering the data it creates, make unauthorized modifications to the child process. This could result in compromising digital media that may be processed by a child process created by a malicious parent process.

[0105] Malicious user mode applications are a possible attack vector. An operating system typically includes administrator level privileges. Processes running with administrator privileges may have unlimited access to many operating system mechanisms and to nearly all processes running on the computer. Thus, in Windows for example, a malicious user mode application running with administrator privileges may gain access to many other processes running on the computer and may thus compromise digital media. Similarly, processes operating in the context of any user may be attacked by any malicious process operating in the same context.

[0106] Loading malicious code into a secure process is a possible attack vector. It may be possible to append or add malicious code to a process. Such a compromised process cannot be trusted and may obtain unauthorized access to any media content or other data being processed by the modified process.

[0107] Malicious kernel mode code is a possible attack vector. An operating system typically includes a "system level" of privilege. In Windows, for example, all code running in kernel mode is typically running as system and therefore may have maximum privileges. The usual result is that all drivers running in kernel mode have maximum opportunity to attack any user mode application, for example. Such an attack by malicious kernel mode code may compromise digital media.

[0108] Invalid trust authorities (TAs) are a possible attack vector. TAs may participate in the validation of media licenses and may subsequently "unlock" the content of a digital media. TAs may be specific to a media type or format and may be implemented by media providers or their partners. As such, TAs may be pluggable and/or may be provided as dynamic link libraries ("DLL"). A DLL or the like may be loaded by executable code, including malicious code. In order for a TA to ensure that the media is properly utilized it needs to be able to ensure that the process in which it is running is secure. Otherwise the digital media may be compromised.

[0109] External attacks are another possible attack vector. There are a set of attacks that don't require malicious code running in a system in order to attack it. For instance, attaching a debugger to a process or a kernel debugger to the machine, looking for sensitive data in a binary file on a disk, etc., are all possible mechanisms for finding and compromising digital media or the processes that can access digital media.

[0110] FIG. **13** is a block diagram showing exemplary attack vectors **1307-1310** that may be exploited by a user or mechanism attempting to access media content or other data **1300** typically present in a computing environment **900** in an unauthorized manner. A protected environment may protect against these attack vectors such that unauthorized access to trusted applications and the data they process is limited and resistance to attack is provided. Such attacks may be made by

users of the system or mechanisms that may include executable code. The media application **100** is shown at the center of the diagram and the attack vectors **1307-1310** tend to focus on accessing sensitive data **1300** being stored and/or processed by the application **100**.

[0111] A possible attack vector **1309** may be initiated via a malicious user mode application **1302**. In the exemplary operating system architecture both the parent of a process, and any process with administrative privileges, typically have unlimited access to other processes, such as one processing media content, and the data they process. Such access to media content may be unauthorized. Thus a protected environment may ensure that a trusted application and the media content it processes are resistant to attacks by other user mode applications and/or processes.

[0112] A possible attack vector **1308** is the loading of malicious code **1303** into a process **1301**. Having a secure process that is resistant to attacks from the outside is typically only as secure as the code running on the inside forming the process. Given that DLLs and other code are typically loaded into processes for execution, a mechanism that may ensure that the code being loaded is trusted to run inside a process before loading it into the process may be provided in a protected environment.

[0113] A possible vector of attack **1310** is through malicious kernel mode code **1304**. Code running in kernel mode **904** typically has maximum privileges. The result may be that drivers running in kernel mode may have a number of opportunities to attack other applications. For instance, a driver may be able to access memory directly in another process. The result of this is that a driver could, once running, get access to a processes memory which may contain decrypted "encrypted media content" (FIG. **11**, **1109**). Kernel Mode attacks may be prevented by ensuring that the code running in the kernel is non-malicious code, as provided by this example.

[0114] A possible attack vector **1307** is by external attacks **1306** to the system **900**. This group represents the set of attacks that typically do not require malicious code to be running on the system **900**. For instance, attaching a debugger to an application and/or a process on the system, searching a machine **900** for sensitive data, etc. A protected environment may be created to resist these types of attacks.

[0115] FIG. **14** is a flow diagram showing the process **1400** for creating and maintaining a protected environment that tends to limit unauthorized access to media content and other data. The sequence **1400** begins when a computer system is started **1402** and the kernel of the operating system is loaded and a kernel secure flag is set **1404** to an initial value. The process continues through the time that a protected environment is typically created and an application is typically loaded into it **1406**. The process includes periodic checking **1408** via the protected environment that seeks to ensure the system remains secure through the time the secure process is needed.

