US 20060020796A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2006/0020796 A1

Aura et al. (43) Pub. Date: Jan. 26, 2006

(54) HUMAN INPUT SECURITY CODES

(75) Inventors: **Tuomas Aura**, Cambridge (GB); **Michael Roe**, Cambridge (GB)

Correspondence Address:
MICROSOFT CORPORATION
ATTN: PATENT GROUP DOCKETING DEPARTMENT
ONE MICROSOFT WAY
REDMOND, WA 98052-6399 (US)

(73) Assignee: **Microsoft Corporation**, Redmond, WA

(21) Appl. No.: **11/170,296**

(22) Filed: **Jun. 28, 2005**

Related U.S. Application Data

(63) Continuation-in-part of application No. 10/401,241, filed on Mar. 27, 2003.

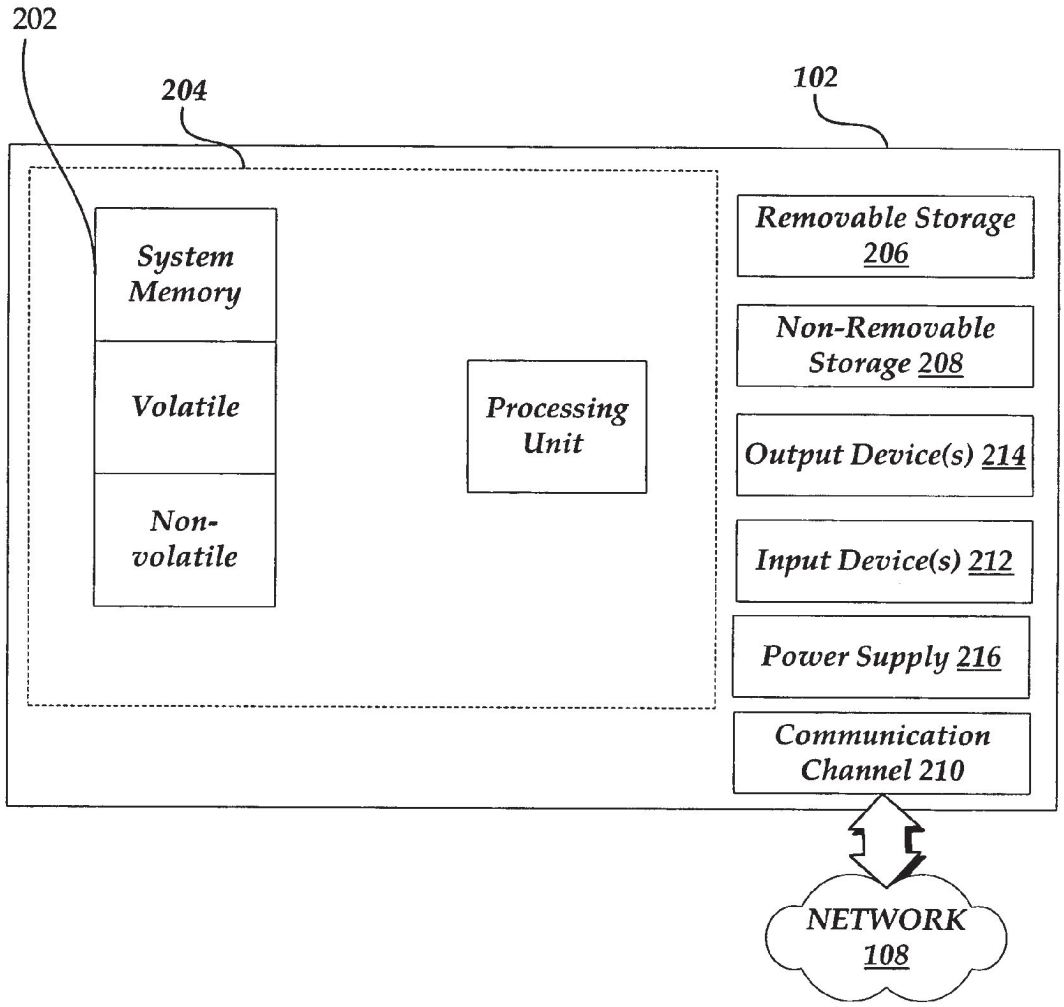
Publication Classification

(51) **Int. Cl.**
H04L 9/00 (2006.01)

(52) **U.S. Cl.** **713/168**

(57) **ABSTRACT**

The hash extension technique used to generate an ECGA may be used to increase the strength of one-way hash functions and/or decrease the number of bits in any situation where some external requirement limits the number of hash bits, and that limit is below what is (or may be in the future) considered secure against brute-force attacks. For example, to decrease the length of human entered security codes (and maintain the same security), and/or to increase the strength of a human entered security code (and maintain the length of the security code), the security code may be generated and/or authenticated using an extended hash method.



OKTA, INC. 1030

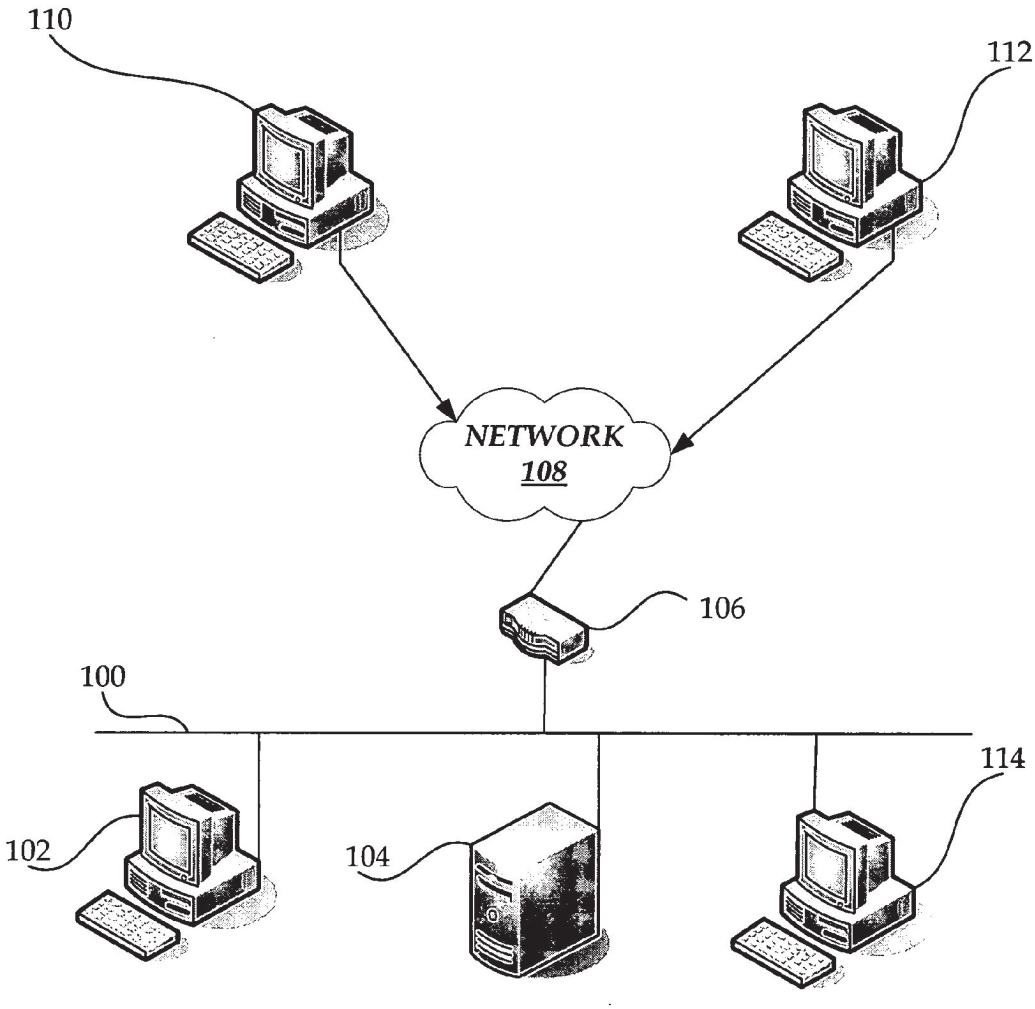


FIG. 1

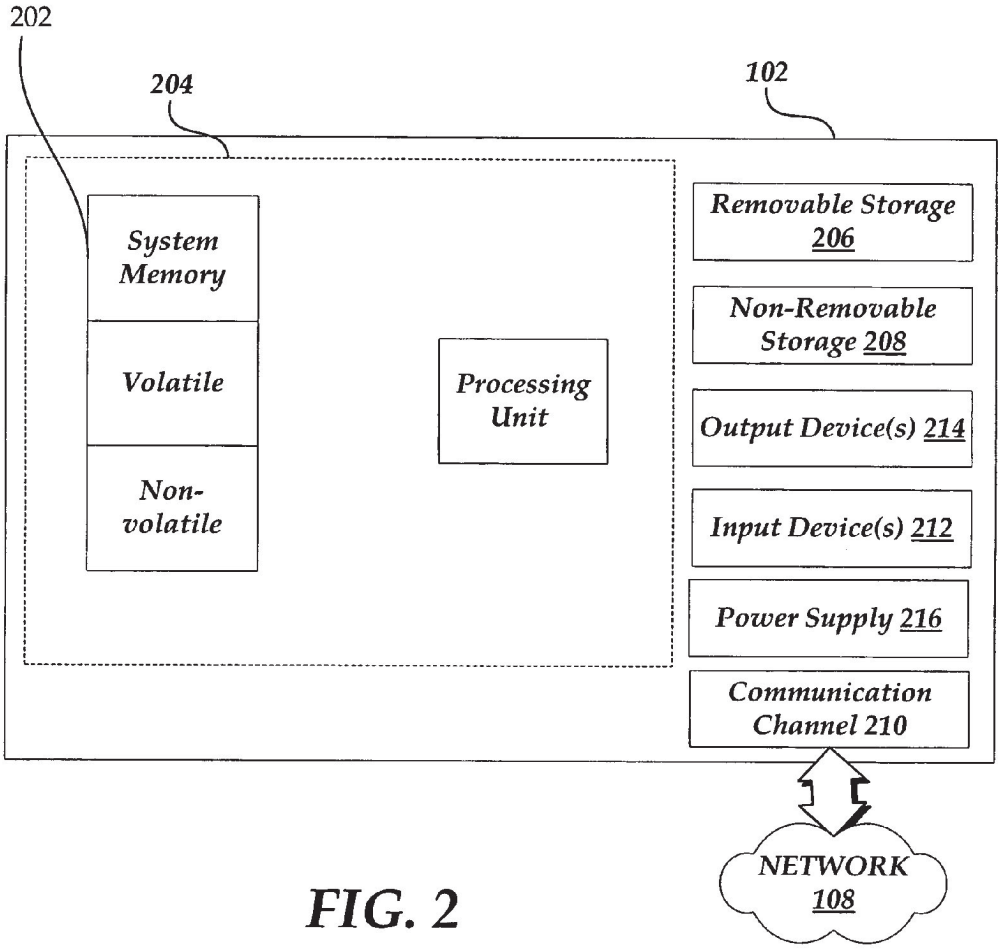

202
204
102
System Memory
Volatile
Non-volatile
Processing Unit
Removable Storage 206
Non-Removable Storage 208
Output Device(s) 214
Input Device(s) 212
Power Supply 216
Communication Channel 210
NETWORK 108

FIG. 2

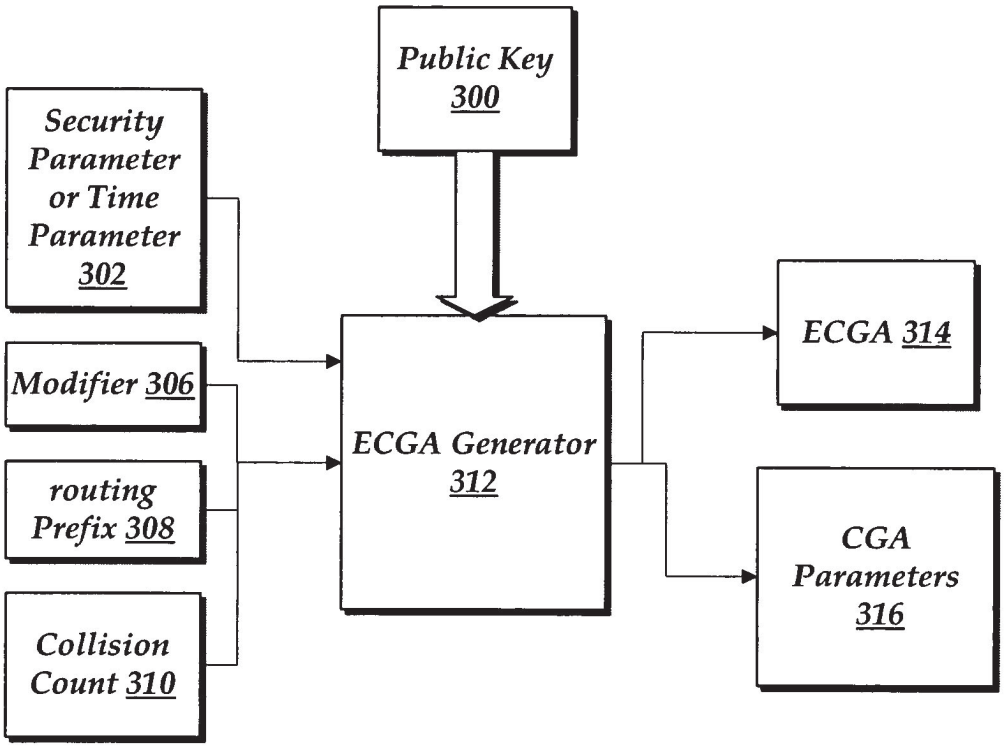

Public Key 300
Security Parameter or Time Parameter 302
Modifier 306
routing Prefix 308
Collision Count 310
ECGA Generator 312
ECGA 314
CGA Parameters 316

FIG. 3

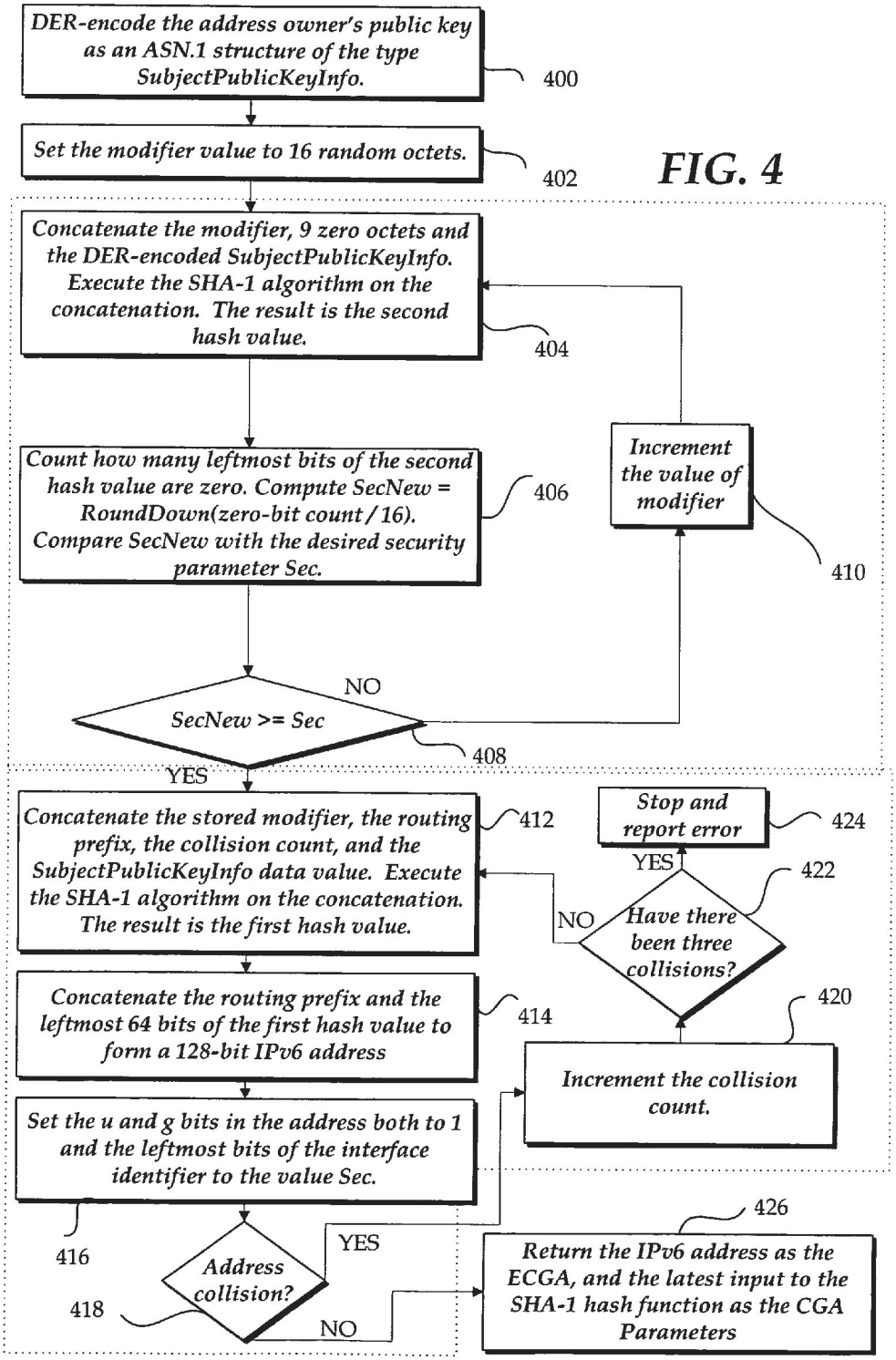

DER-encode the address owner's public key as an ASN.1 structure of the type SubjectPublicKeyInfo.
400
Set the modifier value to 16 random octets.
402
Concatenate the modifier, 9 zero octets and the DER-encoded SubjectPublicKeyInfo. Execute the SHA-1 algorithm on the concatenation. The result is the second hash value.
404
Count how many leftmost bits of the second hash value are zero. Compute SecNew = RoundDown(zero-bit count / 16). Compare SecNew with the desired security parameter Sec.
406
Increment the value of modifier
410
NO
SecNew >= Sec
408
YES
Concatenate the stored modifier, the routing prefix, the collision count, and the SubjectPublicKeyInfo data value. Execute the SHA-1 algorithm on the concatenation. The result is the first hash value.
412
Stop and report error
424
YES
422
NO
Have there been three collisions?
Concatenate the routing prefix and the leftmost 64 bits of the first hash value to form a 128-bit IPv6 address
414
420
Increment the collision count.
Set the u and g bits in the address both to 1 and the leftmost bits of the interface identifier to the value Sec.
416
426
YES
Address collision?
418
NO
Return the IPv6 address as the ECGA, and the latest input to the SHA-1 hash function as the CGA Parameters

FIG. 4

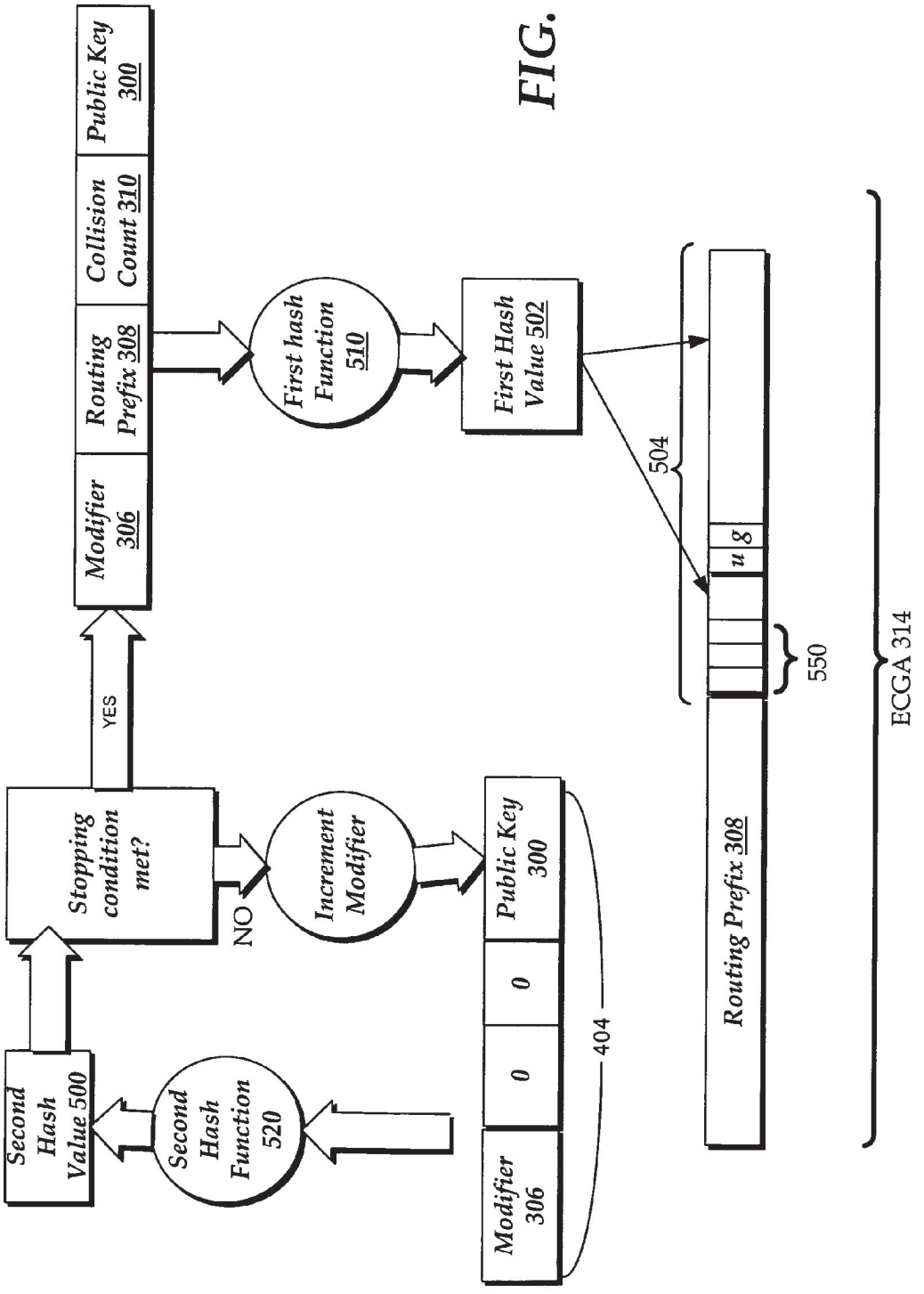

Second Hash Value 500
Second Hash Function 520
Stopping condition met?
YES
NO
Increment Modifier
Modifier 306
0
0
Public Key 300
404
Modifier 306
Routing Prefix 308
Collision Count 310
Public Key 300
First hash Function 510
First Hash Value 502
504
Routing Prefix 308
u
g
550
ECGA 314