[0116] The term "kernel", as used here, is defined as the central module of an operating system for a computing environment, system or device. The kernel module may be implemented in the form of computer-executable instructions and/or electronic logic circuits. Typically, the kernel is responsible for memory management, process and task management, and storage media management of a computing environment. The term "kernel component", as used here, is defined to be a basic controlling mechanism, module, computer-executable instructions and/or electronic logic circuit that forms a portion of the kernel. For example, a kernel component may be a "loader", which may be responsible for loading other kernel components in order to establish a fully operational kernel.

[0117] To summarize the process of creating and maintaining a protected environment:

[0118] 1. Block **1402** represents the start-up of a computer system. This typically begins what is commonly known as the boot process and includes loading an operating system from disk or some other storage media.

[0119] 2. Typically one of the first operations during the boot process is the loading of the kernel and its components. This example provides the validation of kernel components and, if all are successfully validated as secure, the setting of a flag indicating the kernel is secure. This is shown in block **1404**.

[0120] 3. After the computer system is considered fully operational a user may start an application such as a trusted media player which may call for a protected environment. This example provides a secure kernel with an application operating in a protected environment, as shown in block **1406**.

[0121] 4. Once the protected environment has been created and one or more of the processes of the application have been loaded into it and are operating, the trusted environment may periodically check the kernel secure flag to ensure the kernel remains secure, as shown in block **1408**. That is, from the point in time that the trusted application begins operation, a check may be made periodically to determine whether any unauthorized kernel components have been loaded. Such unauthorized kernel components could attack the trusted application or the data it may be processing. Therefore, if any such components are loaded, the kernel secure flag may be set appropriately.

[0122] FIG. **15** is a block diagram showing exemplary kernel components **1520-1530** and other components **1510-1514** utilized in creating an exemplary secure computing environment **1000**. This figure shows a computer system containing several components **1510-1530** typically stored on a disk or the like, several of which are used to form the kernel of an operating system when a computer is started. Arrow **1404** indicates the process of loading the kernel components into memory forming the operational kernel of the system. The loaded kernel **1550** is shown containing its various components **1551-1562** and a kernel secure flag **1590** indicating whether or not the kernel is considered secure for a protected environment. The kernel secure flag **1590** being described as a "flag" is not meant to be limiting; it may be implemented as a boolean variable or as a more complex data structure or mechanism.

[0123] Kernel components **1520-1530** are typically "signed" and may include certificate data **1538** that may enable the kernel to validate that they are the components they claim to be, that they have not been modified and/or are not malicious. A signature block and/or certificate data **1538** may be present in each kernel component **1520-1530** and/or each loaded kernel component **1560**, **1562**. The signature and/or certificate data **1538** may be unique to each component. The signature and/or certificate data **1538** may be used in the creation and maintenance of protected environments as indicated below. Typically a component is "signed" by its provider in such as way as to securely identify the source of the component and/or indicate whether it may have been tampered with. A signature may be implemented as a hash of the

component's header or by using other techniques. A conventional certificate or certificate chain may also be included with a component that may be used to determine if the component can be trusted. The signature and/or certificate data **1538** are typically added to a component before it is distributed for public use. Those skilled in the art will be familiar with these technologies and their use.

[0124] When a typical computer system is started or "booted" the operating system's loading process or "kernel loader" **1551** will typically load the components of the kernel from disk or the like into a portion of system memory to form the kernel of the operating system. Once all of the kernel components are loaded and operational the computer and operating system are considered "booted" and ready for normal operation.

[0125] Kernel component #1 **1520** thru kernel component #n **1530**, in the computing environment, may be stored on a disk or other storage media, along with a revocation list **1514**, a kernel dump flag **1512** and a debugger **1510** along with a debug credential **1511**. Arrow **1404** indicates the kernel loading process which reads the various components **1514-1530** from their storage location and loads them into system memory forming a functional operating system kernel **1550**. The kernel dump flag **1512** being described as a "flag" is not meant to be limiting; it may be implemented as a boolean variable or as a more complex data structure or mechanism.

[0126] The kernel loader **1551** along with the PE management portion of the kernel **1552**, the revocation list **1554** and two of the kernel components **1520** and **1522** are shown loaded into the kernel, the latter as blocks **1560** and **1562**, along with an indication of space for additional kernel components yet to be loaded into the kernel, **1564** and **1570**. Finally, the kernel **1550** includes a kernel secure flag **1590** which may be used to indicate whether or not the kernel **1550** is currently considered secure or not. This illustration is provided as an example and is not intended to be limiting or complete. The kernel loader **1551**, the PE management portion of the kernel **1552** and/or the other components of the kernel are shown as distinct kernel components for clarity of explanation but, in actual practice, may or may not be distinguishable from other portions of the kernel.

[0127] Included in the computing environment **1000** may be a revocation list **1514** that may be used in conjunction with the signature and certificate data **1538** associated with the kernel components **1560** and **1562**. This object **1514** may retain a list of signatures, certificates and/or certificate chains that are no longer considered valid as of the creation date of the list **1514**. The revocation list **1514** is shown loaded into the kernel as object **1554**. Such lists are maintained because a validly-signed and certified component, for example components **1560** and **1562**, may later be discovered to have some problem. The system may use such a list **1554** to check kernel components **1520-1530** as they are loaded, which may be properly signed and/or have trusted certificate data **1538**, but that may have subsequently been deemed untrustworthy. Such a revocation list **1554** will typically include version information **1555** so that it can more easily be identified, managed and updated as required.

[0128] Another component of the system that may impact kernel security is a debugger **1510**. Debuggers may not typically be considered a part of the kernel but may be present in a computing environment **1000**. Debuggers, including those known as kernel debuggers, system analyzers, and the like, may have broad access to the system and the processes running on the system along with any data present. A debugger **1510** may be able access any data in a computing environment **1000**, including media content that should not be accessed in a manner other than that authorized. On the other hand, debugging is typically a part of developing new functionality and it should be possible to debug within protected environments the code intended to process protected media content. A debugger **1510** may thus include debug credentials **1511** which may indicate that the presence of the debugger **1510** on a system is authorized. Thus detection of the presence of a debugger **1510** along with any accompanying credentials **1511** may be a part of the creation and maintenance of protected environments (FIG. **14**, **1400**).

[0129] The computing environment **1000** may include a kernel dump flag **1512**. This flag **1512** may be used to indicate how much of kernel memory is available for inspection in case of a catastrophic system failure. Such kernel dumps may be used for postmortem debugging after such as failure. If such a flag **1512** indicates that system memory is available for inspection upon a dump then the kernel **1550** may be considered insecure as hacker could run an application which exposes protected media in system memory and then force a catastrophic failure condition which may result in the system memory being available for inspection, including that containing the exposed media content. Thus a kernel dump flag **1512** may be used in the creation and maintenance of a protected environments (FIG. **14**, **1400**).

[0130] FIG. **16** and FIG. **17** are flow diagrams showing an exemplary process **1404** for loading kernel components to create an exemplary secure computing environment. This process **1404** begins after the kernel loader has been started and the PE management portion of the kernel has been loaded and made operational. Not shown in these figures, the PE management portion of the kernel may validate the kernel loader itself and/or any other kernel elements that may have been previously loaded. Validation is usually defined as determining whether or not a given component is considered secure and trustworthy as illustrated in part **2** of this process **1404**.

[0131] The term "authorized for secure use" and the like as used below with respect to kernel components has the following specific meaning. A kernel containing any components that are not authorized for secure use does not provide a secure computing environment within which protected environments may operate. The opposite may not be true as it depends on other factors such as attack vectors.

[0132] 1. Block **1601** shows the start of the loading process **1404** after the PE management portion of the kernel has been loaded and made operational. Any component loaded in the kernel prior to this may be validated as described above.

[0133] 2. Block **1602** shows the kernel secure flag initially set to TRUE unless any component loaded prior to the PE management portion of the kernel, or that component itself, is found to be insecure at which point the kernel secure flag may be set to FALSE. In practice the indication of TRUE or FALSE may take various forms; the use of TRUE or FALSE here is only an example and is not meant to be limiting.

[0134] 3. Block **1604** indicates a check for the presence of a debugger in the computing environment. Alternatively a debugger could reside remotely and be attached to the computing environment via a network or other communications media to a process in the computing environment. If no debugger is detected the loading process **1404** continues at block **1610**. Otherwise it continues at block **1609**. Not shown

in the diagram, this check may be performed periodically and the state of the kernel secure flag updated accordingly.