FIG. 5

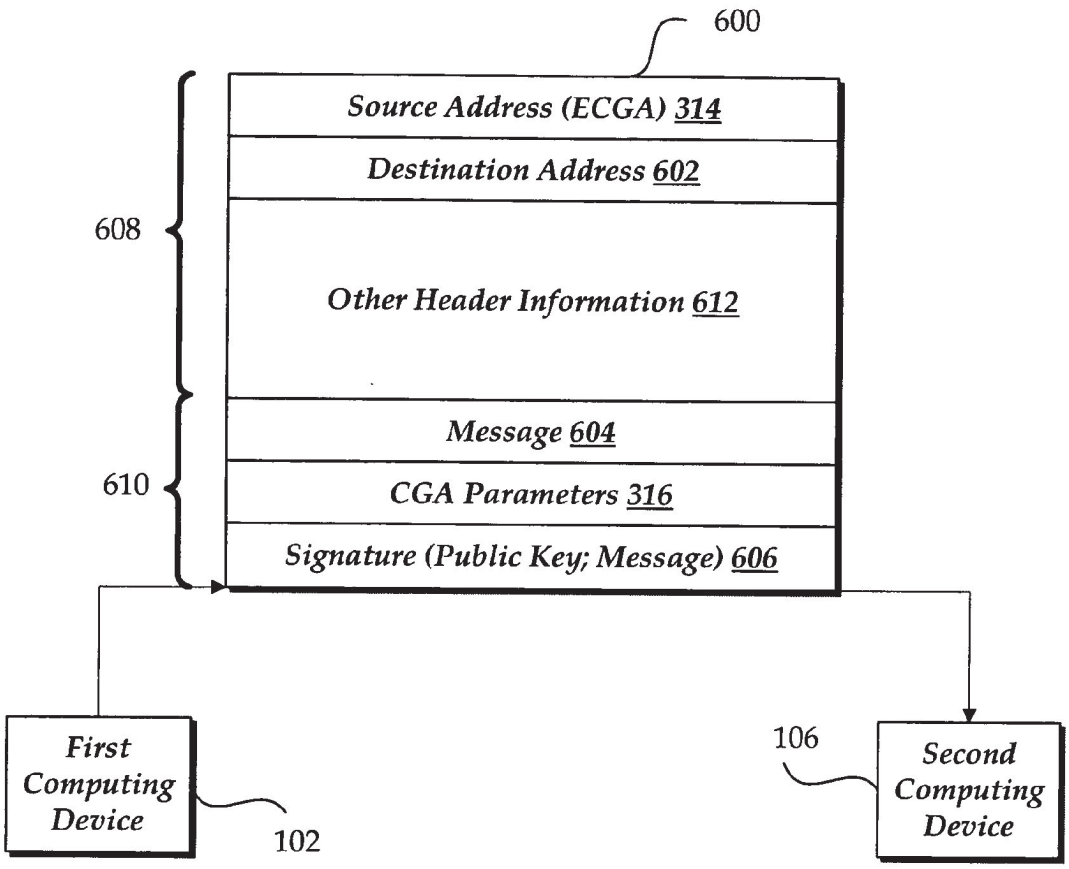

600
Source Address (ECGA) 314
Destination Address 602
608
Other Header Information 612
Message 604
610
CGA Parameters 316
Signature (Public Key; Message) 606
First Computing Device
102
106
Second Computing Device

FIG. 6

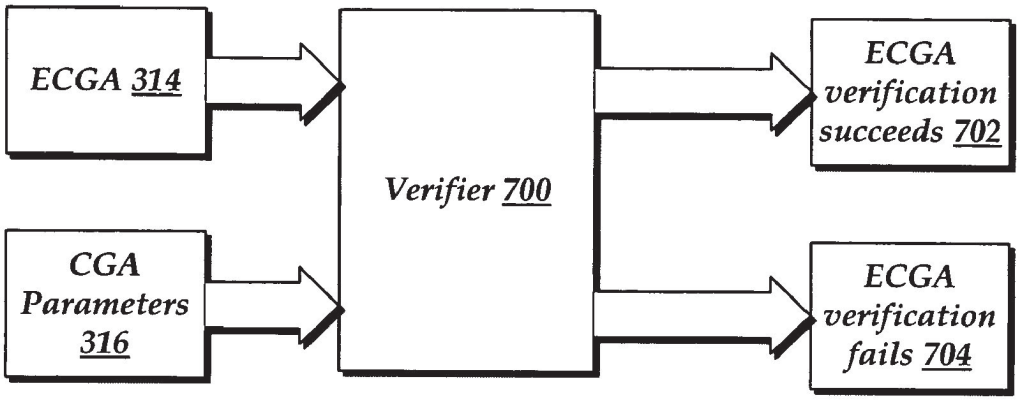

ECGA 314
CGA Parameters 316
Verifier 700
ECGA verification succeeds 702
ECGA verification fails 704

FIG. 7

FIG. 8

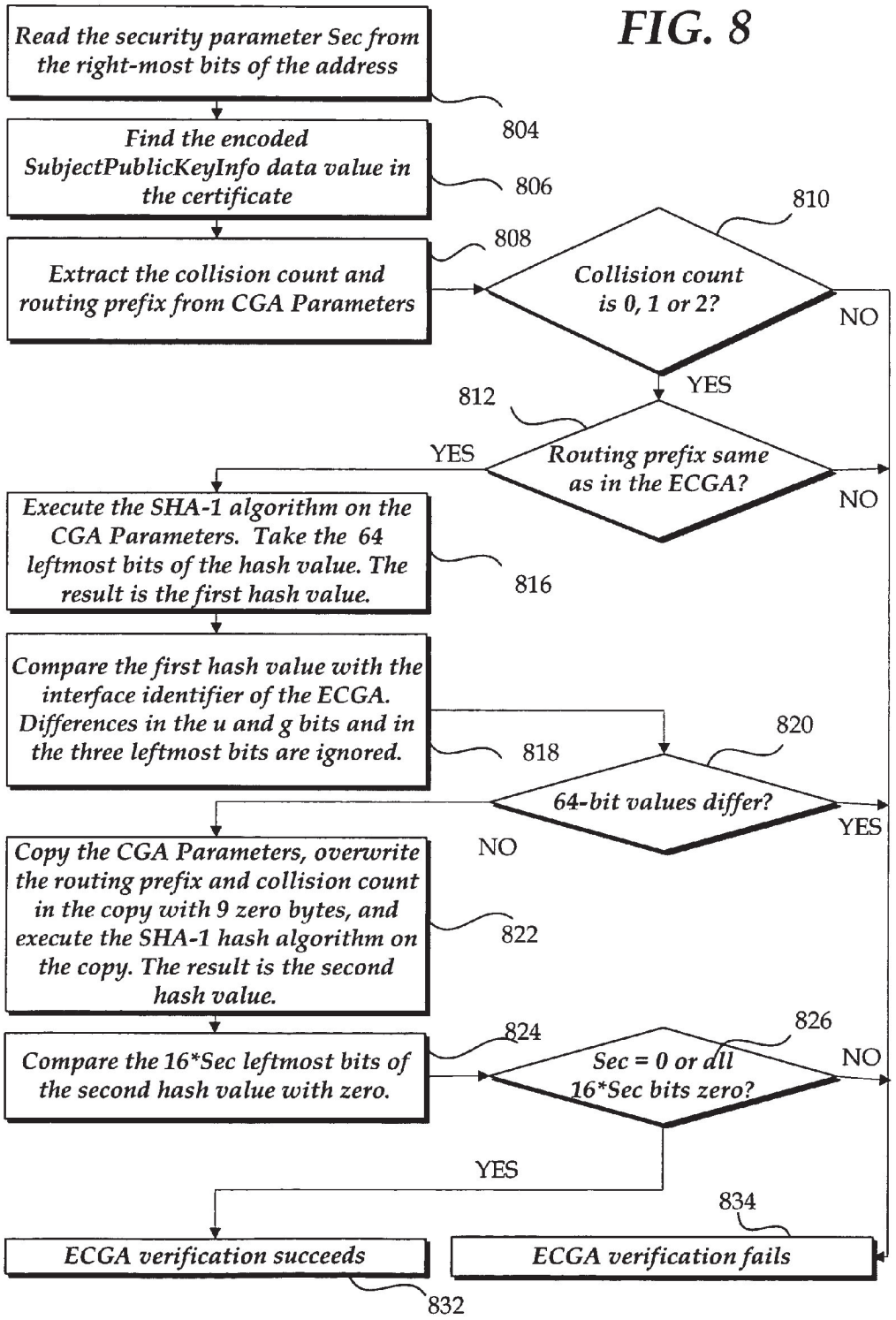
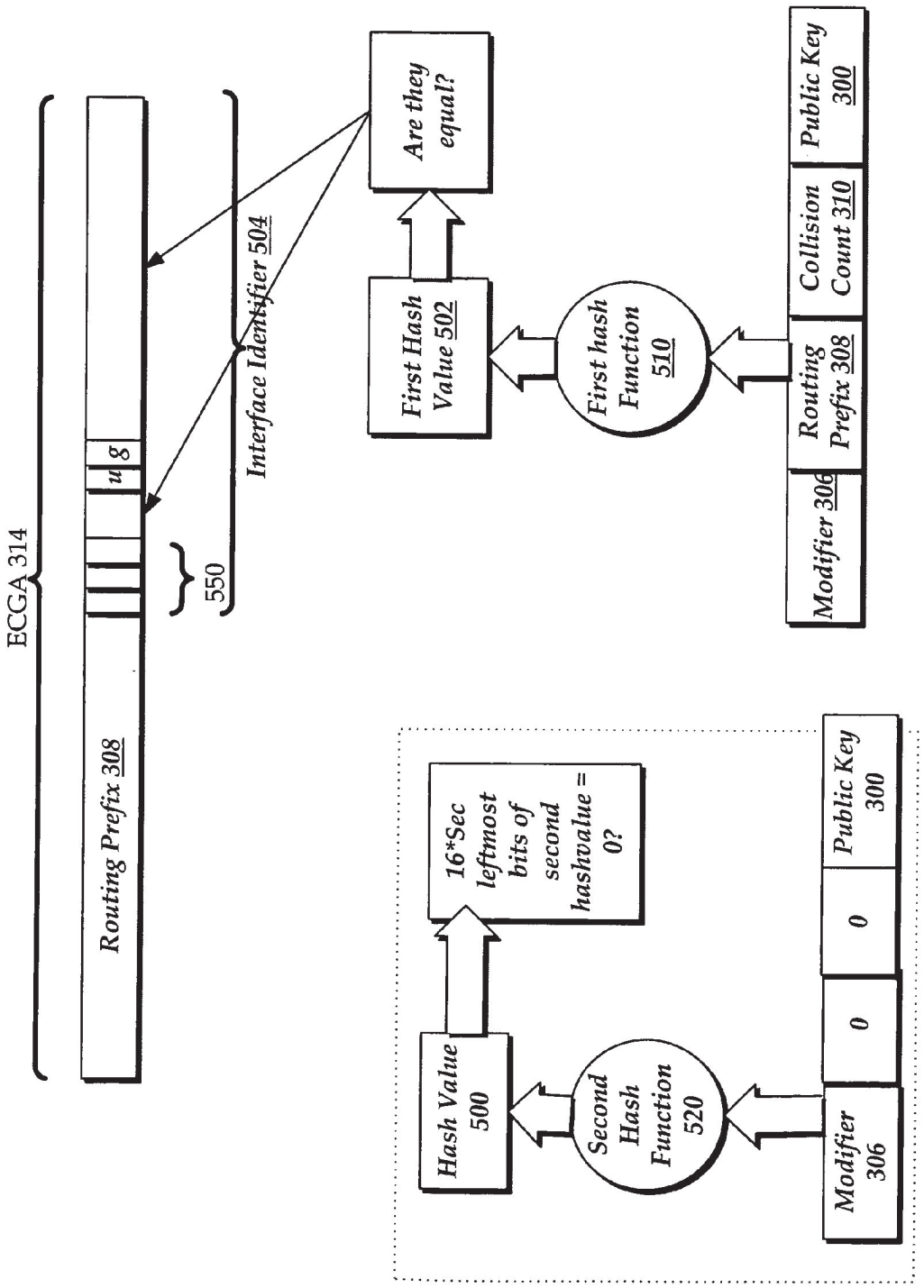

Read the security parameter Sec from the right-most bits of the address
804
Find the encoded SubjectPublicKeyInfo data value in the certificate
806
Extract the collision count and routing prefix from CGA Parameters
808
Collision count is 0, 1 or 2?
810
NO
YES
Routing prefix same as in the ECGA?
812
YES
NO
Execute the SHA-1 algorithm on the CGA Parameters. Take the 64 leftmost bits of the hash value. The result is the first hash value.
816
Compare the first hash value with the interface identifier of the ECGA. Differences in the u and g bits and in the three leftmost bits are ignored.
818
64-bit values differ?
820
YES
NO
Copy the CGA Parameters, overwrite the routing prefix and collision count in the copy with 9 zero bytes, and execute the SHA-1 hash algorithm on the copy. The result is the second hash value.
822
Compare the 16*Sec leftmost bits of the second hash value with zero.
824
Sec = 0 or all 16*Sec bits zero?
826
NO
YES
ECGA verification succeeds
832
ECGA verification fails
834

FIG. 9


ECGA 314
Routing Prefix 308
u
g
550
Interface Identifier 504
Are they equal?
First Hash Value 502
First hash Function 510
Modifier 306
Routing Prefix 308
Collision Count 310
Public Key 300
16*Sec leftmost bits of second hashvalue = 0?
Hash Value 500
Second Hash Function 520
Modifier 306
0
0
Public Key 300

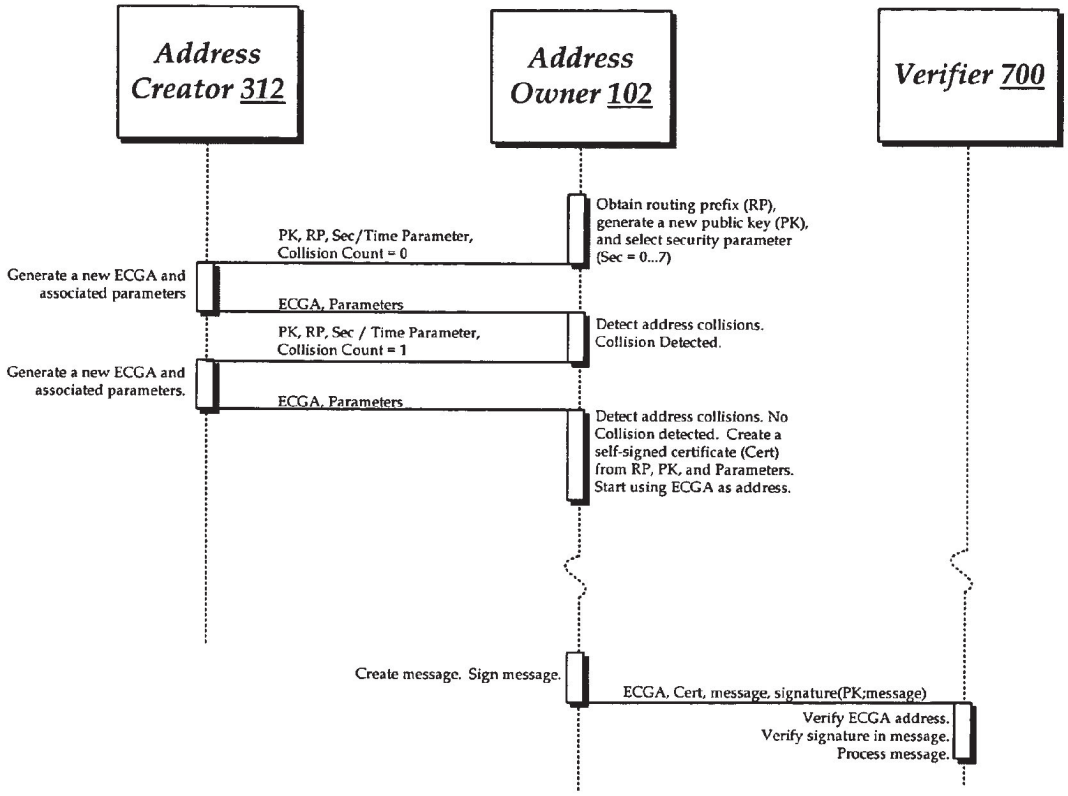

Address Creator 312
Address Owner 102
Verifier 700
Obtain routing prefix (RP), generate a new public key (PK), and select security parameter (Sec = 0...7)
PK, RP, Sec/Time Parameter, Collision Count = 0
Generate a new ECGA and associated parameters
ECGA, Parameters
Detect address collisions. Collision Detected.
PK, RP, Sec / Time Parameter, Collision Count = 1
Generate a new ECGA and associated parameters.
ECGA, Parameters
Detect address collisions. No Collision detected. Create a self-signed certificate (Cert) from RP, PK, and Parameters. Start using ECGA as address.
Create message. Sign message.
ECGA, Cert, message, signature(PK;message)
Verify ECGA address.
Verify signature in message.
Process message.