[0135] 4. If a debugger is detected, block **1606** shows a check for debug credentials which may indicate that debugging is authorized on the system in the presence of a protected environment. If such credentials are not present, the kernel secure flag may be set to FALSE as shown in block **1608**. Otherwise the loading process **1404** continues at block **1610**.

[0136] 5. Block **1610** shows a check of the kernel dump flag. If this flag indicates that a full kernel memory dump or the like is possible then the kernel secure flag may be set to FALSE as shown in block **1608**. Otherwise the loading process **1404** continues at block **1612**. Not shown in the diagram, this check may be performed periodically and the state of the kernel secure flag updated accordingly.

[0137] 6. Block **1612** shows the loading of the revocation list into the kernel. In cases where the revocation list may be used to check debug credentials, or other previously loaded credentials, signatures, certificate data, or the like, this step may take place earlier in the sequence (prior to the loading of credentials and the like to be checked) than shown. Not shown in the diagram is that, once this component is loaded, any and all previously loaded kernel components may be checked to see if their signature and/or certificate data has been revoked per the revocation list. If any have been revoked, the kernel secure flag may be set to FALSE and the loading process **1404** continues at block **1614**. Note that a revocation list may or may not be loaded into the kernel to be used in the creation and maintenance of a protected environments.

[0138] 7. Block **1614** shows the transition to part **2** of this diagram shown in FIG. **17** and continuing at block **1701**.

[0139] 8. Block **1702** shows a check for any additional kernel components to be loaded. If all components have been loaded then the load process **1404** is usually complete and the kernel secure flag remains in whatever state it was last set to, either TRUE or FALSE. If there are additional kernel components to be loaded the load process **1404** continues at block **1706**.

[0140] 9. Block **1706** shows a check for a valid signature of the next component to be loaded. If the signature is invalid then the kernel secure flag may be set to FALSE as shown in block **1718**. Otherwise the loading process **1404** continues at block **1708**. If no component signature is available the component may be considered insecure and the kernel secure flag may be set to FALSE as shown in block **1718**. Signature validity may be determined by checking for a match on a list of valid signatures and/or by checking whether the signer's identity is a trusted identity. As familiar to those skilled in the security technology area, other methods could also be used to validate component signatures.

[0141] 10. Block **1708** shows a check of the component's certificate data. If the certificate data is invalid then the kernel secure flag may be set to FALSE as shown in block **1718**. Otherwise the loading process **1404** continues at block **1710**. If no component certificate data is available the component may be considered insecure and the kernel secure flag may be set to FALSE as shown in block **1718**. Certificate data validity may be determined by checking the component's certificate data to see if the component is authorized for secure use. As familiar to those skilled in the art, other methods could also be used to validate component certificate data.

[0142] 11. Block **1710** shows a check of the component's signature against a revocation list. If the signature is present on the list, indicating that it has been revoked, then the kernel secure flag may be set to FALSE as shown in block **1718**. Otherwise the loading process **1404** continues at block **1712**.

[0143] 12. Block **1712** shows a check of the component's certificate data against a revocation. If the certificate data is present on the list, indicating that it has been revoked, then the kernel secure flag may be set to FALSE as shown in block **1718**. Otherwise the loading process **1404** continues at block **1714**.

[0144] 13. Block **1714** shows a check of the component's signature to determine if it is OK for use. This check may be made by inspecting the component's leaf certificate data to see if the component is authorized for secure use. Certain attributes in the certificate data may indicate if the component is approved for protected environment usage. If not the component may not be appropriately signed and the kernel secure flag may be set to FALSE as shown in block **1718**. Otherwise the loading process **1404** continues at block **1716**.

[0145] 14. Block **1716** shows a check of the component's root certificate data. This check may be made by inspecting the component's root certificate data to see if it is listed on a list of trusted root certificates. If not the component may be considered insecure and the kernel secure flag may be set to FALSE as shown in block **1718**. Otherwise the loading process **1404** continues at block **1720**.

[0146] 15. Block **1720** shows the loading of the component into the kernel where it is now considered operational. Then the loading process **1404** returns to block **1702** to check for any further components to be loaded.