FIG. 10

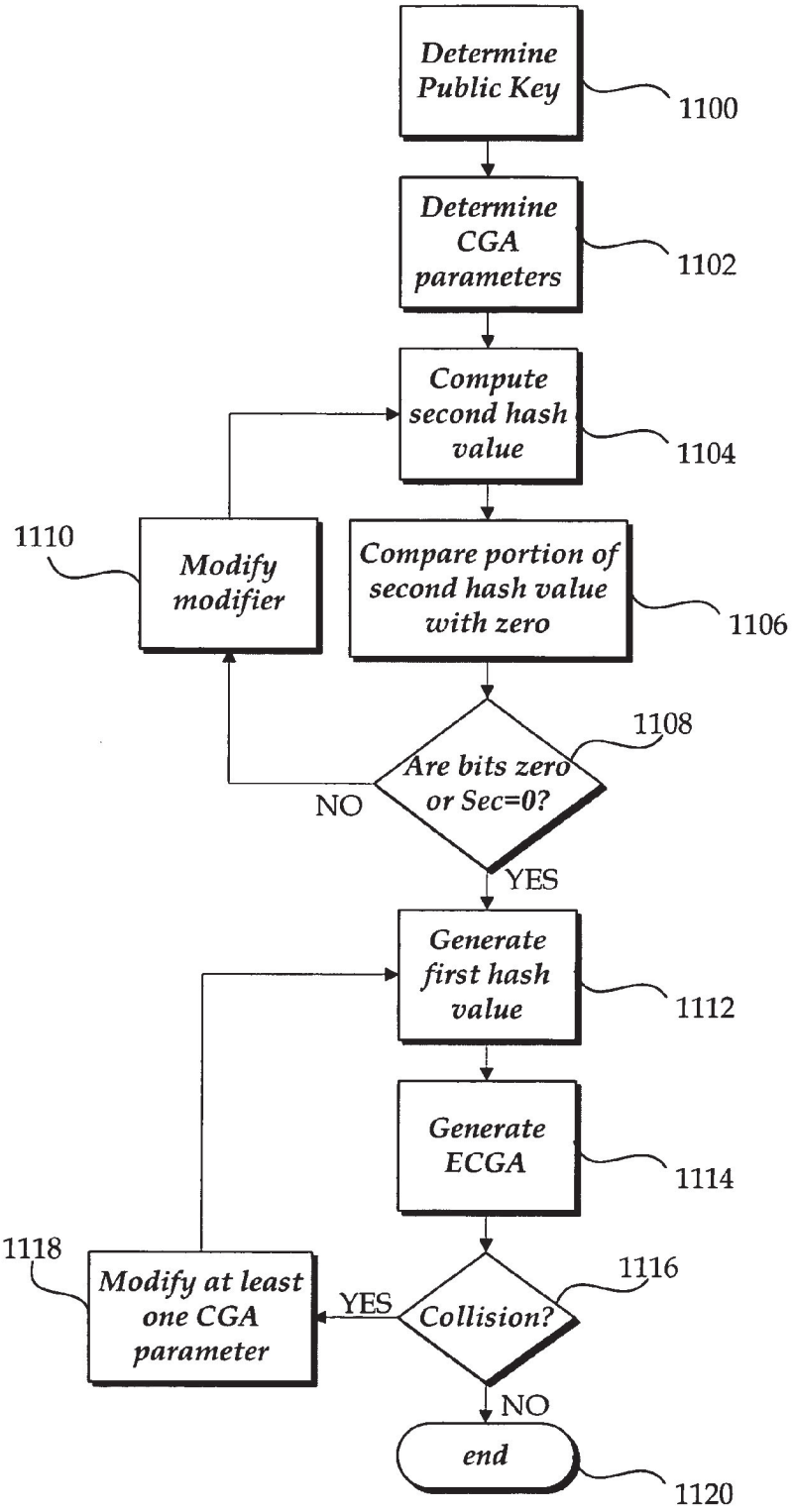

Determine Public Key
1100
Determine CGA parameters
1102
Compute second hash value
1104
1110
Modify modifier
Compare portion of second hash value with zero
1106
1108
Are bits zero or Sec=0?
NO
YES
Generate first hash value
1112
Generate ECGA
1114
1118
Modify at least one CGA parameter
YES
Collision?
1116
NO
end
1120

FIG. 11

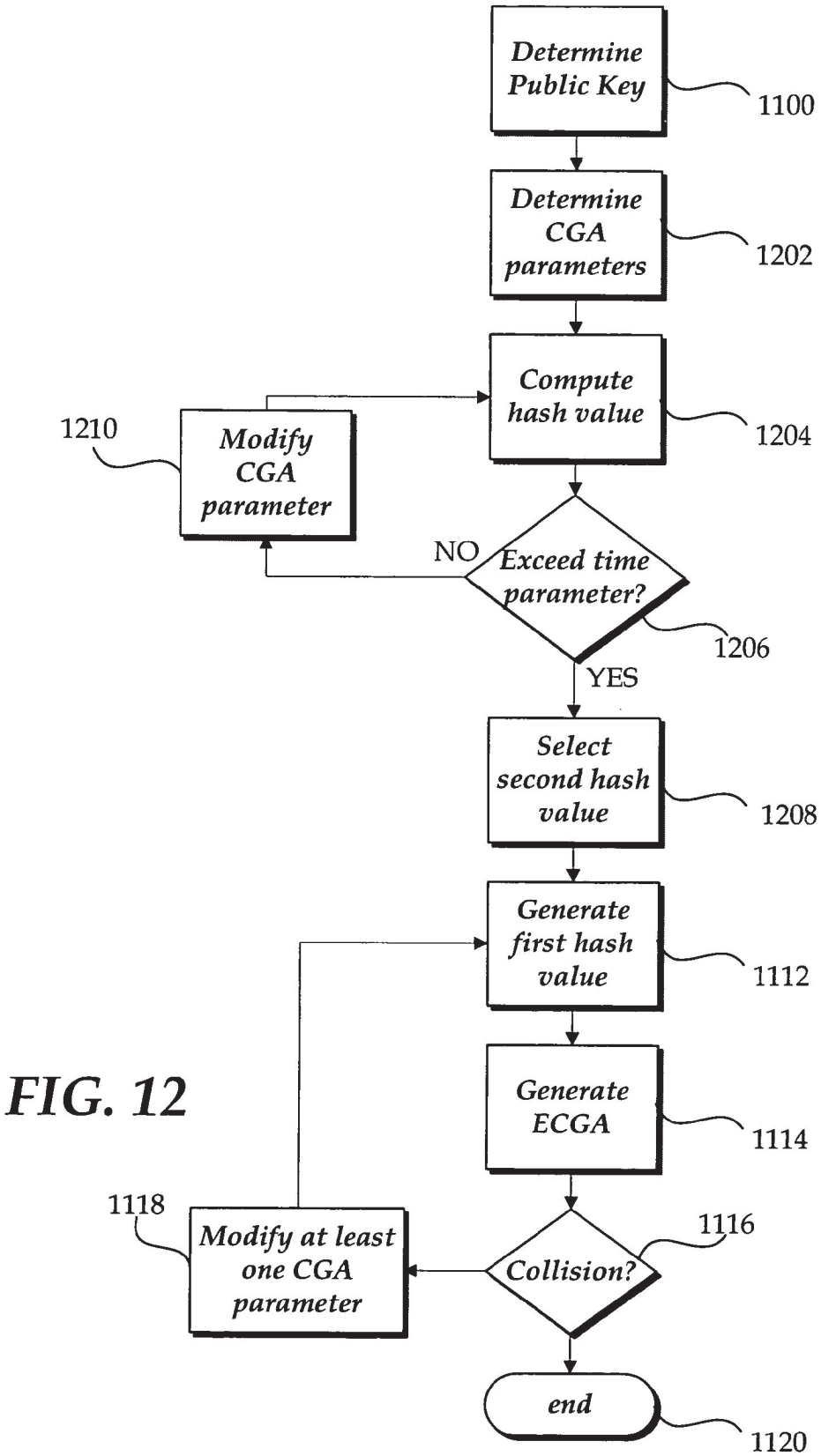

Determine Public Key
1100
Determine CGA parameters
1202
Compute hash value
1204
1210
Modify CGA parameter
NO
Exceed time parameter?
1206
YES
Select second hash value
1208
Generate first hash value
1112
Generate ECGA
1114
1118
Modify at least one CGA parameter
Collision?
1116
end
1120

FIG. 12

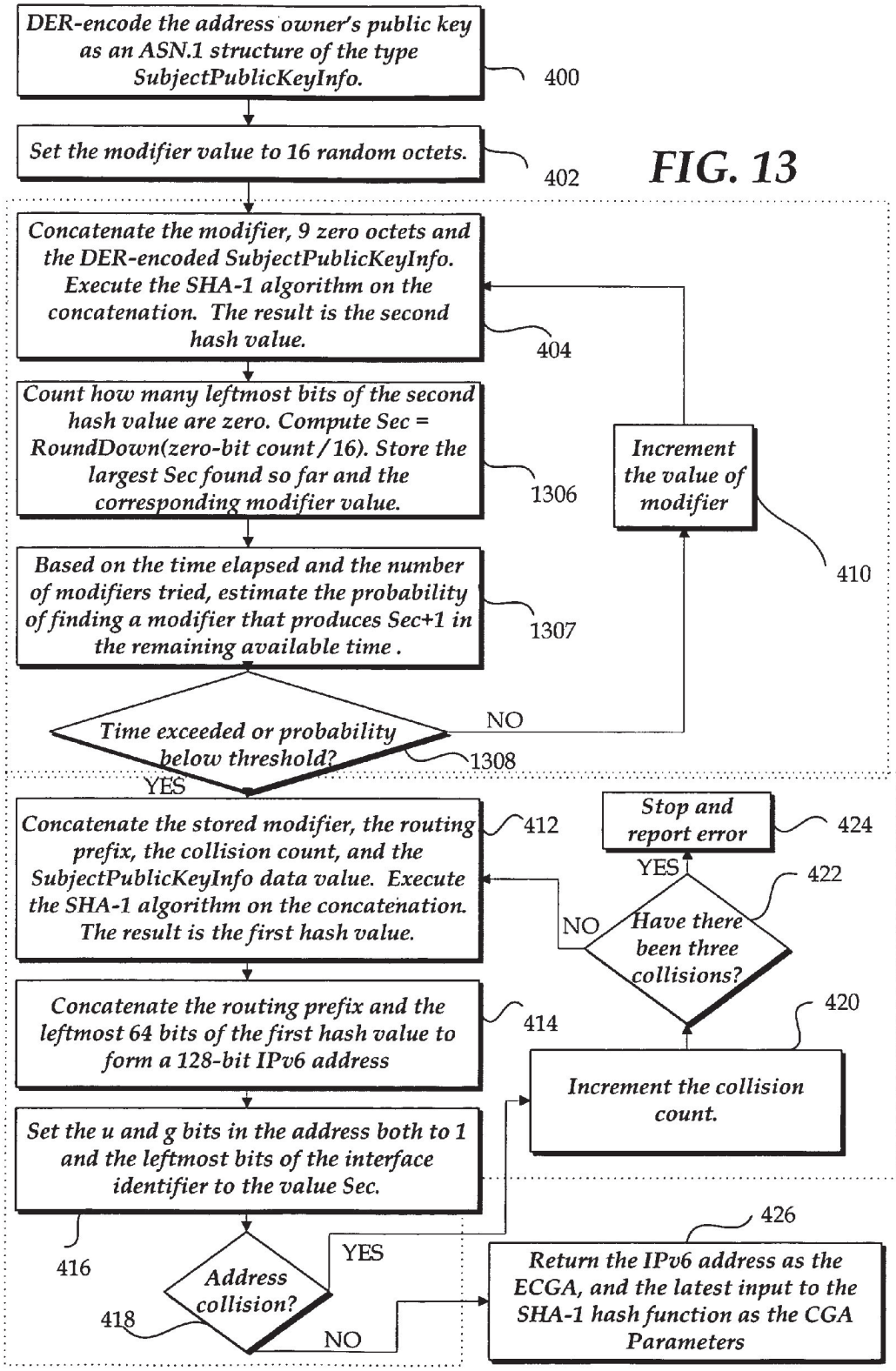

DER-encode the address owner's public key as an ASN.1 structure of the type SubjectPublicKeyInfo.
400
Set the modifier value to 16 random octets.
402
Concatenate the modifier, 9 zero octets and the DER-encoded SubjectPublicKeyInfo. Execute the SHA-1 algorithm on the concatenation. The result is the second hash value.
404
Count how many leftmost bits of the second hash value are zero. Compute Sec = RoundDown(zero-bit count / 16). Store the largest Sec found so far and the corresponding modifier value.
1306
Based on the time elapsed and the number of modifiers tried, estimate the probability of finding a modifier that produces Sec+1 in the remaining available time .
1307
Time exceeded or probability below threshold?
1308
NO
YES
Increment the value of modifier
410
Concatenate the stored modifier, the routing prefix, the collision count, and the SubjectPublicKeyInfo data value. Execute the SHA-1 algorithm on the concatenation. The result is the first hash value.
412
Concatenate the routing prefix and the leftmost 64 bits of the first hash value to form a 128-bit IPv6 address
414
Set the u and g bits in the address both to 1 and the leftmost bits of the interface identifier to the value Sec.
416
Address collision?
418
YES
NO
Stop and report error
424
YES
Have there been three collisions?
422
NO
Increment the collision count.
420
Return the IPv6 address as the ECGA, and the latest input to the SHA-1 hash function as the CGA Parameters
426

FIG. 13

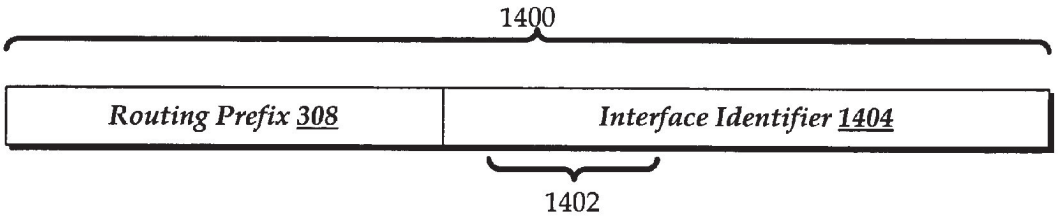


FIG. 14

FIG. 15

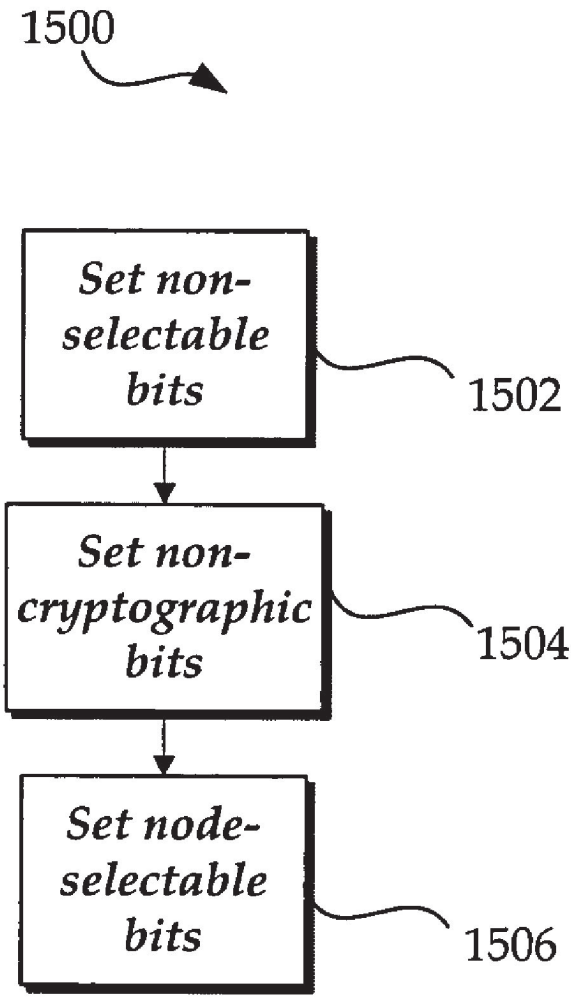

1500
Set non-selectable bits
1502
Set non-cryptographic bits
1504
Set node-selectable bits
1506

FIG. 16

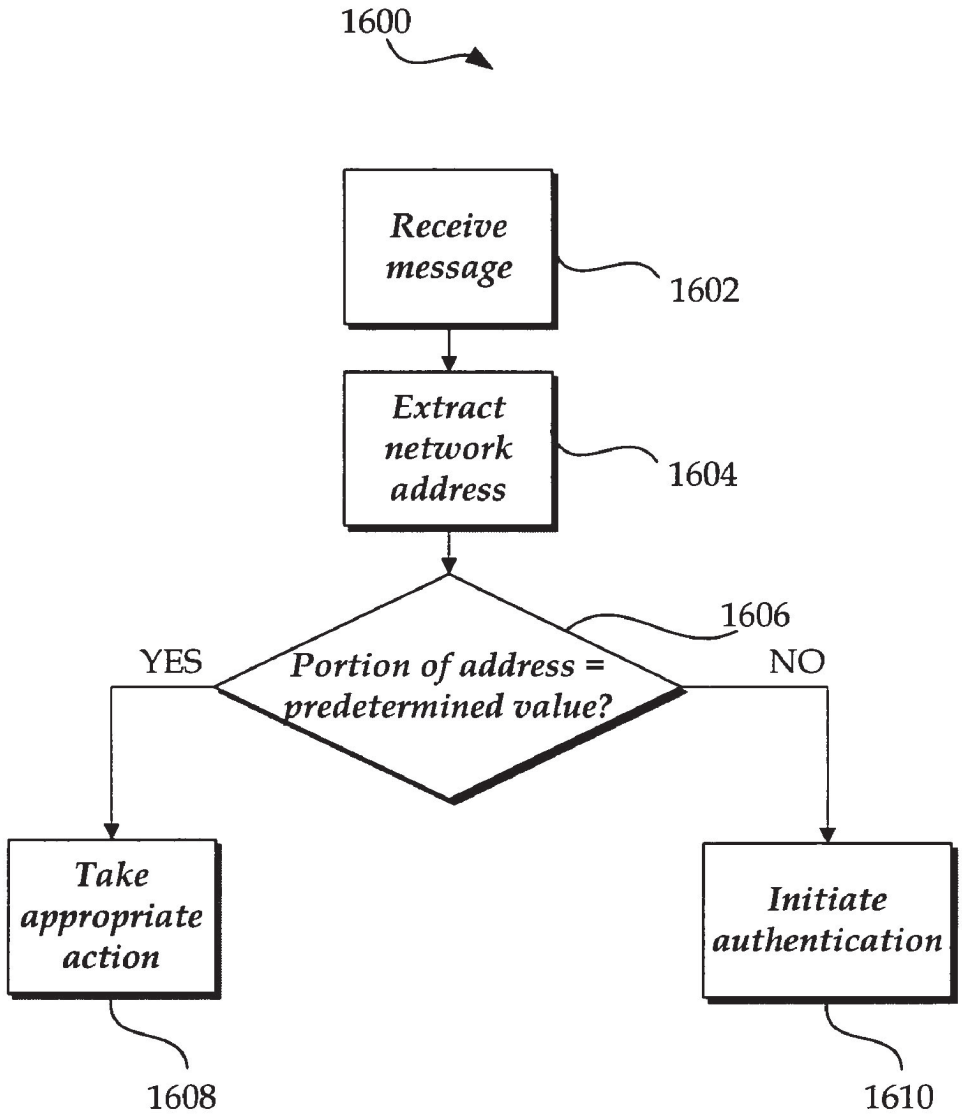

1600
Receive message
1602
Extract network address
1604
1606
YES
Portion of address = predetermined value?
NO
Take appropriate action
1608
Initiate authentication
1610

FIG. 17

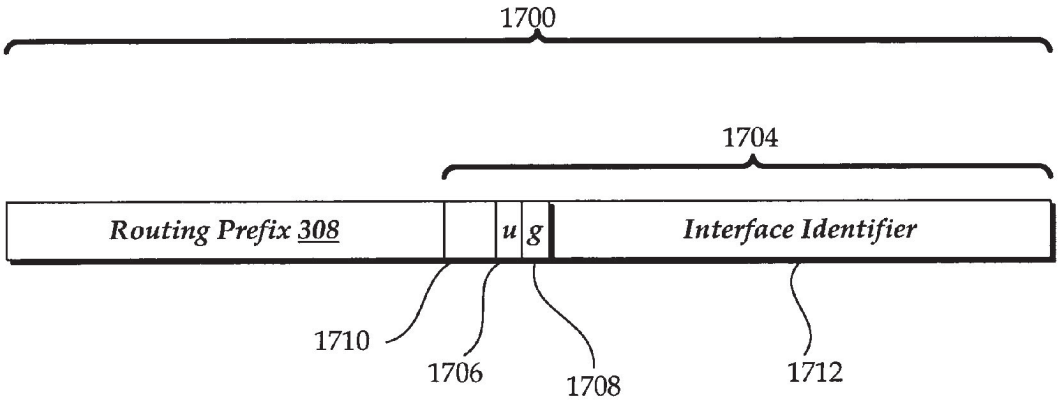

1700
1704
Routing Prefix 308
u
g
Interface Identifier
1710
1706
1708
1712

FIG. 18

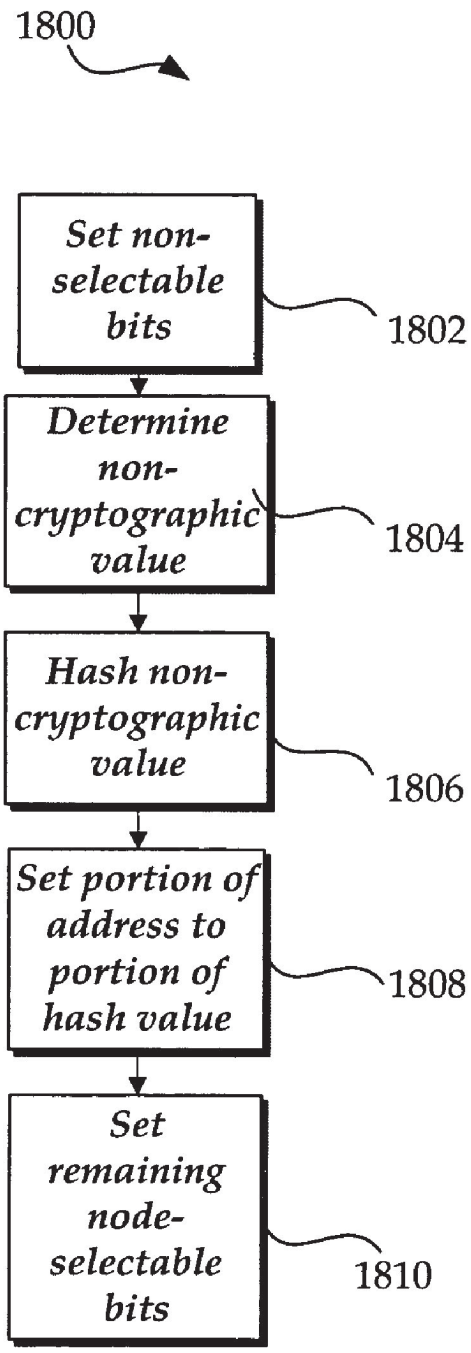
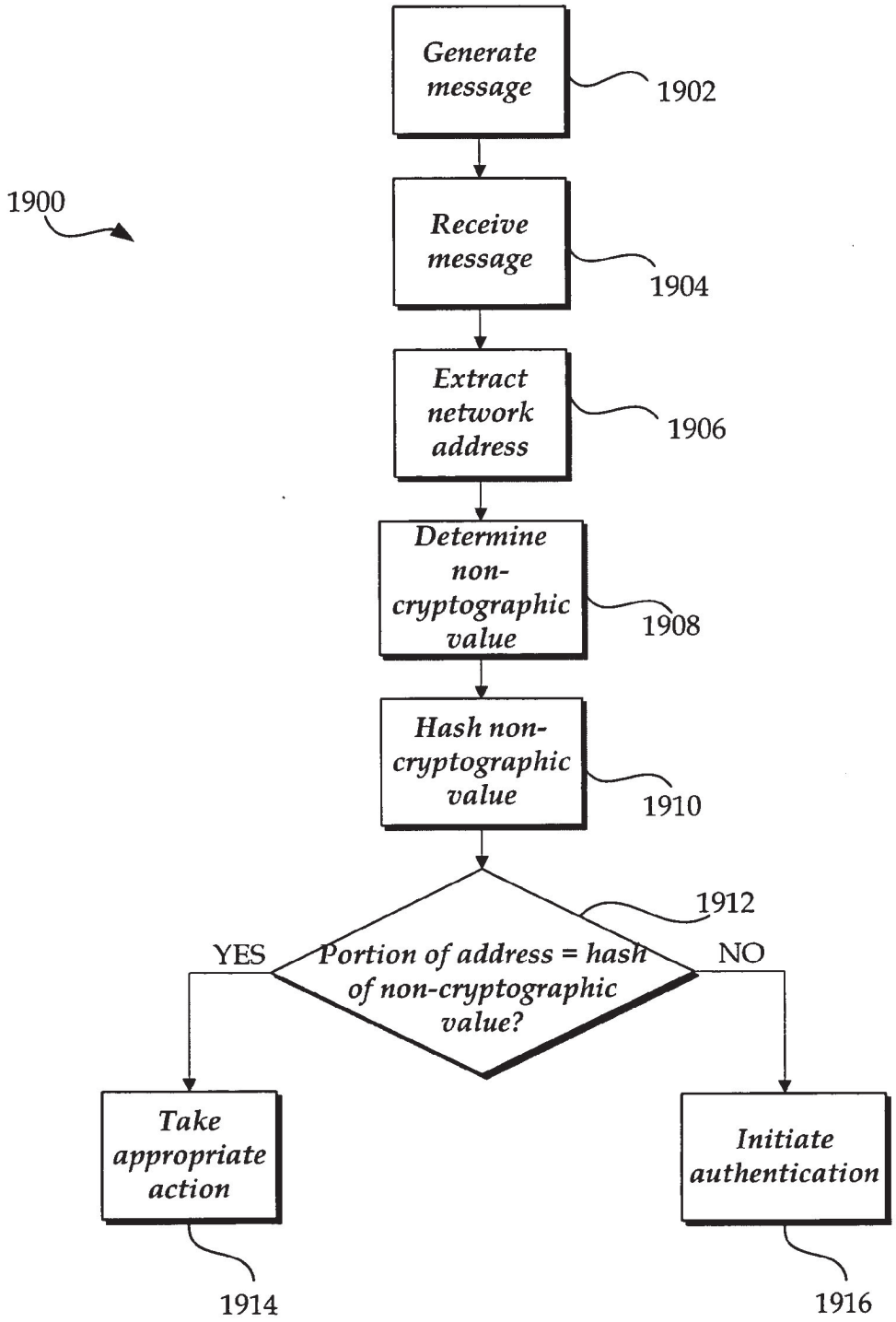