[0147] FIG. **18** is a block diagram showing a secure computing environment **1000** loading an application **100** into an exemplary protected environment **230** to form a trusted application that may be resistant to attack. In this example the kernel may be the same as that described in FIG. **15**, has already been loaded and the system **1000** is considered fully operational. At this point, as an example, a user starts media application **100**. The media application **100** may call for the creation of a protected environment **230** for one or more of its processes and/or components to operate within. The protected environment creation process **1406** creates the protected environment **230** and loads the application **100** and/or its components as described below.

[0148] FIG. **19** is a flow diagram showing an exemplary process **1406** for creating a protected environment and loading an application into the protected environment. This process **1406** includes the initial step of creating a secure process followed by validating the software component to be loaded into it and then loading the software component into the new secure process and making it operational. Upon success, the result may be a software component operating in a protected environment supported by a secure kernel. Such a software component, along with any digital media content or other data it processes, may be protected from various attacks, including those described above.

[0149] 1. Block **1901** shows the start of the protected environment creation process **1406**. This point is usually reached when some application or code calls for a protected environment to operate.

[0150] 2. Block **1902** shows the establishment of a protected environment. While not shown in the diagram, this may be accomplished by requesting the operating system to create a new secure process. Code later loaded and operating in this secure process may be considered to be operating in a protected environment. If the kernel secure flag is set to FALSE then the "create new secure process" request may fail. This

may be because the system as a whole is considered insecure and unsuitable for a protected environment and any application or data requiring a protected environment. Alternatively, the "create new secure process" request may succeed and the component loaded into the new process may be informed that the system is considered insecure so that it can modify its operations accordingly. Otherwise the process **1406** continues at block **1906**.

[0151] 3. Block **1906** shows a check for a valid signature of the software component to be loaded into the new secure process or protected environment. If the signature is invalid then the process **1406** may fail as shown in block **1918**. Otherwise the process **1406** continues at block **1908**. Not shown in the process is that the program, or its equivalent, creating the new secure process may also be checked for a valid signature and the like. Thus, for either the component itself and/or the program creating the new secure process, if no signature is available the component may be considered insecure and the process **1406** may fail as shown in block **1918**. Signature validity may be determined by checking for a match on a list of valid signatures and/or by checking whether the signer's identity is a trusted identity. As familiar to those skilled in the security technology area, other methods could also be used to validate component signatures.

[0152] 4. Block **1908** shows a check of the software component's certificate data. If the certificate data is invalid then the process **1406** may fail as shown in block **1918**. Otherwise the process **1406** continues at block **1910**. If no component certificate data is available the component may be considered insecure and the process **1406** may fail as shown in block **1918**. Certificate data validity may be determined by checking the component's certificate data to see if the component is authorized for secure use. As familiar to those skilled in the art, other methods could also be used to validate component certificate data.

[0153] Block **1910** shows a check of the component's signature against a revocation list. If the signature is present on the list, indicating that it has been revoked, then the process **1406** may fail as shown in block **1918**. Otherwise the process **1406** continues at block **1912**.

[0154] 12. Block **1912** shows a check of the component's certificate data against the revocation list. If the certificate data is present on the list, indicating that it has been revoked, then the process **1406** may fail as shown in block **1918**. Otherwise the process **1406** continues at block **1914**.

[0155] 13. Block **1914** shows a check of the component's signature to determine if it is acceptable for use. This check may be made by inspecting the component's leaf certificate data to see if the component is authorized for secure use. Certain attributes in the certificate data may indicate if the component is approved for protected environment usage. If not the component may be considered to not be appropriately signed and the process **1406** may fail as shown in block **1918**. Otherwise the process **1406** continues at block **1916**.

[0156] 14. Block **1916** shows a check of the component's root certificate data. This check may be made by inspecting the component's root certificate data to see if it is listed on a list of trusted root certificates. If not the component may be considered insecure and the process **1406** may fail as shown in block **1918**. Otherwise the process **1406** continues at block **1920**.

[0157] 15. Block **1918** shows the failure of the software component to load followed by block **1930**, the end of the protected environment creation process **1406**.

[0158] 16. Block **1920** shows the software component being loaded into the protected environment, where it is considered operational, followed by block **1930**, the end of the protected environment creation process **1406**.