1800
Set non-selectable bits
1802
Determine non-cryptographic value
1804
Hash non-cryptographic value
1806
Set portion of address to portion of hash value
1808
Set remaining node-selectable bits
1810

FIG. 19


1900
Generate message
1902
Receive message
1904
Extract network address
1906
Determine non-cryptographic value
1908
Hash non-cryptographic value
1910
1912
Portion of address = hash of non-cryptographic value?
YES
NO
Take appropriate action
1914
Initiate authentication
1916

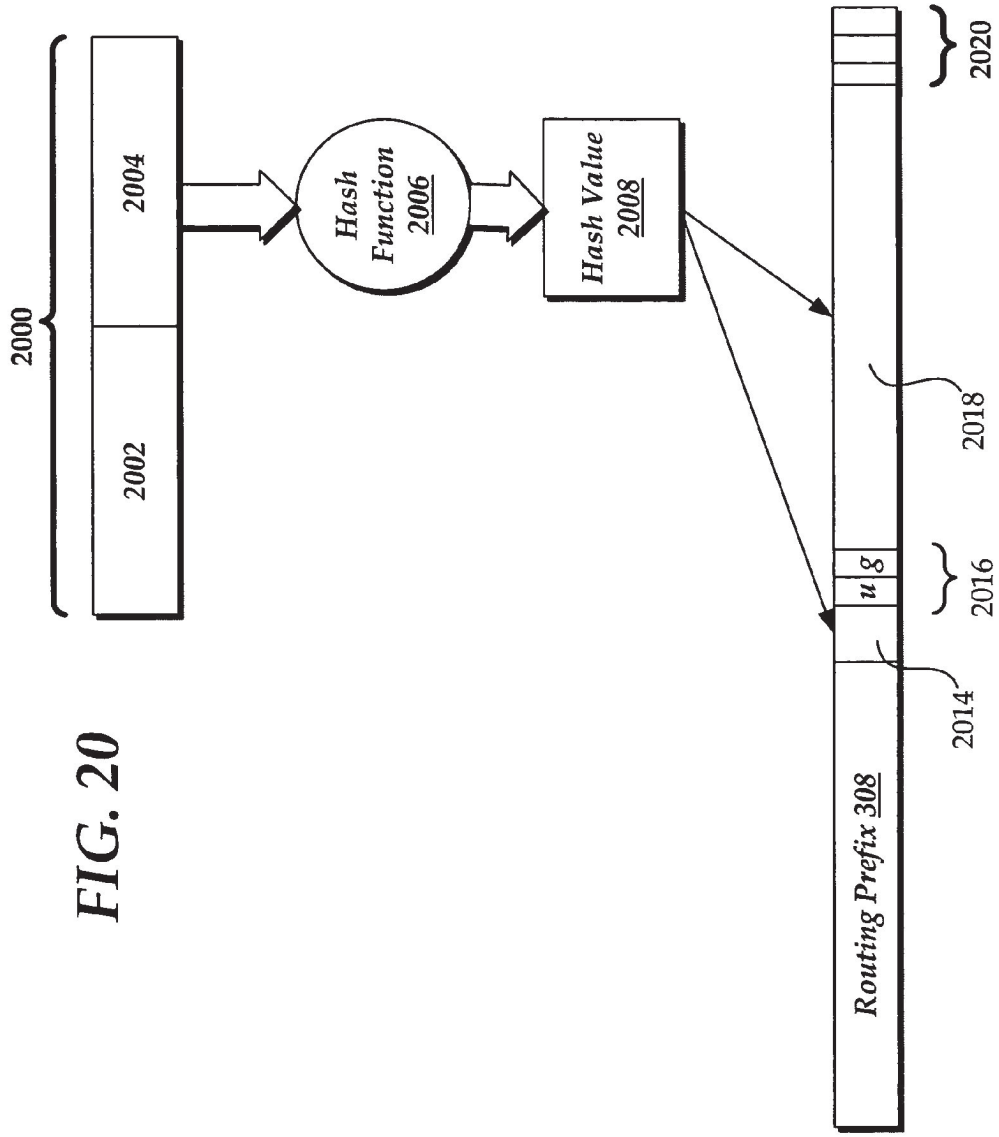

2000
2004
2002
Hash Function 2006
Hash Value 2008
2020
2018
2016
u
g
2014
Routing Prefix 308

FIG. 20

FIG. 21

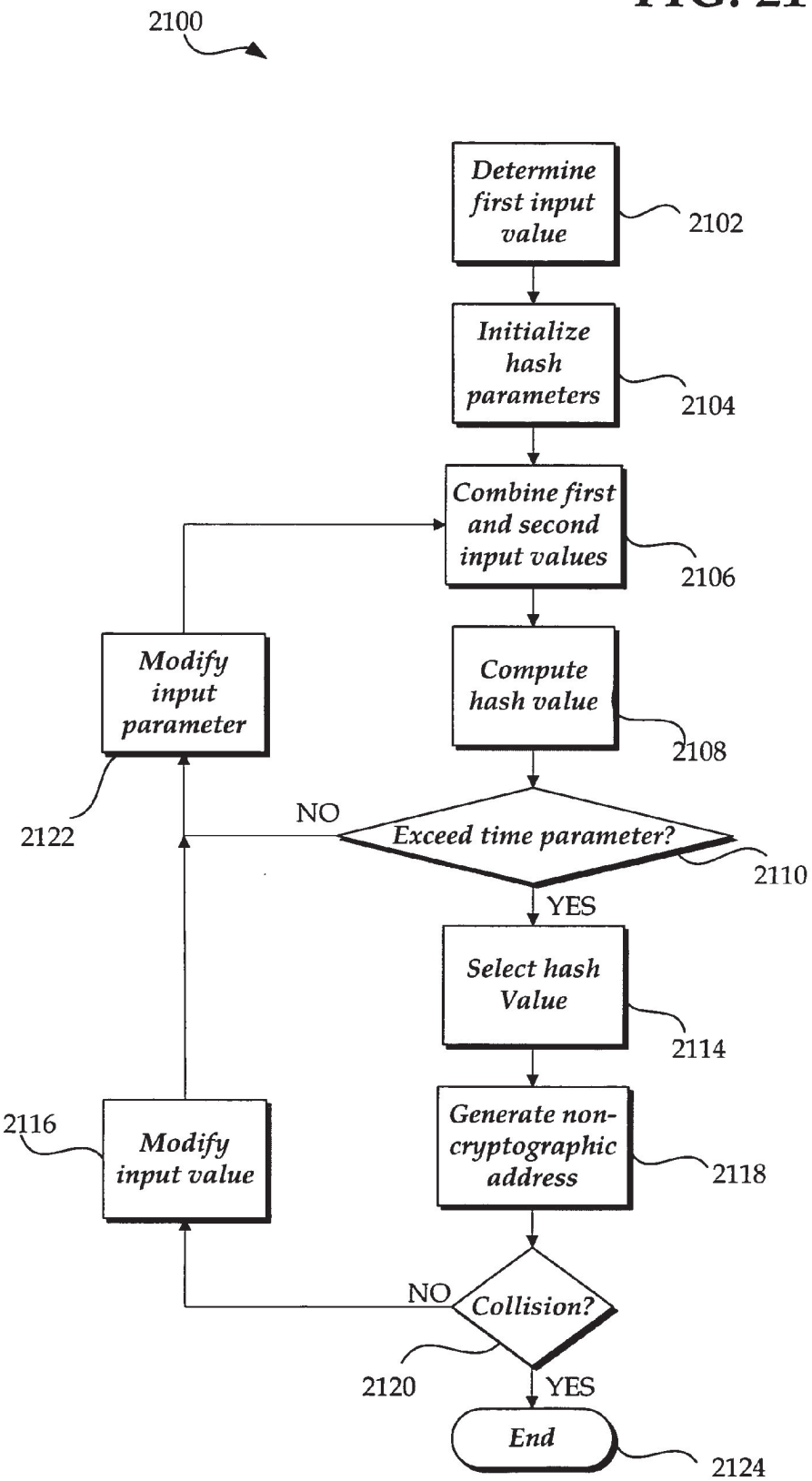

2100
Determine first input value
2102
Initialize hash parameters
2104
Combine first and second input values
2106
Compute hash value
2108
Exceed time parameter?
2110
NO
YES
Modify input parameter
2122
Select hash Value
2114
Generate non-cryptographic address
2118
Collision?
2120
NO
YES
Modify input value
2116
End
2124

FIG. 22

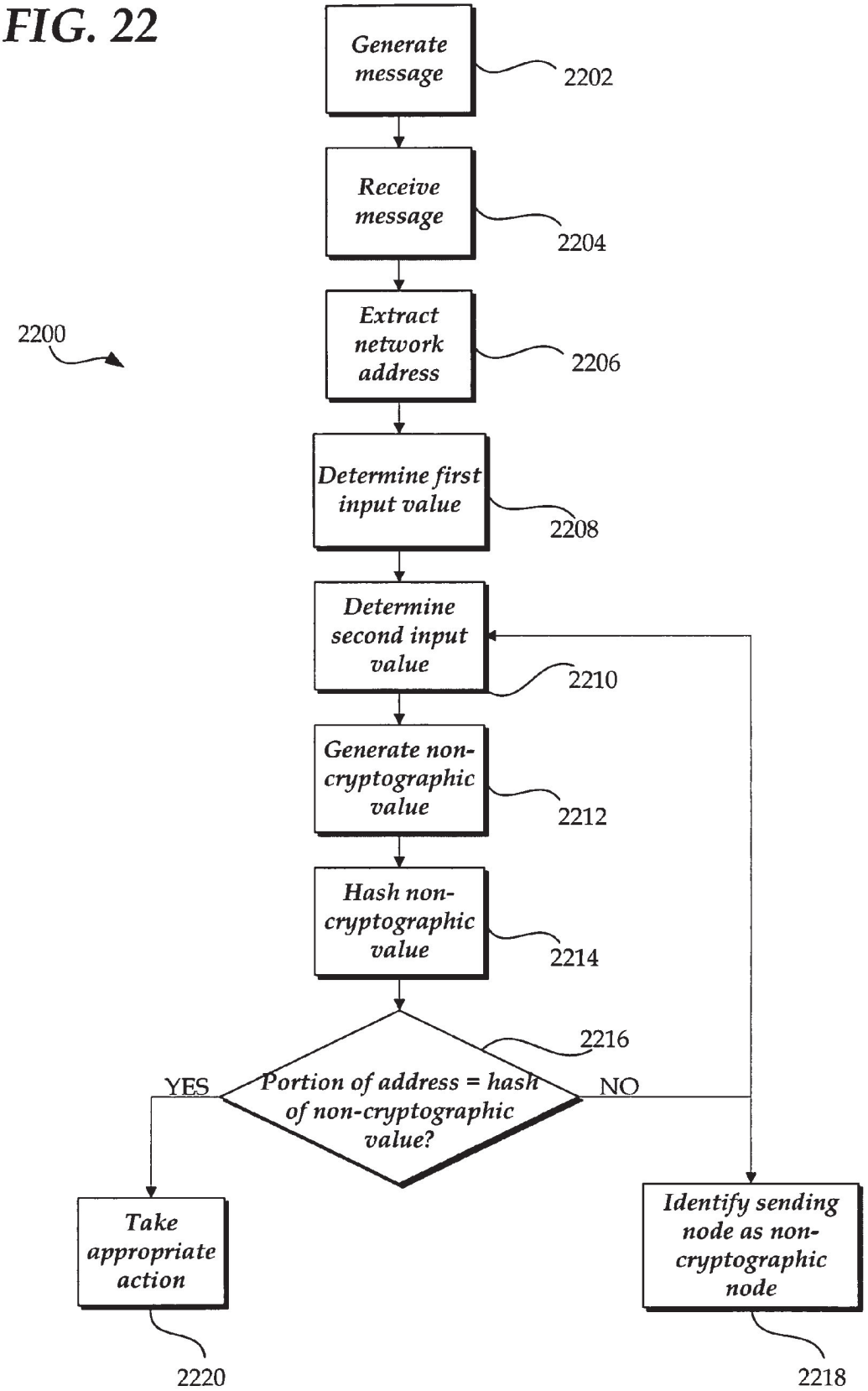

2200
Generate message
2202
Receive message
2204
Extract network address
2206
Determine first input value
2208
Determine second input value
2210
Generate non-cryptographic value
2212
Hash non-cryptographic value
2214
2216
Portion of address = hash of non-cryptographic value?
YES
NO
Take appropriate action
2220
Identify sending node as non-cryptographic node
2218

FIG. 23

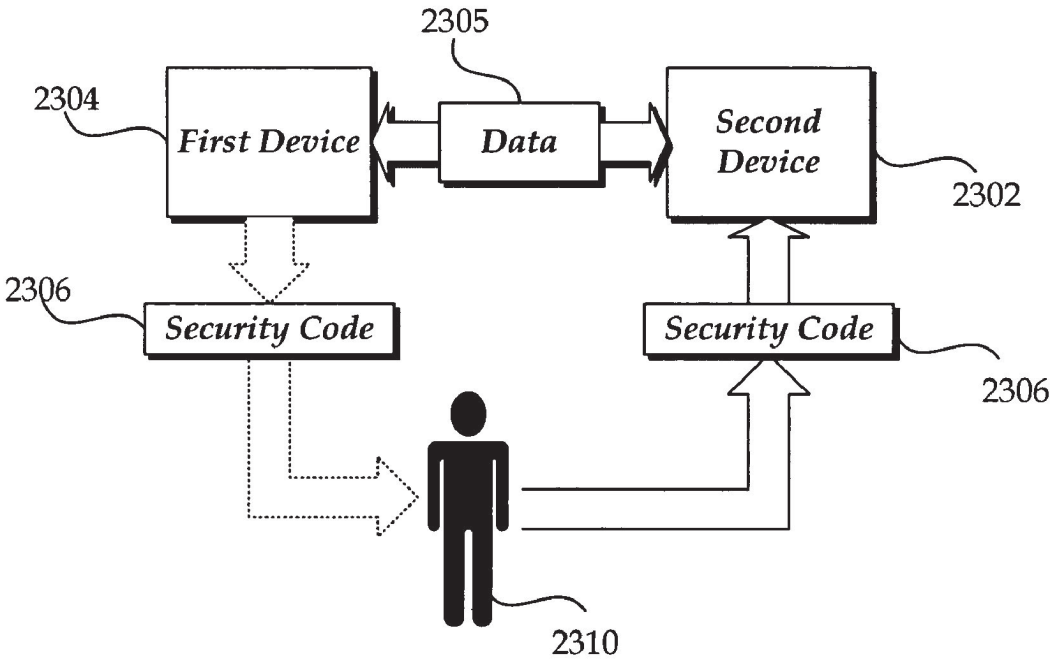

2305
2304
First Device
Data
Second Device
2302
2306
Security Code
Security Code
2306
2310

FIG. 24

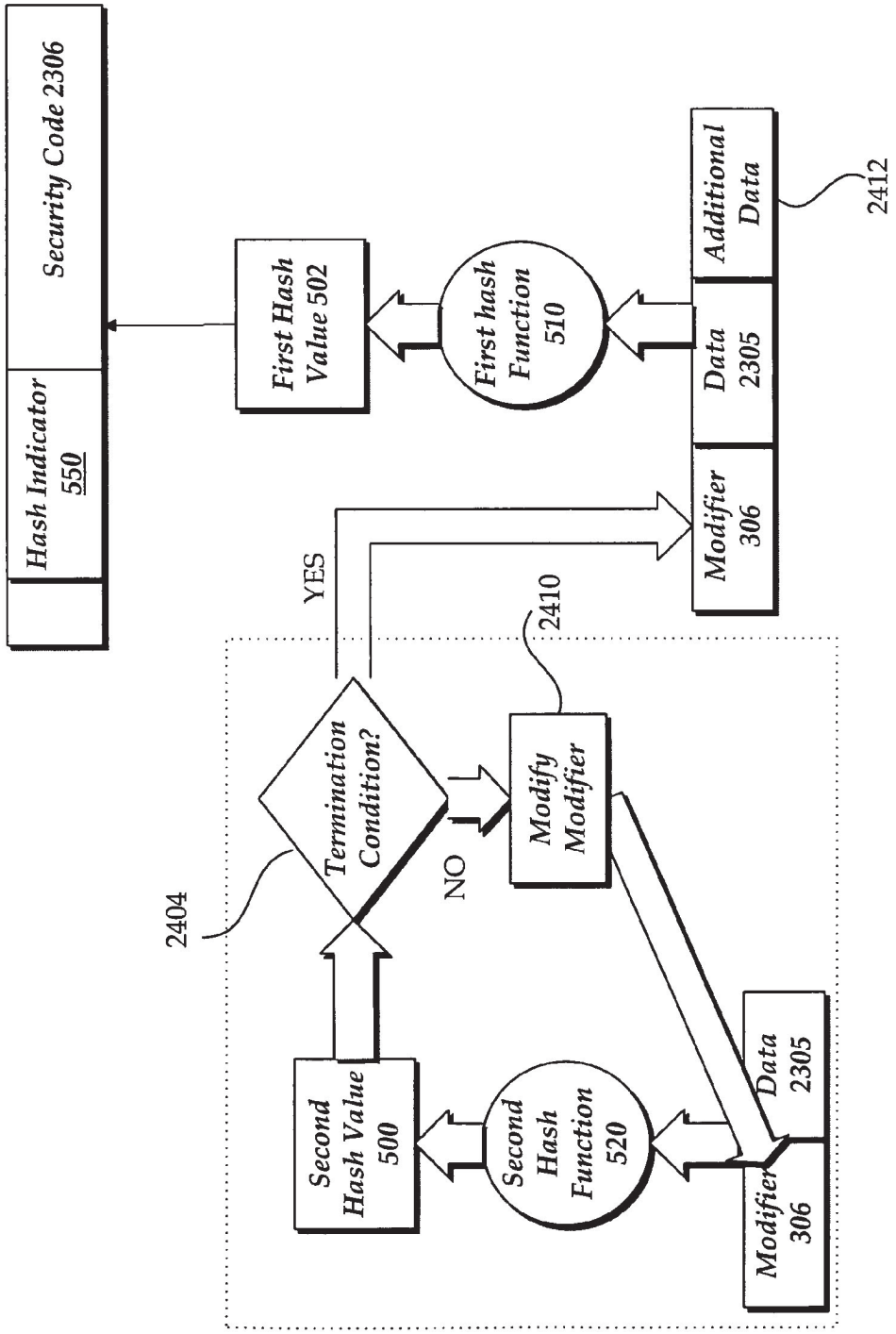

Security Code 2306
Hash Indicator 550
First Hash Value 502
First hash Function 510
Data 2305
Additional Data
2412
Modifier 306
YES
2410
Termination Condition?
NO
Modify Modifier
2404
Second Hash Value 500
Second Hash Function 520
Data 2305
Modifier 306

FIG. 25

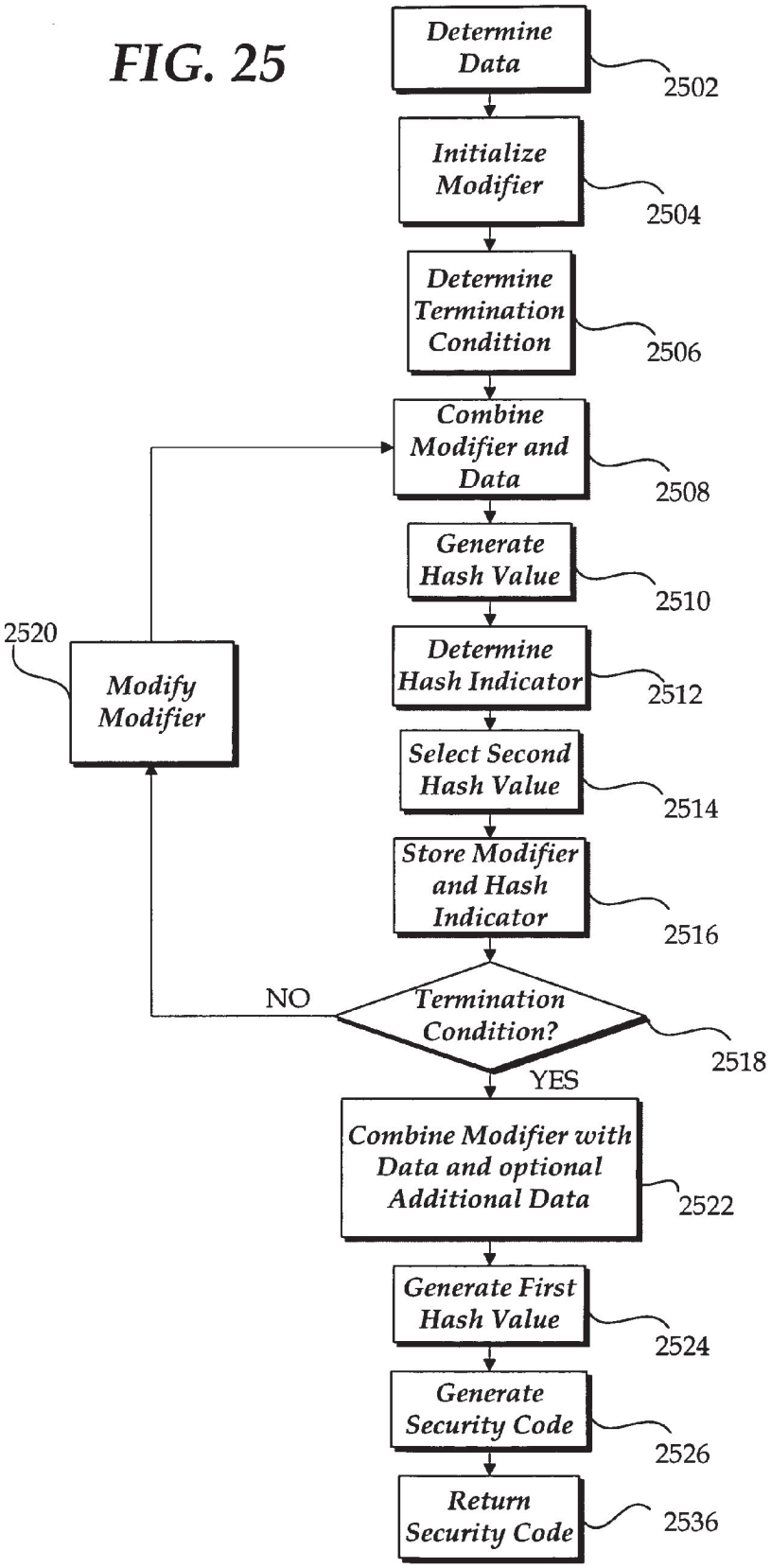
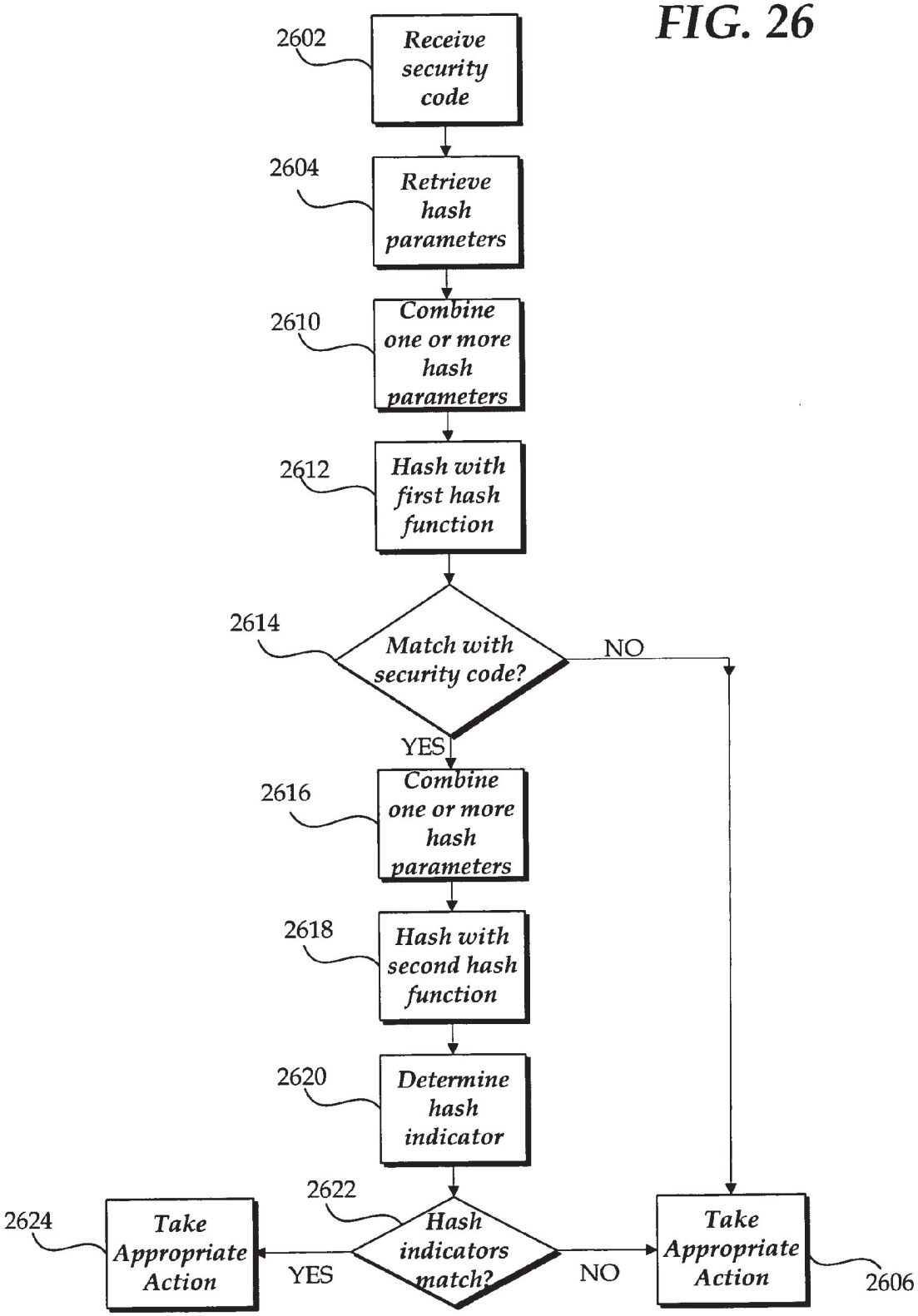

Determine Data
2502
Initialize Modifier
2504
Determine Termination Condition
2506
Combine Modifier and Data
2508
Generate Hash Value
2510
Determine Hash Indicator
2512
Select Second Hash Value
2514
Store Modifier and Hash Indicator
2516
Termination Condition?
2518
NO
YES
2520
Modify Modifier
Combine Modifier with Data and optional Additional Data
2522
Generate First Hash Value
2524
Generate Security Code
2526
Return Security Code
2536

FIG. 26


2602
Receive security code
2604
Retrieve hash parameters
2610
Combine one or more hash parameters
2612
Hash with first hash function
2614
Match with security code?
NO
YES
2616
Combine one or more hash parameters
2618
Hash with second hash function
2620
Determine hash indicator
2622
Hash indicators match?
YES
NO
2624
Take Appropriate Action
Take Appropriate Action
2606

HUMAN INPUT SECURITY CODES

CROSS-REFERENCE TO RELATED APPLICATION(S)

[0001] This application is a continuation in part of and claims priority to U.S. patent application Ser. No. 10/401,241, filed Mar. 27, 2003, which is incorporated herein by reference.

BACKGROUND

[0002] Authentication is an important issue in many types of network communications. In some security protocols, a human user may act as a secure channel. That is, the human user is required by these protocols to enter a secret or authentic value into a first device which will authenticate the user or a second device to the first device. For example, an authentication mechanism for UPnP2 devices may require the human user to type into a computing device (the first device) a registration code that is either shipped on paper with the device or readable from a display within the device. In this manner, the user is authenticated to use the first device. For another example, an authenticated key-exchange protocol may require a human user to read an authentication code to another human user over a telephone to prevent man-in-the-middle attacks against the protocol. In yet another example, users may be required to verify the hash codes of trusted public keys or root certificates (e.g., self-signed SSL certificates in a network browser application). In yet another example, a user may enter a security code (such as a hash of a public key) of a media device (first device) (e.g., television set, digital video recorder, stereo sound system, and the like) to a newly purchased universal remote control (the second device). The remote control device may then transfer the user input security code to the media device to allow the remote control to be authenticated and used with the media device.

[0003] The foregoing examples of the related art and limitations related therewith are intended to be illustrative and not exclusive. Other limitations of the related art will become apparent to those of skill in the art upon a reading of the specification and a study of the drawings.

SUMMARY

[0004] The following presents a simplified summary of the disclosure in order to provide a basic understanding to the reader. This summary is not an extensive overview of the disclosure and it does not identify key/critical elements of the invention or delineate the scope of the invention. Its sole purpose is to present some concepts disclosed herein in a simplified form as a prelude to the more detailed description that is presented later.

[0005] Using a human user as a secure channel may have limitations on the length of the security code. For example, human users have a limited capability to remember long security codes, and if the security code is long enough, a human user may have difficulty entering the security code correctly. In addition, human users, even if able to remember and/or copy a long security code, may find the task unpleasant. Typically, a human user may be willing to input a value with 20-40 bits of entropy, while many security protocols require the transfer of a much longer value (e.g., 50-80 bits) to be secure against brute-force attacks. This problem may exacerbate over time, as the human capacity to enter security codes remains the same while the power and speed of computing devices continue to increase.

[0006] For example, a public key of a remote-control device may be transferred to a television to allow the remote to be used with the television. In this example, the television public key is 100 digits long, but any length of key may be used. Typically, a media device may recognize an authorized remote control with a key exchange between the two devices (e.g., the remote control and the media device). The key exchange between the devices may be implemented through a certificate or Kerberos. However, to require a user to transfer the public key of the television to the remote control may be burdensome and fraught with error. The public key may be hashed in some examples. However, with a 128 bit hash value of the public key, the hash value may still be approximately 20 digits long, which may still be considered burdensome to a user. On the other hand, a user input code having only 5-10 digits, may only encode approximately 30-60 bits of information, which may be considered insecure. Thus, to extend the security of user entered security codes, and/or to decrease the length of user entered security codes, an extensible hash may be used. Examples of extensible hashes are described in the parent application, U.S. application Ser. No. 10/401,241, filed Mar. 27, 2003, and the IETF Request for Comment No. 3972, Cryptographically Generated Addresses, March 2005, at http://www.ietf.org/rfc/rfc3972.txt, both of which are incorporated herein by reference.

[0007] A generic and extensible format for CGAs and an associated authentication mechanism may provide much of the security of heavyweight authentication mechanisms. In some cases, the extensible format of a CGA may lower administrative and communicative overhead. The extensible CGAs may effectively remove or reduce the 64-bit limit on the length of a cryptographic hash value. Although extensible CGAs may increase the cost of address generation, extensible CGAs may correspondingly increase the cost of "brute-force" attacks (i.e., exhaustive searches for hash collisions or creation of large pre-computed databases of interface identifiers from an attacker's own public key(s) used to find matches for many addresses). In many cases, extensible CGAs may maintain or slightly increase the cost of address use and verification as compared to prior CGA use and verification. For purposes of brevity, extensible CGAs will hereinafter be referred to as ECGAs.

[0008] In one example of generating an ECGA, an address owner computes two hash values using the public key and other parameters. A second hash value sets an input parameter for a first hash value, and at least a portion of the network address is derived from the first hash value. The combination of the two hash values increases the computational complexity of generating new addresses and, consequently, the cost of brute-force attacks. The combination of two hash values allows the address owner to select levels of security above the 64-bit limit of prior CGA-based authentication mechanisms.

[0009] As another measure against brute-force attacks, the routing prefix (i.e., the portion which is not selectable by the address owner) of the address may be included in the first hash input. Including the routing prefix in the first hash calculation increases the cost of pre-computation attacks by

making some brute-force attacks against global-scope addresses more expensive because the attacker must do a separate brute-force search for each address prefix. Should the address owner move between networks and thereby change the corresponding routing prefix, the second hash value can be reused because it may be independent of the value of the routing prefix. In this manner, a computationally expensive part of address generation may be avoided and/or reduced. Because the high overhead of re-calculating the second hash function may not be necessary when the routing prefix changes, ECGAs may be suitable for mobile communications applications.

[0010] The input to either or both of the first and second hash functions may be formatted as defined in IETF Request For Comments 3972 or as another data structure that contains the public key and possibly other parameters. In another example, the input to either or both of the first and second hash functions may be formatted as part of a self-signed certificate. A self-signed certificate is a standard format for storing and transferring public keys in Internet protocols. The signature on the certificate proves that the public-key owner wants to use the ECGA. Additionally, certificates make it easier to use ECGA-based and Public Key Infrastructure (PKI)-based address authentication side by side in the same protocols. Some protocols, however, may need to save octets and transfer only the public key and other parameters, rather than a full self-signed certificate. An optimized parameter format, for example, one similar to the format in RFC 3972, may be defined for this purpose. Moreover, protocols may optimize the transfer of the parameters by avoiding retransmission of parameters already known the receiver or ones sent in other parts of the protocol.

[0011] The address owner may use the ECGA as its own address when sending a message or, if sending the digitally signed message from another address, may indicate that the message is being sent on behalf of the owner of the ECGA. The recipient of the message with the public signature key and other CGA parameters may authenticate the source of the message by performing two hash functions on the public key and other parameters and by verifying the signature on the message.

[0012] In an ECGA-based authentication mechanism, the recipient of the message may be either an separate entity or the same entity that created the message. An example of the latter is the message creator storing messages for retrieval at a later time. When the creator of the message retrieves the message, the authentication mechanism assures that the integrity of the message has not been compromised while it has been stored.

[0013] In the various examples discussed below, one or more of the various problems described above have been reduced or eliminated, while other examples are directed to other improvements. Many of the features described above and/or other features will be more readily appreciated as the same becomes better understood by reference to the following detailed description considered in connection with the accompanying drawings.

DESCRIPTION OF THE DRAWINGS

[0014] The present description will be better understood from the following detailed description read in light of the accompanying drawings, wherein:

[0015] **FIG. 1** is a schematic diagram showing an example communications network in which an authentication mechanism can be practiced;

[0016] **FIG. 2** is a schematic diagram showing an example computer device on which at least apportion of which the authentication mechanisms may be implemented;

[0017] **FIG. 3** is a schematic diagram showing example inputs and example outputs of example address generation procedure;

[0018] **FIG. 4** is a flowchart of an example procedure for creating an extensible cryptographically generated address by the sending device;

[0019] **FIG. 5** is a schematic diagram showing an example derivation of an extensible cryptographically generated address from a public key of the sending device and other parameters;

[0020] **FIG. 6** is a schematic diagram showing example components of an authenticated message sent from the sending device to a receiving device;

[0021] **FIG. 7** is a schematic diagram showing example inputs and example outputs of an example address ownership verification procedure;

[0022] **FIG. 8** is a flowchart of an example procedure for verifying that a public key belongs to the owner of an extensible cryptographically generated address;

[0023] **FIG. 9** is a schematic diagram showing an example verification of a binding between a public key and an extensible cryptographically generated address by a receiving device;

[0024] **FIG. 10** is a sequence diagram showing an example life-cycle of the authentication process;

[0025] **FIG. 11** is a flowchart of an example procedure for creating an extensible cryptographically generated address by the sending device;

[0026] **FIG. 12** is a flowchart of an example procedure for creating an extensible cryptographically generated address by the sending device;

[0027] **FIG. 13** illustrates a flowchart of an example procedure for generating an extensible cryptographically generated address by the sending device;

[0028] **FIG. 14** is a schematic diagram showing an example non-cryptographic address;

[0029] **FIG. 15** is a flowchart of an example procedure for generating a non-cryptographic address of **FIG. 14**;

[0030] **FIG. 16** is a flowchart of an example procedure for authenticating a non-cryptographic address of **FIG. 14**;

[0031] **FIG. 17** is a schematic diagram showing another example non-cryptographic address;

[0032] **FIG. 18** is a flowchart of an example procedure for generating a non-cryptographic address of **FIG. 17**;

[0033] **FIG. 19** is a flowchart of an example procedure for authenticating a non-cryptographic address of **FIG. 17**;

[0034] **FIG. 20** is a schematic diagram showing an example derivation of an non-cryptographic address from a first and second input value;

[0035] **FIG. 21** is a flowchart of an example procedure for generating a non-cryptographic address of **FIG. 20**;

[0036] **FIG. 22** is a flowchart of an example procedure for authenticating a non-cryptographic address of **FIG. 20**;

[0037] **FIG. 23** illustrates a schematic diagram of an example system using a human as a secure channel to communicate a security code;

[0038] **FIG. 24** illustrates a schematic diagram of en example generation of a security code;

[0039] **FIG. 25** illustrates a flowchart of an example method of generating a security code; and

[0040] **FIG. 26** illustrates a flowchart of an example method of authenticating a security code.

[0041] Like reference numerals are used to designate like parts in the accompanying drawings.

DETAILED DESCRIPTION

[0042] 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 example may be constructed or utilized. The description sets forth the functions of the example and the sequence of steps for constructing and operating the example. However, the same or equivalent functions and sequences may be accomplished by different examples.

[0043] Although the present examples are described and illustrated herein as being implemented in an authentication mechanism 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 authentication, authorization, and access control systems.

[0044] The following description will introduce generation of an extensible CGA (hereinafter ECGA) and how the ECGA may be authenticated by an authentication mechanism. Implementation options will then be discussed and how the authentication mechanism may operate in other cases. Features of ECGAs may be used to provide a simple, lightweight, and/or deployable authentication mechanism, such as to authenticate a message or other information. For example, the authentication mechanism may determine whether the information can be trusted to be from the sender address or other entity it purports to be from. All or a portion of the authentication mechanism may be applied to any type of message or information to enhance security and/or authentication.

[0045] An authentication mechanism may be deployable within a network environment, such as the example network of **FIG. 1**. As shown in **FIG. 1**, a computing device **102** on a local area network (LAN) **100** may be accompanied by a server **104** and/or a router **106**. The router allows the devices on the LAN to communicate over an internetwork **108** to remote computing devices such as device **110**. The Internet is one example of an internetwork. Any of the devices, including the router, in **FIG. 1** may send a message to any other device. In protocols typically used today, the message contains the network address of the sender (the "FROM address") and the network address of the recipient ("the TO address").

[0046] Consider, as a first example, that device **102** is operated by a systems administrator authorized to reconfigure the router **106**. For security's sake, the router does not accept commands from devices other than device **102**. But when the router receives a reconfiguration command message with a FROM address set to the address of device **102**, the router believes the message is authentic and carries out the command. (This example is for illustrative purposes only: Router communications are generally much more secure than is depicted here.) However, it is very easy to create and send a message with a deceptive FROM address. For example, device **112** could create a router reconfiguration command message and put device **102**'s network address in the FROM address field. Without a mechanism for authenticating a message, the router cannot know which device actually sent the message. The router does not know that the message is not authentic and carries out the command, possibly compromising the security of all of the devices on the LAN. As a second example, device **102** uses a "Neighbor Discovery" protocol to resolve the network address of a router to a link-layer address (i.e., MAC address) and then uses the router when communicating with remote devices such as device **110**. Without an authentication mechanism in place, device **114** may impersonate router **106**. Messages from device **102** intended for device **110** would then go to device **114** which may read the contents of the messages, discard the messages (thus launching a denial-of-service attack against device **102**), and/or alter the messages before sending them on to device **110**. As a third example, device **114** may interfere with the process in which device **102** obtains a network address. For example, if device **102** executes a "Duplicate Address Detection" protocol to check that no other network node is using the same address as device **102**, device **114** may generate messages that make it appear as if all addresses were already in use (i.e., owned) by some other network node. This way, device **114** can prevent device **102** from obtaining an address on the network. These are just three examples illustrating the ubiquitous need for authentication. The examples are extendable to any sort of message sent to any sort of device using any sort of communications method or protocol.