[0159] FIG. **20** is a block diagram showing an exemplary trusted application utilizing an exemplary protected environment **230** periodically checking **1408** the security state **1590** of the secure computing environment **1000**. In this example, the computing environment **1000** and the kernel **1550** may be the same as those described in FIG. **15** and FIG. **16**. The kernel **1550** has already been loaded and the computer **1000** is considered fully operational. Further, a protected environment has been created and the appropriate components of the trusted application have been loaded into it and made operational, establishing a trusted application utilizing a protected environment **230**, hereafter referred to simply as the "protected environment".

[0160] The protected environment **230** may periodically check with the PE management portion of the kernel **1552** to determine whether the kernel **1550** remains secure over time. This periodic check may be performed because it is possible for a new component to be loaded into the kernel **1550** at any time, including a component that may be considered insecure. If this were to occur, the state of the kernel secure flag **1590** may change to FALSE and the code operating in the protected environment **230** has the opportunity to respond appropriately.

[0161] For example, consider a media player application that was started on a PC **1000** with a secure kernel **1550** and a portion of the media player application operating in a protected environment **230** processing digital media content that is licensed only for secure use. In this example, if a new kernel component that is considered insecure is loaded while the media player application is processing the media content, then the check kernel secure state process **1040** would note the kernel secure flag **1590** has changed to FALSE indicating the kernel **1550** may no longer be secure.

[0162] Alternatively, the revocation list **1545** may be updated and a kernel component previously considered secure may no longer be considered secure, resulting in the kernel secure flag **1590** being set to FALSE. At this point the application may receive notification that the system **1000** is no longer considered secure and can terminate operation, or take other appropriate action to protect itself and/or the media content it is processing.

[0163] FIG. **21** is a flow diagram showing an exemplary process **1408** for periodically checking the security state of the secure computing environment. This process **1408** may be used by a protected environment **230** to determine if the kernel remains secure over time. The protected environment **230** may periodically use this process **1408** to check the current security status of the kernel. The protected environment **230** and/or the software component operating within it may use the current security status information to modify its operation appropriately. Periodic activation of the process may be implemented using conventional techniques.

[0164] The diagram in FIG. **21** shows a sequence of communications **1408**, illustrated with exemplary pseudo code, between the protected environment **230** and the PE management portion of the kernel **1552**. This communication may include a check of the version of a revocation list which may give an application the ability to specify a revocation list of at least a certain version. This communications sequence may be cryptographically secured using conventional techniques.

[0165] 1. The protected environment **230** makes a IsKernelSecure(MinRLVer) call **2120** to the PE management portion of the kernel to query the current security state of the kernel. Included in this call **2120** may be the minimum version (MinRLVer) of the revocation list expected to be utilized.

[0166] 2. The PE management portion of the kernel checks to see if the protected environment, which is the calling process, is secure. If not, then it may provide a Return (SecureFlag=FALSE) indication **2122** to the protected environment and the communications sequence **1408** is complete. This security check may be done by the PE management portion of the kernel checking the protected environment for a valid signature and/or certificate data as described above.

[0167] 3. Otherwise, the PE management portion of the kernel checks the kernel secure flag in response to the call **2120**. If the state of the flag is FALSE then it may provide a Return(SecureFlag=FALSE) indication **2124** to the protected environment and the communications sequence **1408** is complete.

[0168] 4. Otherwise, the PE management portion of the kernel checks the revocation list version information for the revocation list. If the revocation list has version information that is older than that requested in the IsKernelSecure(MinRLVer) call **2120** then several options are possible. First, as indicated in the diagram, the PE management portion of the kernel may provide a Return(SecureFlag=FALSE) indication **2126** to the protected environment and the communications sequence **1408** is complete.

[0169] Alternatively, and not shown in the diagram, an appropriate version revocation list may be located and utilized, all kernel components may be re-validated using this new or updated list, the kernel secure flag updated as appropriate and the previous step #**3** of this communications sequence **1408** repeated.

[0170] 5. Otherwise, the PE management portion of the kernel may provide a Return(SecureFlag=TRUE) indication **2128** to the protected environment and the communications sequence **1408** is complete.

[0171] FIG. **22** is a block diagram showing an exemplary computing environment **800** including a representation of a protected environment **230**, a trusted media system **200**, and other related elements. Exemplary personal computer **800** is similar to that shown in FIG. **8** with the addition of kernel components **1520-1530** that may be stored on the disk **810** along with the other operating system code and the like. Media application **100** and/or a digital rights management system **1004** may be stored on the disk **810** along with other application programs. These components **1520-1530** and applications **100**, **1004** may be loaded into system memory **809** and considered operational. Shown loaded in system memory **809** is a trusted application **200** utilizing a protected environment **230** and media content **110**.