[0047] To fend off attacks such as those described in the first two examples of the previous paragraph, a recipient of a message may authenticate the sender of the message, that is, determine if the message was actually sent by the sender indicated in the message. When the recipient is able to reliably determine the true sender of a message, then the recipient can choose to act on the message only if it is sent by an appropriate or trusted device. To fend off attacks of the sort described in the third example of the previous paragraph, a network node needs a method of generating a new network address and for proving that it has, with high probability, generated the address itself. Such a method prevents malicious network nodes from falsely claiming ownership of the new address with a significant success rate.

[0048] In the description that follows, the authentication mechanism is described with reference to acts and symbolic representations of operations that are performed by one or more computers, unless indicated otherwise. As such, it will be understood that such acts and operations, which are at times referred to as being computer-executed, include the manipulation by the processing unit of a computing device of electrical signals representing data in a structured form. This manipulation transforms the data or maintains the data

at locations in the memory system of the computing device, which reconfigures or otherwise alters the operation of the computer in a manner well understood by those skilled in the art. The data structures where data are maintained are physical locations of the memory that have particular properties defined by the format of the data. However, while the authentication mechanism is being described in the foregoing context, it is not meant to be limiting as those of skill in the art will appreciate that various of the acts and operations described hereinafter may also be implemented in hardware.

[0049] Referring to **FIG. 2**, the authentication mechanism may be implemented in communications between network nodes on connected computer networks. Each of the network nodes resides in a computing device that may have one of many different computer architectures. For descriptive purposes, **FIG. 2** shows a schematic diagram of an example computing device architecture usable for these devices. The architecture portrayed is only one example of a suitable environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing devices be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in **FIG. 2**. The authentication mechanism is operational with numerous other general-purpose or special-purpose computing or communications environments or configurations. Examples of well known computing systems, environments, and configurations suitable for use with the invention include, but are not limited to, mobile telephones, pocket computers, personal computers, laptop devices, handheld devices, set top boxes, servers, routers, network address translators, switches, wireless access points, multiprocessor systems, microprocessor-based systems, minicomputers, mainframe computers, and distributed computing environments that include any of the above systems or devices.

[0050] In a basic configuration as shown in **FIG. 2**, a computing device **102** typically includes at least one processing unit **200** and memory **202**. The memory **202** may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.), or some combination of the two. The basic configuration is illustrated in **FIG. 2** by the dashed line **204**.

[0051] The storage media devices may have additional features and/or functionality. For example, the storage media devices may include additional storage (removable and non-removable) including, but not limited to, PCMCIA cards, magnetic and optical disks, magnetic tape, external hard drives, and any other suitable removable or non-removable storage. Such additional storage is illustrated in **FIG. 2** by removable storage **206** and non-removable storage **208**.

[0052] Computer-storage media include volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. Memory **202**, removable storage **206**, and non-removable storage **208** are all examples of computer-storage media. Computer-storage media include, but are not limited to, HDD, RAM, ROM, EEPROM, flash memory, other memory technology, CD-ROM, zip drives, digital versatile disks, other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage, other magnetic storage devices, and any other media that can be used to store the desired information and that can be accessed by the computing device.

[0053] Computing device **102** may also contain communication channel **210** that allow the host to communicate with other devices. Communication channel **210** may include one or more communication channels and 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 include 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 includes wired media, such as wired networks and direct-wired connections, and wireless media such as acoustic, radio, infrared, and other wireless media. The term computer-readable media as used herein include both storage media and communications media.

[0054] The computing device **102** may also have input components **212** such as a keyboard, mouse, pen, a voice-input component, a touch-input device, etc. Output components **214** include screen displays, speakers, printer, etc., and rendering modules (often called "adapters") for driving them. The computing device **102** has a power supply **216**.

[0055] Extensible Cryptographically Generated Addresses

[0056] A device, such as the computing device **102**, may write a message in a way that the message could only have been written by this particular device, e.g., the message may be authenticated as coming from the sending computing device such as by using an authentication mechanism. The authentication mechanism may be unilateral in that the recipient does not need to have further communications with the sender for completing the authentication process. For example, the message may contain everything the message recipient, such as the router **106** of **FIG. 1**, needs to decode the message and to determine that it must have come from a particular device such as the computing device **102**.

[0057] The authentication mechanism may be based on public key cryptography used in combination with the selection of a network address of the message sender. The selected network address may include an extensible Cryptographically Generated Address (ECGA) which is based on the public key and other parameters. In the example of **FIG. 1**, the ECGA address is an address of the computing device **102**. One way this address may be derived from the public key of the device **102** is described in connection with **FIGS. 3, 4**, and **5**.

[0058] The ECGA may be generated in any suitable manner, such as by an ECGA generator **312** shown in **FIG. 3**. An ECGA generator **312**, which may be implemented in the computing device **102** of **FIG. 1**, may take one or more inputs (or any combination thereof) which may include one or more of the public key **300** of the address owner, the routing prefix **308** of the network of the address owner, a collision count **310** data value (which may be initially set to zero), a modifier **306** data value, a security parameter (Sec) **302**, a time parameter, and the like. The modifier **306** may be any suitable modifier, which in one example may be a

random 16-octet string. The security parameter **302** may be any suitable security input, which may be an unsigned 3-bit integer having a value between 0 and 7. The value of the security parameter **302** may be selected by the sending device (address owner), such as by computing device **102**. The time parameter may define the time to compute a second hash value in generating the ECGA. The ECGA generator **312** outputs a new ECGA address **314**. The output ECGA address may be in accordance with any appropriate protocol, including without limitation Internet Protocol version 6 (IPv6), a link-layer address (MAC address), and the like. The ECGA generator may output associated CGA parameters **316**, and an optional self-signed certificate.

[**0059**] In one example, the output ECGA may be in accordance with the IPv6 protocol. As shown in **FIG. 5**, an IPv6 address **314** is 128 bits long. It is generally divided into two parts. The leftmost 64 bits are the routing prefix **308**, which are used for routing Internet Protocol (IP) packets across the Internet to the destination network. The rightmost 64 bits are the interface identifier **504**, which identifies an individual node within a local network. The interface identifier **504** part of the address is referred to as the "node-selectable" portion because the sending device **102** is free to set this part as it sees fit. The interface identifiers may be chosen in any suitable way, e.g., randomly, as long as no two nodes on the same network share the same value, assigned by a local administrator, and the like. While the division of the 128-bit IPv6 address **314** into two 64-bit parts is typical, the division can be done in some other proportion. For example, the length of the interface identifier **504** may be increased at the expense of decreasing the length of the routing prefix **308**, or some parts of the interface identifier may be determined by a routing protocol making them effectively a part of the address prefix.

[**0060**] The network address of the sending node may be identified as an ECGA in any suitable manner, including without limitation one or more of a preset portion of the routing prefix, a preset portion of the interface identifier, a header or message content in the message, and the like. For example, the network address itself may identify the address as an ECGA. In one example, the interface identifier **504** may include two bits which may be set to indicate that the network address is an ECGA. In IPv6 addresses, the 7th bit from the left in the 64 bit interface identifier **504** is the Universal/Local bit or "u" bit. It is usually set to 1 to mean that the interface identifier **504** is configured from an EUI-**64** identifier from the interface hardware and, thus is globally unique. The 8th bit from the left is the Individual/Group or "g" bit, which is usually set to 1 for multicast addresses. The bit combination u=1, g=1 is currently unused because a multicast address (as indicated by g=1) cannot be globally unique (as indicated by u=1). In this manner, the bit combination of u=1, g=1 may be allocated for indicating ECGA addresses. Allocating one or more bits of the network address, such and the u and/or g bits of an IPv6 address may facilitate mixed use of cryptographically generated and non-cryptographically generated addresses in any security protocol without weakening the security of either address type. However, it is to be appreciated that ECGA addresses and authentication may be used even if such allocation is not made.

[**0061**] The routing prefix or the interface identifier portion of the ECGA may include a hash indicator indicating the type or length (or both) of the hash value used in generating the ECCA. In one example, the hash indicator includes the security parameter (Sec) **302**, which may indicate the level of security. If a time parameter is used to determine the security level when generating the address, the determined security level may be indicated by the security parameter or hash indicator exactly in the same way as if no time parameter were used. That is, the mechanism for determining the security level may be independent of the mechanism for communicating the security level in the address. However, it is to be appreciated that any suitable hash indicator may be used to indicate the selected second hash value or a set of possible hash values.

[**0062**] If the hash indicator value is encoded into the address bits, an attacker may not be able to change the value of the security parameters or other inputs to the first and/or second hash values without also changing the address. In one example, the hash indicator is an unsigned integer. Although any suitable bit length may be selected for the hash indicator, **FIG. 5** illustrates one example length of the security parameter as 3 bits. The hash indicator **550** may be in any suitable portion of the interface identifier **504**. For example, as shown in **FIG. 5**, the hash indicator may be encoded in the leftmost bit position(s) of the interface identifier or node selectable portion of the ECGA. The hash indicator may allow the address owner to increase the computational cost of address generation and, thus to also correspondingly increase the cost of brute-force attacks against the address. Thus, in some cases, as described further below, incrementing Sec **302** by one may add 16 bits to the length of the hash that the attacker must break.

[**0063**] To generate an ECGA, the network address, such as an IPv6 address, may include a hash value based on public key information. However, the node selectable portion of the network address in IPv6 is typically 64 bits. In some special protocols, the node selectable portion may be longer at the at the expense of decreasing the length of the routing prefix. In other protocols, such as NAT and firewall traversal protocols, the node selectable portion may be shorter, such as 48 bits, because the routing prefix is longer or some bits of the interface identifier are determined by the protocol. In this manner, the length of a hash output is limited to approximately 64 bits and sometimes to as little as 48 bits in IPv6. However, in some cases, approximately 48 or 64 bits may not provide sufficient security and/or may not contain a sufficiently strong cryptographic hash value.

[**0064**] The ECGA includes a hash of the public key as a portion of the network address. As shown in **FIG. 5**, at least a portion of the interface identifier **504** may include at least a portion of a hash of the public key **300**. The hash of the public key may be generated by any suitable first hash function **510**, including without limitation MD5, SHA-1, N-hash, and the like.

[**0065**] With reference to the flowchart of **FIG. 11**, generating an ECGA may begin with determining **1100** the address owner's public key. The address owner's public key may be determined in any suitable manner. For example, the public key may be determined by the ECGA generator such as by retrieving the public key from memory of the address owner's computing device, retrieving the public key from a certification authority, and the like. Determining the public key may include transforming or encoding the public key

into a format usable by the ECGA generator. For example, as shown in operation **400** of **FIG. 4**, the public key of the address owner may be DER-encoded as an ASN.1 structure, which may have a type such as SubjectPublicKeyInfo.

[**0066**] The ECGA parameters may be determined **1102**. For example, as shown in operation **402** of the example flowchart of **FIG. 4**, the modifier data value (e.g., modifier **306** of **FIG. 3**) may be initialized as **16** random octets or any other appropriate modifier value. The routing prefix (e.g., routing prefix **308** of **FIG. 3**) may be set to the routing prefix value advertised by the router at the address owner's local subnet. The collision count (e.g., collision count **310** of **FIG. 3**) may be initialized to zero. The public key may be set to the DER-encoded public key of the address owner. The determination of the ECGA parameters may include transforming or encoding the ECGA parameter values into a format usable by the ECGA generator. For example as shown in operation **404** of **FIG. 4**, each ECGA parameter may be concatenated to form a CGA Parameters data structure as defined in IETF RFC 3972, which is incorporated herein by reference. In another example, the CGA parameters may be DER encoded as an ASN.1 structure.

[**0067**] The second hash value may be computed **1104**. Although called the "second" hash value herein, it is actually the first hash value calculated in time. The second hash value may be a hash of a combination of the public key **300** and the one or more of the CGA parameters determined in operation **1102**. For example, the second hash value may be a hash of a combination of the modifier **306** and the public key **300** of **FIG. 3**. For example, the modifier **306** and public key **300** may be concatenated **404** with zero-valued routing prefix and collision count to form a CGA Parameters data structure as defined in IETF RFC 3972 as shown in **FIG. 5** and the method of **FIG. 4**. The concatenation may be hashed by a second hash function **520** shown in **FIG. 5** to provide a second hash value **500** shown in **FIG. 5**. The second hash function may be any suitable hash function, such as the SHA-1 algorithm, and may be the same as or different from the first hash function **510**.

[**0068**] The authentication mechanism may use the SHA-1 hash function for the implementation of both the first and second hash functions. The output of the hash function may be longer than is required by the algorithm, such as the 160-bits output of SHA-1. In that case, the hash value may be truncated to the appropriate length, for example, by taking the required number of leftmost bits of the hash function output. The SHA-1 algorithm is a one-way hash function, meaning that it takes a message and converts it into a fixed-length string of digits, also called a message digest. The speed of the hash function may not be important with respect to the address generation process. The purpose of the second hash function is to set the value of at least one input parameter for the first hash function (such as the modifier), and it is the combination of the two hash values that increases the computational complexity of generating new addresses. Since the second hash function exponentially increases the cost of hashing, the small speed differences between SHA-1 and other hash functions may be irrelevant in the present invention. While SHA-1 is used here as an example of a hash function, any other cryptographic function with the second-preimage-resistance property may be used instead.

[**0069**] In one example, the result of the second hash function may be the second hash value **500** in **FIG. 5**. The bits of a portion of the second hash value may be compared **1106** to zero, as shown in **FIG. 11**. For example, as shown in **FIG. 4**, the N*Sec leftmost bits of the second hash value **500** may be compared with zero (**406** and **408** in **FIG. 4**). For example, a second hash value of 00001 ccd45bfe68e38b8d47bab2cd070781d765f has a count value of 19 left-most 0 bits. If N=16, this corresponds to Sec=1 and N*Sec=16. The modifier and the determined count value may be stored in any suitable manner, such as in a data store.

[**0070**] The value N may be any suitable value which may depend on the protocol for which the ECGA is used. As shown in step **406** of **FIG. 4**, the value of N may be 16, however, it is to be appreciated that any suitable value for N may be used such as 4, 8, 12, and the like. If the N*Sec leftmost bits are all zero, or if Sec=0, (step **408** and **1108**), the process continues with the step **412**, **1112** of **FIGS. 4 and 11** respectively. Otherwise, the value of the modifier **306** (or other CGA parameter that is an input to both the first and second hash values) is modified **1110** in any suitable manner, as shown in **FIG. 11**. For example, another random number may be chosen for the modifier. In another example, as shown in **FIG. 4**, the modifier may be incremented in step **410**. The process of computing the second hash value continues again with step **404**, **1104** of **FIGS. 4 and 11** respectively, of determining the second hash value using the newly modified modifier and the public key.

[**0071**] The initial value of the CGA parameters such as the modifier **306** in step **402**, **1102** and the method of modifying the modifier in step **410**, **1110** can be chosen arbitrarily. In order to avoid repeatedly trying the same value for modifier **306**, the initial value may be chosen randomly. The quality of the random number generator may not be important as long as the same values are not repeated frequently. One way to modify modifier **306** is to increment the content octets as if they were a 128-bit integer (with any byte order) **410**.

[**0072**] For values of the security parameter (Sec) **302** greater than zero, this process is not guaranteed to terminate after a certain number of iterations. The brute-force search in steps **404-410** takes, on the average, approximately 2^{16*Sec} iterations to complete (the median number of iterations is $0.69*2^{16*Sec}$).

[**0073**] The value of Sec may be difficult to determine. For example, the greater the value of Sec, the greater the processing cost may be to generate the ECGA. However, if Sec is static, increases in computational speed in the marketplace may reduce the security of the resulting ECGA over a period of time. For example, if the cost of computing power halves every 18 months, the number of zero bits (N*Sec) required in the second hash value should be incremented by 1 every 18 months to maintain the same level of security against brute force attacks. However, the prediction of computing power over the next 5-50 years may be difficult. Moreover, asking the user or administrator to increase the value of N*Sec may be impossible or impracticable if the ECGA is generated automatically at the time the operating system installation or bootstrapping. Moreover, a predetermined value of N*Sec may be too small, which may cause the security of the generated ECGA to be less than desired. Conversely, a predetermined value of N*Sec may be too large, which may cause unacceptable delays in address generation.

[0074] A future cost of brute force attacks may be implied by the cost of generating the ECGA, e.g., the computing power of the computing system generating the ECGA. For example, the input N*Sec (e.g., Sec **302** of **FIG. 3**) may be replaced by a time input parameter indicating a time to be used for the address generation. In this manner, the generation of the second hash value may be based on the speed of the computing device generating the second hash value. The time input parameter may be set or initialized (such as in step **1102**) by any suitable party, such as a device manufacturer or operating system vendor or by the user and may be set to the time that the user may be willing to spend on ECGA generation. For example, the time input may be set within a range of approximately 1-60 seconds if the user is waiting for the computations to approximately 1 CPU day or more if the computation is done slowly in the background. If the ECGA generation is done by a fast computer or dedicated machine other than the device for which the address in intended, the computation time may be set depending on the capacity of the machine used for the ECGA generation. More particularly, rather than having an input N*Sec **302** based on a number referring to a number of bits, the time parameter which is input to the ECGA generator **312** may refer to the time to compute the second hash value **500** of **FIG. 5**. The time parameter may be in any suitable form and/or units, such as an integer or real number value in seconds, CPU cycles, and the like. The ECGA generator may compute multiple second hash values during a time period defined by the time parameter input, and the output second hash value **500** may be selected as the 'most secure' hash value generated in that time. The most secure hash value may be determined in any suitable manner, such as by the generated hash value in the input time which has the greatest number of zero bits. If the number of zero bits is expressed as N*Sec, where Sec is a security parameter encoded in some bits of the address, the most secure hash value may be determined by selecting the hash value that has the greatest number of zero bits rounded down to the nearest integer multiple of N. The greatest number of zero bits may be in a predetermined portion of the hash value (such as a predetermined number of the left-most bits), may be the greatest number of zero bits within the complete computed hash value, may be the greatest number of sequential zero bits within the computed hash value, the greatest hash extension, and the like.

[0075] **FIG. 12** illustrates an alternative method of generating an ECGA based on time rather than a security parameter. With reference to **FIG. 12**, the public key may be determined **1100**, similar to the public key determination of **FIG. 11**. The determination **1202** of the CGA parameters may include initializing the time parameter to a time allowed to compute the second hash value. A hash value of the combination of the public key and one or more CGA parameters, such as the modifier, may be computed **1204**, such as by the ECGA generator. The method may check **1206** if the time specified by the time parameter has been exceeded. The ECGA generator may determine if the time parameter has been exceeded in any suitable manner such as by comparing the time parameter to a clock noting elapsed time, a number of computer cycles spent in computing the one or more hash values, and the like. Alternatively, the termination condition **1206** for the hash computation may be implemented by an external interrupt received from a hardware timer or by a signal received from another thread of software on the same machine. If the time parameter has not been exceeded, one or more CGA parameters used as input to the second hash value, such as the modifier, may be modified **1210**. For example, the modifier may be modified in any suitable manner as described above, such as selecting another random octet, incrementing a previous value of the modifier, and the like. The modified parameter(s) may be combined with the public key to compute **1204** the next iteration of a hash value.

[0076] If the time parameter has been exceeded in the hash value generation, the second hash value may be selected **1208** from the one or more generated hash values computed by step **1204**. In one example, one or more of the most secure generated hash values may be stored in a hash value data store. Each hash value may be stored associated with the CGA parameter(s) used to compute the hash value. The second hash value may be selected in any suitable manner, such as by selecting the 'most secure' hash value generated in the time period specified by the time parameter. For example, after each hash value is generated **1206**, the number of zero bits may be counted in the selected portion and compared to the number of count of zero bits of a previous computed hash value. The CGA parameters used to compute the hash value with the greatest number of zero bits in the selected portion may be stored. In this manner, the stored CGA parameters at the end of the time defined by the time parameter may be selected for input to the first hash function **1112**. The stored and selected values may be stored in any suitable manner, such as in a data store, and may be associated with one or more other values, which may be stored in a data matrix of the data store.

[0077] The flowchart in **FIG. 13** shows an example of how the process of **FIG. 12** may be implemented in the particular case of the ECGA format defined in RFC 3972. Compared to the flowchart of **FIG. 5** where the security parameter Sec value is an input to the ECGA generation process is, the Sec value in **FIG. 13** is an output of the computation, and a time parameter is received as input. The generation will end if the time parameter is exceeded **1308**. During the brute-force search loop (steps **404**, **1306**, **1308**, and **410**), the hash value that matches the highest found Sec value is stored **1306**. After the time parameter is exceeded and the loop terminates, the modifier value that produced the highest found Sec value will be used for the remainder of the address generation.

[0078] The time parameter limits the maximum time that the CGA generation may take. The address generator may further reduce the average time consumed by the address generation by stopping the search for new modifiers if it is unlikely that a stronger address than has already been found would be found in the remaining time. That is, if a second hash value corresponding to security parameter Sec (i.e., N*Sec zero bits in the second hash value) has been found, the generator may calculate the probability of finding a second hash value with the next higher security parameter Sec+1 in the remaining time within the limits of the time parameter. This probability may be compared with a predetermined threshold, such as 1%, 50% or 99%. If the probability is below the threshold, the generator may stop trying further modifier values and use the most secure hash value and modifier found so far (i.e., one with security parameter Sec). When N=8 or N=16, a CGA parameters value (such as a modifier value) that produces a reasonably secure Sec

value may be found early in the process and continuing the brute-force search loop until the time parameter is exceeded is very unlikely to find a CGA parameters value (such as a modifier value) that produces an even higher Sec value. Therefore, the early stopping has the potential of significantly reducing the average time spent on ECGA generation.