1. A computing device comprising:

a processor;

memory coupled to the processor;

a protected media pipeline implemented at least in part by the processor and the memory, the protected media pipeline configured to process digital media, the protected media pipeline comprising:

a media source coupled to a first secure connection over which the digital media is received via the media source into the protected media pipeline; and

a media sink coupled to a second secure connection over which processed digital media is transferred via the media sink out of the protected media pipeline.

2. The computing device of claim **1** coupled to a protected input via the first secure connection and via which the digital media is initially received into the computing device, where the protected input is configured to limit unauthorized access to the digital media.

3. The computing device of claim **1** coupled to a protected output via the second secure connection and via which the processed digital media is transferred from the computing device, where the protected output is configured to limit unauthorized access to the processed digital media.

4. The computing device of claim **1** where the first secure connection is configured to limit unauthorized access to the digital media.

5. The computing device of claim **1** where the second secure connection is configured to limit unauthorized access to the processed digital media.

6. The computing device of claim **1**, the protected media pipeline further comprising a protected space that includes at least one transform mechanism that is configured to transform the digital media, where the protected space is configured to limit unauthorized access to the digital media and the processed digital media.

7. The computing device of claim **1**, where the protected media pipeline is configured to limit unauthorized access to the digital media and the processed digital media.

8. A method performed on a computing device that comprises a processor, memory, and a protected media pipeline that is configured to process digital media, the method comprising:

receiving, by a media source over a first secure connection into the protected media pipeline, the digital media, where the media source is a component of the protected media pipeline;

transferring, by a media sink over a second secure connection out of the protected media pipeline, processed digital media, where the media sink is a component of the protected media pipeline.

9. The method of claim **9**, the receiving based on the computing device being coupled to a protected input via the first secure connection and via which the digital media is initially received into the computing device, where the protected input is configured to limit unauthorized access to the digital media.

10. The method of claim **9**, the transferring based on the computing device being coupled to a protected output via the second secure connection and via which the processed digital media is transferred from the computing device, where the protected output is configured to limit unauthorized access to the processed digital media.

11. The method of claim **9** where the first secure connection is configured to limit unauthorized access to the digital media.

12. The method of claim **9** where the second secure connection is configured to limit unauthorized access to the processed digital media.

13. The method of claim **9** further comprising transforming, by at least one transform mechanism, the digital media, where the at least one transform mechanism is a component of a protected space of the protected media pipeline, where the protected space is configured to limit unauthorized access to the digital media and the processed digital media.

14. The method of claim **9**, where the protected media pipeline is configured to limit unauthorized access to the digital media and the processed digital media.

15. At least on computer-readable media that comprises:

memory that includes computer-executable instructions that, based on execution by a computing device that comprises a protected media pipeline that is configured to process digital media, configure the computing device to perform actions comprising:

receiving, via a media source over a first secure connection into the protected media pipeline, the digital media, where the media source is a component of the protected media pipeline;

transferring, via a media sink over a second secure connection out of the protected media pipeline, processed digital media, where the media sink is a component of the protected media pipeline.

16. The at least on computer-readable media of claim **15**, the receiving based on the computing device being coupled to a protected input via the first secure connection and via which the digital media is initially received into the computing device, where the protected input is configured to limit unauthorized access to the digital media.

17. The at least on computer-readable media of claim **15**, the transferring based on the computing device being coupled to a protected output via the second secure connection and via which the processed digital media is transferred from the computing device, where the protected output is configured to limit unauthorized access to the processed digital media.

18. The at least on computer-readable media of claim **15** where the first secure connection is configured to limit unauthorized access to the digital media, or where the second secure connection is configured to limit unauthorized access to the processed digital media.

19. The at least on computer-readable media of claim **15**, the actions further comprising transforming, by at least one transform mechanism, the digital media, where the at least one transform mechanism is a component of a protected space of the protected media pipeline, where the protected space is configured to limit unauthorized access to the digital media and the processed digital media.

20. The at least on computer-readable media of claim **15**, where the protected media pipeline is configured to limit unauthorized access to the digital media and the processed digital media.

* * * * *