[0079] The early stopping condition may be evaluated **1307** as a part of the brute-force search loop in **FIG. 13**. In that case, the termination condition **1308** for the loop may be triggered not only when the time parameter is exceeded but also when the early stopping condition holds, as shown in **FIG. 13**. The differences between **FIG. 5** and **FIG. 13** illustrate the differences between the ECGA generation algorithm (such as defined in IETF RFC 3972) and the present example when the time parameter implementation is applied to the standard CGA addresses of RFC 3972.

[0080] Given a threshold probability p for finding more secure CGA parameter values, maximum time for address generation MaxTime, and the already elapsed time t, the number of modifier values tried so far n, and largest already found security parameter Sec, the early stopping condition may be computed in the logarithmic scale (i.e., in terms of bit counts). For example, the generator may terminate the brute-force search if t>MaxTime, or if t>MinTime and N*(Sec+1)>log2((Max−t)*n/t)−log2(ln(1/(1−p))). MinTime is a minimum time for which the computation should take. The appropriate minimum time depends on the amount of variation in the CPU power available to the computation and should be set to be large enough to prevent random variations in the computation speed to have an effect on the stopping condition. Alternatively, the rate at which hashes are computed (n/t) may be estimated by other means, such as by measuring the performance of the CPU(s) prior to the ECGA generation.

[0081] For a given probability threshold p, the termination condition may be approximated by computing the numeric value of the last term and by rounding the terms up or down to the nearest integer (i.e. to full bits). For example, if N=16 and p=99%, the above stopping condition can be approximated as: t>Max or (t>Min and 16*(Sec+1)>RoundDown(log2((Max−t)*n/t))+7).

[0082] The early stopping condition may increase the probability that the search for Sec+1 will continue only if there is a high probability of finding a match in the remaining time. The 16- or 8-bit increments in hash length (N=16 or N=8) are relatively large steps. Therefore, the generator often stops early because it has found a match for one level of security (Sec) and has little change of reaching the next level (Sec+1). In the majority of cases, this will cause the address generation to terminate much before the allocated time. On some machines, however, the address generation will take the full allocated time MaxTime.

[0083] Once the second hash value has been generated (as illustrated in either **FIG. 11** through a security parameter and **FIG. 12** through a time parameter), the extensible cryptographic address generation can be generated by determining **1112** the first hash value. Any suitable process may be used to determine the first hash value based on the second hash value.

[0084] For example, steps **412** through **424** of **FIG. 4** or **FIG. 13** may be used to determine the first hash value using the first hash function. For example, the public key may be concatenated with one or more other parameters (as shown in **FIG. 4** and **FIG. 5**), such as a composite of one or more data fields, including any one or more of CGA parameters (such as the modifier **306**) used to generate the second hash value, the routing prefix **308**, and/or the collision count **310**). The modifier value **306** is set in the preceding steps of the second hash computation, and may be retrieved from a suitable data store. The collision count **310** is initially set to zero and may be incremented when an address collision occurs, as explained further below. The routing prefix **308** may be obtained by the address creator in any suitable manner such as from the network **100**. For example, the routing prefix may be determined by listening on the node's network link for advertisements from the local router **106**. For details on how this works on an IPv6 network, see the IETF RFC 2462 "IPv6 Stateless Address Autoconfiguration" and IETF RFC 3971 "Secure Neighbor Discovery (SEND)," which are herein incorporated in their entirety for everything they describe.

[0085] The combined public key and CGA parameter(s) may be hashed, such as by the first hash function which may include the SHA-1 algorithm (step **412** if **FIG. 4**). The result is the first hash value **502** shown in **FIG. 5**.

[0086] It is to be appreciated that the routing prefix **308** is not an essential input to the first hash function of the above step. The omission of the routing prefix **308** from the input to the first hash function could result in the first and second hash values being identical if all parts of the first and second hash inputs are identical. One solution to prevent two identical hash results is to use a different hash function for the first and second hash functions. Additionally or alternatively, two completely different hash values may be instantiated from the SHA-1 algorithm by appending a predetermined and static second modifier (e.g., a byte with the value one or zero) to the inputs of the first and/or second hash value computation. As another example, the two hash values may be computed by dividing a longer hash value, such as a 160-bit SHA-1 hash, into two parts. In this manner, one portion of the hash value may be used as the first hash value, and a second portion of the hash value may be used as the second hash value.

[0087] As shown in **FIGS. 11 and 12**, the ECGA may be generated **1114** based on at least a portion of the computed first hash value. For example, at least a portion of the first hash value may be used as at least a portion of the interface identifier (or other portion) of the ECGA. In one example, as shown in **FIG. 4** and **FIG. 13**, the 64-bit routing prefix **308** and the leftmost 64 bits of the first hash value **502** may be concatenated **414** to form a 128-bit IPv6 address **314**. It is to be appreciated that other protocols may require or allow that the routing prefix and/or interface identifier portions may be selected as other bit lengths as appropriate.

[0088] In step **416** and as illustrated in the schematic diagram of **FIG. 5**, the address may be modified to indicate that the address is an ECGA. For example, as noted above, the "u" and "g" bits in the address (i.e., the user interface portion formed by the first hash value) may be both set to 1, which may identify the address as an ECGA. In one example, the address may be modified to include the hash indicator. For example, where a security parameter Sec is used (as described above with reference to **FIG. 11**), the

security parameter or count value may be included in at least a portion of the ECGA, such as the interface identifier. In one example, the three leftmost bits of the interface identifier may be set to the value for Sec. In this manner, the first three bits of the first hash value may be replaced by the security value or the portion of the first hash value inserted into the interface identifier portion of the ECGA may be shifted over to make room for insertion of the security value in the ECGA. If the time parameter is used to generate the ECGA, the value of Sec or other hash indicator is determined by the allowed computation time and speed of the CPU(s) used for the ECGA generation. After the Sec value (or other hash indicator value) is determined, the value may be treated exactly as if it had been received as input to the ECGA generation. That is, the time parameter need not be encoded into the address bits or communicated when the address is used. Instead, the selected Sec value or other hash indicator is encoded into the address or communicated by other means. The hash indicator may be encoded into any portion of the interface identifier, such as the three leftmost bits of the interface identifier or the node selectable portion of the ECGA. It is to be appreciated that the u bit, the g bit, the hash indicator, and the like encoded into the interface identifier of the ECGA may replace those bits in those locations of the first hash value, or alternatively may be placed in the ECGA and the first hash value located around the encoded parameters. Rather than encoding the ECGA indicator (e.g., u and g bits), the hash indicator and the like into the ECGA address, one or more of the indicators may be sent in protocol messages, implied by the protocol specification, implied by the context of the communication, sent in a certificate, sent in the CGA Parameters data structure defined in IETF RFC 3972 discussed further below, and the like.

[**0089**] The computing device **102** may check whether the address **314** in step **1116** of **FIGS. 11 and 12** (also shown as step **418** in **FIG. 4**) is already in use by another node in the network, i.e., is there an address collision. For example, the computing device **102** may check whether there is an address conflict with respect to the generated ECGA **314**. An address collision may be detected in any suitable manner. For example, different protocols may provide different ways of determining collisions. In the Dynamic Host Configuration Protocol (DHCP), which is a protocol for assigning dynamic IP addresses to devices on a network, the server **104** keeps track of IP addresses and collisions between the addresses it leases. For IPv6, the mechanism is called "duplicate address detection."

[**0090**] If the device that uses the ECGA as its address moves to another network, the ECGA parameters used to compute one or more of the first and second hash values may be modified **1118**, and a new ECGA may be computed based on one or more modified ECGA parameters by returning to computing **104** the second hash value or returning to computing **1112** the first hash value. The CGA Parameters may be modified **1118** in any suitable manner. For example the modifier may be incremented or recomputed, the collision count may be reset to zero or incremented, and the like.

[**0091**] In one example, the collision count **310** value allowed in ECGA generation may be capped. For example, as shown in **FIG. 4** and **FIG. 13**, the encoded CGA Parameters structure **304** may be incremented in step **420**. The collision count may be compared **422** to a predetermined maximum number of allowed collisions, such as three. If there have been a predetermined number of collisions (such as three), the process may terminate. If the process terminates, an error may be reported at step **424**. If the process does not terminate, the address generation process may begin anew with computing a new first hash value **412** with the incremented collision count value.

[**0092**] If no address collision has been detected, the process may terminate **1120** successfully as shown in **FIGS. 11 and 12**. In a successful termination, in one example, shown in **FIG. 4**, a CGA Parameters data structure defined in IETF RFC 3972 is created by concatenating the modifier, routing prefix, collision count, public key (which may be stored in a DER-encoded ASN.1 SubjectPublicKey data item), and possible extension fields.

[**0093**] A self-signed certificate **316** (e.g., a X.509 v3 certificate, or any other suitable certificate) may be created and signed **426** using the public key **300**. The certificate may be in any suitable form and may include any appropriate parameters to aid in verifying the ECGA. For example, the certificate may be created using the public key (which may be stored in a SubjectPublicKey data item). The certificate may include an extension (such as an extnvalue data item) which encodes one or more of the CGA parameters used in generating either or both the first hash value and the second hash value. The CGA parameters included in the certificate or optimized parameter format may include the hash indicator, in addition to or alternative to including the second hash indicator in the ECGA.

[**0094**] In some cases, for example, in protocols using an optimized parameter format (e.g., the CGA Parameters data structure defined in IETF RFC 3972) rather than a certificate, creating and signing **426** may be skipped, and the optimized parameter format may be created instead. The certificate and/or optimized parameter format may be generated in the methods illustrated in either or both **FIGS. 11 and 12**.

[**0095**] As explained above, the first hash value **502** may create a portion of (such as 59 bits) of an interface identifier **504** (which may have 64 bits) in the address **314**. The second hash function increases the cost of brute-force attacks, i.e., where the attacker varies the hash input in order to match its own public key with somebody else's address. In addition to matching the 59 address bits with the first hash value, an attacker must match the predetermined number of zero bits (e.g., the left-most zero bits) with a second hash value, as indicated by a security parameter and/or a zero count parameter which may be included in the ECGA as noted above. As noted above, the predetermined number of zero bits may be N*Sec or the number of zero bits in the selected second hash value using a time parameter. This technique, called hash extension, effectively increases the hash length beyond the 64-bit boundary of earlier proposals. In the N*Sec example, the resulting level of security is equal to a hash length of 59+N*Sec bits.

[**0096**] In some cases, if the routing prefix **308** of the address changes but the address owner's public key **300** does not, the old value of the modifier **306** may be used. In this manner, it may be unnecessary to repeat the brute-force search of steps **404** through **410** of **FIG. 4** (or steps **1104-1108**) if the routing prefix **308** is not included in the computation of the second hash value **500**.

[0097] Authentication of Extensible Cryptographic Addresses

[0098] The authentication of ECGAs may be performed as defined in IETF RFC3972, which is incorporated herein by reference. The procedure for the authentication is the same regardless of whether the ECGA generation procedure takes as input a Sec value that determines the number of zero bits in the second hash value or a time parameter that determines the time used for the ECGA generation.

[0099] By constructing the unique ECGA **314** of the computing device **102** based on its public key **300**, an association between the ECGA **314** of the computing device **102** and its public key **300** is created. This association allows a recipient **106** in **FIG. 6** of a message to authenticate the computing device **102** as the sender of the message by verifying the association and a signature on the message. In order to authenticate a message as coming from a specific ECGA address **314**, as the address owner, the computing device **102** may sign the message with its public key **300**. As illustrated in **FIG. 6**, the computing device **102** then sends to the computing device **106** the packet **600** which may include the message **604**, the signature **606**, and the CGA parameters data structure **316**. The receiving computing device may verify the ECGA such as by using the public key of the sender, one or more of the CGA parameters used to generate the first and/or second hash values, and/or the hash indicator such as the number of zero bits in a predetermined portion of the second hash value.

[0100] Referring now to **FIGS. 6, 7, 8**, and **9**, the computing device **102** sends the message **600** to the recipient computing device **106** which may be configured similarly to or different from the computing device **102** illustrated in **FIG. 2**. The message **600** may contain any suitable content, among other things, such as one or more of a standard and/or non-standard IP header **608** and a message body **610**. The IP header **608** may include one or more of the source address **314**, the destination address **602**, and any other suitable header information **612**. The body **610** of the message **600** may include one or more of a message **604**, a CGA parameters data structure, a digital signature **606** such as that created during the address generation process, and any other suitable information. The message content data **604** may be in plain-text (i.e., not hashed or encrypted). The digital signature **606** may be generated from data included in the message.

[0101] When the computing device **106** receives the authenticated message **600**, the receiving computing device **106** may employ a verifier **700** (**FIG. 7**) to attempt to authenticate the identity of the sender of the message. The CGA verifier **700** in **FIG. 7** may take one or more inputs from the received message **600** to verify the sender. For example, the ECGA verifier may take the senders address **314** and the CGA parameters **316**.

[0102] The verifier **700** may output an indication **702** that the message **600** is trustworthy. If the verifier **700** provides the indication **702** that the message is trustworthy, the verifier knows that the CGA parameters **316** contain the public key **300** of the address owner **102**. The verifier **700** may then use the public key **300** to authenticate signed messages **600** from the address owner **102** or to exchange a session key with the address owner **102**. If the sender cannot be verified, the verifier may output an indication **704** that the message cannot be authenticated.

[0103] **FIGS. 8 and 9** illustrate an example verification process executable by the verifier **700**. Initially, the address **314** may be recognized as an ECGA. For example, the "u" and "g" bits in the address may be compared at step **800** to one. If either bit is zero, the address **314** is a non-CGA address and verification may not be possible or necessary **834**. Alternatively, some other mechanism may be used to indicate that the address is an ECGA, or this may be deduced from the context or other protocol messages. For example, some protocols may require some addresses to be ECGAs. Continuing with step **804**, the security parameter Sec **302** or number of zero bits in the second hash value may be read from the predetermined portion of the ECGA. For example, the security parameter or the number of zero bits may be read from the three leftmost bits of the interface identifier of the address **314**. Next, in step **806**, the value of the public key **300** is determined. For example, the public key may be retrieved from an authorized provider, retrieved from the CGA parameters **316**, or the public key may be retrieved in any suitable manner. The value of the public key may be a SubjectPublicKeyinfo data value.

[0104] In step **808**, the CGA parameters (such as the CGA Parameters data structure) may be found and decoded. For example, the CGA parameters may be received from the address owner with the signed message or it may be communicated in some other way, such as via an online database. The value of the collision count **310** may checked to determine if it is less than the predetermined maximum number of collisions, which as noted above may be three. Thus, in one example, the value of the collision count may be checked to determine if it is 0, 1, or 2. If not, verification fails in step **834**. Next, in step **812**, the value of the data for the routing prefix **308**, which may be stored in the CGA Parameters structure **304316**, may be checked to determine whether it is equal to the routing prefix (i.e., the leftmost 64 bits or another number of bits depending on the protocol) of the address **314** in the message **600**. If they differ, the verification may fail in step **834**. If verification fails, any suitable responsive action may be taken, such as the verifier **700** may issue an indication **704** (of **FIG. 7**) that the authentication of the message **600** (of **FIG. 6**) has failed verification.

[0105] The public key **300** and one or more data values of the CGA parameters (which may be stored in the CGA Parameters structure **304**) may be combined, such as concatenated, in step **816**. The appropriate first hash algorithm (e.g., SHA-**1** algorithm) may then be executed on the combination. The result is the value for first hash value.

[0106] In step **818**, the predetermined part (e.g., leftmost 64 bits) of the computed first hash value may be compared with the interface identifier **504** of the address **314** (i.e., the rightmost 64 bits of the address). Differences in predetermined bit settings, such as the "u" and "g" bits and in the predetermined bits storing the security parameter/time parameter (e.g., the three leftmost bits of the interface identifier) may be ignored. If the 64-bit values differ (other than in the predetermined bit values), the verification may fail **834** and in response, the verifier **700** may issue the indication **704** that the ECGA verification process has failed.

[0107] If the interface identifier of the ECGA matches the selected portion of the computed first hash value, the data values for the public key **300** and the modifier **306** may be

combined **822**. For example, the public key and the modifier **306** from the CCA Parameters data structure may be concatenated. For another example, the routing prefix and collision count in the CGA parameters may be set to zero. The second hash function (e.g., the SHA-1 algorithm) is then executed on the combination. The result is a value for the second hash value. In step **824**, the bits in a predetermined portion (e.g., left-most bits) of the second hash value may be compared **824** to zero or to another predetermined value. In one example, the number of zero bits in the predetermined portion may be compared to the value N*Sec where the security parameter Sec may be extracted from the ECGA. If any one of the N*Sec bits is non-zero, the verification may fail **834**. In response, the verifier **700** may issue the indication **704**, which says the authentication process has failed. If Sec equals zero, then authentication or verification never fails from step **824** since the number of leftmost bits of the second hash value to compare is zero (i.e., 16*0=0).

[**0108**] If the address authentication succeeds at step **832**, the verifier **700** knows that the public key **300** belongs to the owner of the address **314**. The verifier **700** of the computing device has authenticated the address **314** as the ECGA of the computing device **102** which sent the packet **600**. The recipient computing device **106** may then be free to accept the message content data **604** as authentic and may act accordingly.

[**0109**] Referring to **FIG. 10**, an example full life cycle of the authentication process is illustrated as a message sequence diagram. The process begins with the address owner (i.e., the computing device **102**) providing the input to the address creator (i.e., generator **312**). The input may include the public key, the security value and/or time parameter, the initial collision count of zero, and/or an appropriate routing prefix. The address creator receiving the inputs may generate a protocol compliant ECGA based upon the inputs and the public key. The address generator may return the computed ECGA and the CGA parameters to the address owner. The address owner may determine if a collision has occurred with the generated ECGA. If so, the address owner may increment the collision count and request a new ECGA from the address creator. The request may contain one or more of the input parameters, and/or the address creator may store the input parameters for a time until no collision is detected. If no collisions are detected, the address owner can start using the address for communication. The ECGA may then be used to route a message through the network. For example, the address owner may create a message and may sign the message with its public key. The message may be sent to a recipient which may access a verifier to authenticate the message. The sent message may contain the sender's network address, e.g., the ECGA, the CGA parameters, the message, and the signature of the message. The verifier may verify the message, e.g., verify that the ECGA corresponds to the CGA parameters and that the signature is verifiable, and the like.

[**0110**] While using a value for security value **302** and/or time parameter above zero is optional when the computing device generates the packet **600** for transmission to the recipient computing device **106**, the address verifier **700**, may support all security parameter values. More particularly, the authentication procedure may require a constant amount of computation and/or may be relatively fast, regardless of the value of the security or time parameters. Moreover, if verifiers did not support higher security values, there may be less incentive for an address owner to use ECGAs.

[**0111**] The values of modifier **306** and collision count **310** in the CGA Parameters data structure may be ignored in the process of authentication executed by the verifier **700**, except for checking that collision count is in the allowed range in step **810** and including both values in the appropriate hash inputs in steps **816** and **822**.

[**0112**] The verifying party can be the same entity as the message creating party. For example, the computing device **102** creates messages **600** and stores them locally or remotely in one of the storage device **206** or **208** in **FIG. 2**. At a later time, the computing device retrieves the messages **600** and employs the verifier **700**, which may be local, to authenticate the retrieved messages. In fact, the present invention is employable in any situation in which a message is retrieved and its integrity needs to be checked in order to provide some enhanced level of confidence that it is safe to process.

[**0113**] In protocols where the self-signed certificate is used rather than the optimized parameter format (e.g., CGA Parameters data structure defined in IETF RFC 3972), the signature verification may be added. For simplicity, it is suggested that all protocols that use the certificate **316** require the verification of its signature and validity period.

Further Embodiments of ECGAS

[**0114**] ECGA-based authentication may be attractive for performance and cost. For example, ECGAs may be used as a backup when there is no appropriate PKI certificate available, or when contacting an on-line authority has too high a performance penalty. In many cases, there may be no or reduced problems in combining ECGA-based authentication with other equally strong authentication methods.

[**0115**] Applications of Unilateral Authentication

[**0116**] As described above, the authentication mechanism for ECCAs associates an authenticated message with its creator. ECGA authentication of ownership may be applicable to applications other than those described above. Potential applications for ECGA-based authentication include proof of address ownership in secure neighbor discovery and duplicate address (collision) detection, Mobile IPv6 binding update authentication, key exchange for opportunistic IPSec encryption and authentication, and the like.

[**0117**] The protocol (such as IPv6) address autoconfiguration, duplicate address (collision) detection (DAD), and neighbor discovery (ND) may be threatened by denial-of-service attacks. One vulnerability of protocols is that it is not clear who owns an IP address and who is authorized to control the mapping between an IP address and link-layer addresses. For example, in stateless autoconfiguration, an IPv6 node picks an arbitrary IPv6 address in the network where it is located and sends a broadcast message to check that nobody else is already using the same address. An attacker can prevent other nodes from obtaining an IP address by responding to all duplicate address detection messages and claiming to be using any address that the target nodes pick. Implementation of ECGAs may reduce

this dilemma in that duplicate address detection messages could be authenticated using the message address and associated public key.

[0118] Neighbor discovery is the IPv6 equivalent of ARP, i.e., a protocol for mapping IP addresses into link-layer addresses. One vulnerability of mapping IP addresses to link-layer addresses is that an attacker can redirect packets away from their right next-hop destination by spoofing neighbor discovery messages. ECGA addresses can reduce or prevent the attacks described above. In many cases ECGA addresses may be particularly suitable for securing these protocols because the goal is to authenticate an IP address and not some higher-level identifier. A node can create a new ECGA address and then prove its ownership of the address by signing the DAD messages. Similarly, signatures on ND messages may prevent or reduce the spoofing attacks. In both protocols, individual IP packets are signed. Thus, the optimized parameter format may be appropriate for these applications.

[0119] In an example of the Mobile IPv6 draft specification, the mobile node informs its correspondents about its current location by sending binding updates which must be authenticated to prevent denial-of-service attacks. The current Mobile IPv6 draft uses a relatively weak, non-cryptographic authentication method based on the assumption that some network routes are likely to be secure. An alternative may be to use ECGA-based public-key authentication. ECGA addresses may be particularly suitable for this purpose because Mobile IP uses IP addresses as the only node identifier. ECGA-based authentication may also reduce the number of protocol messages. Thus, there may be a case for specifying ECGA-based authentication as an optional optimization to Mobile IPv6. If both end nodes of a connection have an ECGA address, the end nodes may use the ECGAs in a key exchange and create an IPSec security association for encryption and data authentication. This kind of authentication may be done "opportunistically", i.e., whenever both end nodes support ECGA. The benefits of opportunistic protection may include one or more of preventing IP source spoofing attacks, privacy against passive eavesdropping, and reducing filtering of IP packets by network intermediaries. In many cases, the sending and receiving nodes are authenticating each other's IPv6 addresses, not the host names or users. In this case, the ECGA-based authentication may prevent IP source-address spoofing but not DNS spoofing where the attacker interferes with the mapping of domain names to IP addresses. In the future, Secure DNS may give a reasonable assurance of the authenticity of IP addresses. Together, Secure DNS and ECGA may provide strong authentication of hosts by their domain names. There have also been proposals for storing PKI certificates or public keys in the DNS. An advantage of combining ECGAs with Secure DNS is that the name service need not store the public keys. It only needs to provide a reliable binding between host names and addresses, which is its original function.

[0120] For further information see Tuomas Aura, "Cryptographically Generated Address (CGA)", January 2003, submitted for publication, which is herein incorporated in its entirety for everything it describes.

[0121] Non-Cryptographic Addressing

[0122] As noted above, a CGA or ECGA may be indicated within the network address itself, such as by setting both the u and g bits of an IPv6 interface identifier to 1. Likewise, a network address containing a u or g bit which is not equal to one, is not a CGA or ECGA. In this manner, a receiving node may identify a network address as compliant or non-compliant with a CGA or ECGA authentication protocol, which may imply that the sending node associated with the address is compliant or non-compliant with CGA or ECGA authentication protocols.

[0123] Proper identification of an address and its associated node as non-compliant with CGA or ECGA may allow use of non-CGA or non-ECGA authentication methods in message traffic over the same network or protocol which may require or allow CGA or ECGA authentication for some nodes. In this manner, down-level nodes may use a CGA or ECGA network, while still limiting the ability of attacking nodes to present a CGA or ECGA as a non-cryptographic address.

[0124] The identification of an address as a CGA, ECGA, non-cryptographic address, or other type of address may be used to prioritize the messages for processing. For example, messages where the sender address is an authentic CGA or ECGa may be ranked above processing of other message and/or address types. Messages with an authentic non-cryptographic address may have a priority below that of a CGA or ECGa, but may have priority in processing above messages that do not contain a sender's address or which do not have an authenticated CGA, ECGA, or non-cryptographic address. It is to be appreciated that any other prioritization of processing and/or authentication of messages based on the sender's address type may be used.

[0125] Any suitable method may be used to identify a non-CGA or non-ECGA (hereinafter non cryptographic address) while decreasing the ability of an attacking node to spoof a down-level node to gain access to possibly less secure authentication methods. For example, the address bits of a non-cryptographic address may be generated such that the non-cryptographic address cannot be or is probably not an encoding of the hash of a public key. In this manner, a receiving node may properly identify the capabilities of the sending node.

[0126] In one example, a non-cryptographic address may be generated such that one or more node-selectable bits of the non-cryptographic address are constant or filled with a predetermined value which is known to peer nodes. Node selectable bits of a network address may include any bits of the network address which are not mandated or controlled by the protocol. For example, in IPv6, the interface identifier is node-selectable except for some mandated bits which identify features of the address, such as the u and g bits discussed above. Thus, the node-selectable bits of an IPv6 address include those bits of the interface identifier not including bits such as the u and g bits. The predetermined value set within at least a portion of the node-selectable portion of the network address may be any suitable value, such as a series of zeros, or any other suitable value. In one example, as shown in the example address **1400** of **FIG. 14**, the 32-48 leading bits **1402** of the interface identifier **1404** of an IPv6 non-cryptographic address may be set to zero, however, it is to be appreciated that any suitable portion of the non-selectable bits may be preset to indicate a non-cryptographic address. The predetermined value, such as a series of zeros, in a predetermined location of the node-

selectable portion of the address may decrease the likelihood that the interface identifier matches a hash of a public key.

[0127] **FIG. 15** illustrates an example method **1500** of generating a non-cryptographic address as shown in **FIG. 14**. The address owner or authorized third party may generate the non-cryptographic address by setting **1502** the non-selectable portions of the network address in accordance with the governing protocol. For example, the non-selection portions of the network address may include a routing prefix and/or other bits which indicate a feature of the network address, the address owner, the message type (e.g., u and g bits), and the like. The predetermined portion of the node-selectable bits of the network address may be set **1504** to a predetermined value. For example, as noted above, the 38-48 leading bits of the interface identifier may be set to zero. The remaining portion of the node-selectable bits may be set **1406** in accordance with address generation techniques which may include identifying collisions, sub-network identification, and the like.

[0128] The address owner may then use the non-cryptographic address to identify itself when sending messages to a receiving node of the network. **FIG. 16** illustrates an example method **1600** of authenticating a message identifying a sending node with a non-cryptographic address. The receiving node may receive **1602** the message and extract **1604** the network address of the sending node. The predetermined portion of the network address may be compared **1606** to the predetermined value indicating a non-cryptographic address. The predetermined value may be retrieved in any suitable manner, such as through a request to a certified authority or address allocation authority, retrieval from memory, through a request to the sending node, and the like. If the predetermined portion of the network address equals the predetermined value, the receiving node may determine that the sending address is a non-cryptographic address and take **1608** appropriate action, which may be to accept the message without authentication, to initiate an authentication method that is acceptable by down-level devices, and the like. If the predetermined portion of the network address does not indicate a non-cryptographic address, the receiving node may initially determine that the sending address may be a CGA or non-CGA and initiate **1610** the appropriate CGA or ECGA authentication method, such as one of those described above.

[0129] In another example, the non-cryptographic address may be generated by computing the hash of a value (hereinafter called a non-cryptographic value) other than the public key of the address owner. The non-cryptographic value to be hashed may be any suitable value that may not be confused with a public key, such as a random number, link-layer address, a string identifying the node as non-CGA or non-ECGA compliant, and the like. For CGAs governed by RFC-3972, the public key has a specific ASN.1 encoding that may be avoided in selecting the non-cryptographic value.

[0130] With reference to **FIG. 17**, the non-cryptographic address **1700** may include a routing prefix **308** and an interface identifier **1704**. The interface identifier may include one or more bits which are not node-selectable such as the u and g bits **1706**, **1708**. At least a portion of the node-selectable portion of the network address (e.g., the portion **1710** and portion **1712** of the interface identifier **1704** which may exclude the set bits **1706**, **1708**) may contain a hash of the non-cryptographic value. The portion of the node-selectable portion containing the hash of the cryptographic value may be the same portion of the network address which contains a hash of the public key in accordance with the CGA or ECGA protocol.

[0131] **FIG. 18** illustrates an example method **1800** of generating a non-cryptographic address such as that shown in **FIG. 17**. The address owner or authorized third party may generate the non-cryptographic address. The non-selectable portions of the network address may be set **1802** in accordance with the governing protocol. For example, the non-selectable portions of the network address may include a routing prefix and/or other bits which indicate a feature of the network address, address owner, the message type (e.g., u and g bits), and the like. The non-cryptographic value may be determined **1804** in any suitable manner, such as using a random number generator, a predetermined value, a link-layer address, and the like. The non-cryptographic value may be hashed **1806** using any suitable hash function, which may be the same as or different from the first or second hash function used in generating a CGA or ECGA. The predetermined portion of the node-selectable bits of the network address may be set **1808** to at least a portion of the hash of the non-cryptographic value. For example, 59 predetermined bits or 40 right-most bits (other than the non-selectable bits) of the interface identifier may be set to include the left-most values of the hash of the non-cryptographic value. The remaining portion of the node-selectable bits (such as those retained for a security parameter, hash indicator, and the like) may be set **1810** in accordance with address generation techniques which may include identifying collisions, sub-network identification, and the like. The non-cryptographic value used to generate the non-cryptographic address may be saved in an appropriate data store.

[0132] The address owner may then use the non-cryptographic address to identify itself when sending messages to a receiving node of the network. **FIG. 19** illustrates an example method **1900** of authenticating a message sent by a sending node identified by a non-cryptographic address such as that illustrated in **FIG. 16**. The sending node may generate **1902** a message. The message may include the non-cryptographic address as the sending node identifier. The message may also contain the non-cryptographic value used to generated the address of the sending node. The non-cryptographic value may be included in any suitable location and/or manner in the message. For example, the non-cryptographic value may be included in a certificate in the body of the message, the non-cryptographic value may be included in a header and/or body of the message, and the like. Alternatively, the non-cryptographic value may be omitted from the message. For example, the receiving node may have prior knowledge of the non-cryptographic value and may retrieve the non-cryptographic value from a data store, the receiving node may retrieve the non-cryptographic value from a source other than the message, and the like.

[0133] The receiving node may receive **1904** the message and extract **1906** the network address of the sending node. The receiving node may determine **1908** the non-cryptographic value in any suitable manner. For example, the receiving node may extract the non-cryptographic value from the message, may retrieve the non-cryptographic value from memory, may request that the sending node or other

node send the non-cryptographic value in a subsequent message, may receive a separate message from the sending node indicating the non-cryptographic value, and the like. The receiving node may hash **1910** the non-cryptographic value. The predetermined portion of the network address may be compared **1912** to at least a portion of the computed hash of the non-cryptographic value. If the predetermined portion of the network address equals the portion of the predetermined value, the receiving node may determine that the sender's address is a non-cryptographic address and take **1914** appropriate action. Appropriate action may include any suitable action such as accepting the message without authentication, to initiate an authentication method that is acceptable by down-level devices, and the like. If the predetermined portion of the network address does not indicate a non-cryptographic address, the receiving node may determine that the sending address may be a CGA or ECGA and initiate **1816** the appropriate CCA or ECGA authentication method, such as one of those described above. It is to be appreciated that the extension to CGAs described above may be implemented in generating and authenticating a non-cryptographic address including a hash of a non-cryptographic value.

[**0134**] At least a portion of the non-cryptographic value may be known to or discoverable by the receiving node to authenticate the non-cryptographic address. For example, the non-cryptographic value may be a combination, such as a concatenation, of a plurality of input values. For example, the non-cryptographic value may be a concatenation of a first input value and a second input value such as a random number, a counter value, or an identifier specific to the node that will use the address. The receiving node may have prior knowledge of the first input value, e.g., through determination from an authorized provider, through determination from information in a prior or related communication, through determination from context information, or through knowledge of a predetermined value for that sending node or for all non-CCA identifier nodes. Any suitable first and second input values may be used, as noted above, such that the resulting combination in the non-cryptographic value may not be confused with (or be similar to) a public key and/or the input string used in generating a corresponding CGA and/or ECGA.

[**0135**] The message may include the hash of the non-cryptographic value and the second input value (e.g., random number) used as a portion of the non-cryptographic value, which as noted above may be included in the sender's address, in a header of the message, and/or body of the message. In this manner, the message may not include an unencoded reference to the first input value, e.g., in plain text. Similarly, if the receiving node has prior knowledge of the second input value, such as the random number, the message may include the hash of the non-cryptographic value and the first input value, which as noted above may be included in the sender's address, in a header of the message, and/or body of the message. In this manner, the message may not include the second input value. Similarly, if the receiving node has prior knowledge of both the first and second input values of the non-cryptographic value, neither the first nor the second input values may be included in the message other than in their encoded versions within the hash of the non-cryptographic value. Although the description here includes a first and second input value, it is to be appreciated that more than two input values may be used as appropriate to form the non-cryptographic value.

[**0136**] In one example, the first input value may be predetermined for a particular node (or type of node) and known to a receiving node. The second input value of the non-cryptographic value may be one of a plurality of possible second input values. The receiving node may know the plurality of possible second input values, but may not know which one of the possible second input values was used to generate the non-cryptographic value. If the number of possible second input values is small as compared to the computational power of the receiving node, the receiving node may try one or more of the possible second input values to determine if any one of the possible second input values was used.

[**0137**] In one example, as shown in **FIG. 20**, a first input value **2002** and a second input value **2004** may be concatenated to form a non-cryptographic value **2000**. The first input value may be any suitable value, such as one or more of a subnet prefix, a link-layer address, NetBIOS, DNS name, DNS domain suffix of the sending node, a predetermined non-cryptographic address identifier, and the like. The second input value may be any appropriate value such as a random number, a member of a set of possible second input values, and the like, which may be generated and/or modified in a manner similar to that described above with reference to the modifier value. The non-cryptographic value **2000** may be input to a hash function **2006**, which may be the same as or different from the first or second hash functions described above with reference to **FIG. 5**. The result of the hash function may be used to determine the hash **2008** of the non-cryptographic value, such as by selecting from the plurality of results of the hash function over a period of time specified by a time parameter (discussed further below), using a security value, or the direct result of the hash function **2006**. If the time parameter is set to approximately zero, the method may select the first hash result generated. In this manner, the computation time spent in generating the non-cryptographic address may be limited.

[**0138**] At least a portion of the hash of the non-cryptographic value may be inserted into at least a portion of the non-cryptographic address. For example, as shown in **FIG. 20**, at least a portion of the selected value **2008** may be concatenated with the routing prefix **308** and the preset bits **2016** (in accordance with the governing protocol). In this manner, a portion of the selected hash value **2008** may be inserted as portion **2014** of the non-cryptographic address, another portion may be inserted as portion **2018** of the non-cryptographic address, and/or another portion may be inserted as portion **2020** of the non-cryptographic address.

[**0139**] As noted above, the first and/or second input value may be communicated to the receiving node in any suitable manner, such as through a header in the message, in the body of the message (such as in a certificate), or may be known to the receiving node. For example, as shown in **FIG. 20**, the second input value **2004** may be communicated to the receiving node through the sender's non-cryptographic address by inserting the second input value in the portion **2020** of the non-cryptographic address. It is to be appreciated that the order and lengths of the concatenations to form the non-cryptographic address may be modified to meet the needs and/or concerns of a particular protocol and/or node.

In one example, the second input value may be inserted into an address portion **2014** having 16 bits and the portion of the result **2012** may be inserted into address portions **2018** and **2020**, which together may have 48 bits. At least a portion of the non-cryptographic value (e.g., the first input value and/or the second input value) used to generate the non-cryptographic address may be saved in an appropriate data store.

[**0140**] **FIG. 21** illustrates an example method **2100** of generating a non-cryptographic address such as that shown in **FIG. 20**. The first input value may be determined **2102** in any suitable manner, such by identifying the predetermined first input value for that address owner. For example, the first input value may be one or more of a subnet prefix, a link-layer address, NetBIOS, DNS name, DNS domain suffix of the address owner, a predetermined value indicating a non-cryptographic address, and the like. In one example, the predetermined value indicating a non-cryptographic address may be a string such as "this is not a cryptographic address" or any other suitable string or value.

[**0141**] The hash parameters may be initialized **2104**, in any suitable manner. For example, the second input value may be initialized such as by generating a random number, selecting a random number from a predetermined set of numbers, and the like. The time to compute the hash value may be set to zero and started running to track the time used in computing the hash value. The collision count may be set to zero. The first and second input values may be combined **2106**, such as concatenated. The combination of the first and second hash values may be hashed **2108** to generate a hash output. The time to compute the hash output may be compared **2110** to a predetermined or input time parameter. If the time parameter has not been exceeded, one or more portions of the non-cryptographic value may be modified **2122**. For example, the first and/or second input values may be modified, such as by incrementing the value by **1**, selecting the next input value in the set of possible input values, generating another random number, and the like. The newly modified input value may be combined **2106** and another hash value computed **2108**. This process may repeat until the time parameter has been exceeded. If the time parameter has been exceeded, the hash value of the non-cryptographic value may be selected **2114** from the generated hash values as described above with reference to **FIG. 12**.

[**0142**] The non-cryptographic address may be generated **2118** in any suitable manner based on the hash of the non-cryptographic value. For example, the routing prefix may be concatenated with at least a portion of the selected hash of the non-cryptographic value. If the second input value is known to or can be determined by a receiving node (such as by trying one or more of a set of possible second input values), the second input value may be omitted from the non-cryptographic address. In another example, the second input value may be concatenated with the routing prefix and the hash of the non-cryptographic value to form the non-cryptographic address.

[**0143**] The generated non-cryptographic address may be compared **2120** to existing addresses to determine if an address collision occurs. If a collision occurs, one or both of the first and second input values may be modified **2116**, and the process returned to combining **2106** the first and second input values. This process may be repeated until no collision with the generated non-cryptographic address is detected or the number of collisions exceeds a maximum collision parameter. If no collision is detected, the generation of the non-cryptographic address may end **2124**. The resulting non-cryptographic address may be provided to the address owner, stored in a data store, and the like. Additionally, one or more of the input values and/or the non-cryptographic value used to generate the cryptographic address may be stored in an appropriate data store. In one, if address owner is a down-level node, a network administrator or address service may compute the non-cryptographic address for the address owner and communicate the address to the address owner such as through a network connection or configuration interface, such as a user interface or network management tool. The non-cryptographic address may be used by the address owner.

[**0144**] As noted above, the sending node as the owner of a non-cryptographic address may be a down-level device. As such, the sending node may not be able to hash a public key and/or form a certificate. Rather, the sending node may itself use legacy protocols to authenticate messages from other nodes (e.g., responses sent by receiving nodes). In some cases, the address owner may not even know that the receiving node may verify its address as a non-cryptographic address. It is to be appreciated that the hash extension method described above may be used to extend the non-cryptographic generation and authentication. In this manner, the sending node may indicate that it is a down level device, and as such, unable to verify a CGA or ECGA, and the receiving node may determine that the sending node has a non-cryptographic address.

[**0145**] In one example, the receiving node may initiate authentication of a message based on the non-cryptographic address of the sending node. However, during the process of authentication, the receiving node may identify the address of the sending node as a non-cryptographic address. In some cases, if a non-cryptographic address is detected, the authentication of the non-cryptographic address may include no further action to authenticate, or alternative (or legacy) protocols may be implemented.

[**0146**] **FIG. 22** illustrates an example method **2200** of determining if an address generated by a method similar to **FIG. 21** can be identified as a non-cryptographic address. The method **2200** may be implemented in conjunction with the authentication methods of CGAs and/or ECGAs. The methods may be implemented in serial, in parallel, and/or may be combined to reduce computation, and the like. The sending node may generate **2202** a message. The message may include the non-cryptographic address as the sending node identifier. The message may also contain the none of, one of, or both of the first input value and the second input value used to generate the address of the sending node. For example, as noted above, the address of the sending node may include the routing prefix and the at least a portion of the hash of the non-cryptographic value. The message may omit or exclude referencing or disclosing an unencoded version of either the first or the second input value.

[**0147**] The receiving node may receive **2204** the message and extract **2206** the network address of the sending node. The receiving node may determine **2208** the first input value in any suitable manner. For example, the receiving node may extract the subnet prefix (or link-layer address, and the like)

from the address as the first input value, may retrieve a predetermined non-cryptographic identifier from memory as the first input value, may request that the sending node send the first input value in a subsequent message, may receive a separate message from the sending node indicating the first input value, and the like. The receiving node may determine **2210** the second input value. For example, the receiving node may select one second input value from a set of possible second input values, extract the second input value from the address of the sending node, and the like. In one example, the set of possible input values may be a set of integers from 0-16 and the receiving node may select one of the possible second input values. It is to be appreciated that the number, range, and/or values of the possible second input values may be predetermined based on one or more factors including the length of the second input value, computational power of down-level nodes and/or receiving nodes, the size of the network, the number of down-level nodes, and the like.

[**0148**] The receiving node may generate **2212** the non-cryptographic value in any suitable manner, such as by concatenating the first and second input values, hashing the first and/or second input value in accordance with an extended method as discussed above with ECGAs, and the like. The receiving node may hash **2214** the non-cryptographic value. The predetermined portion of the network address may be compared **2216** to at least a portion of the computed hash of the non-cryptographic value. If the predetermined portion of the network address equals the portion of the hash of the non-cryptopgraphic value, the receiving node may identify **2218** that the sending address is a non-cryptographic address and take appropriate action, which may be to accept the message without authentication, to initiate an authentication method that is acceptable by down-level devices, and the like. If the predetermined portion of the network address does not indicate a non-cryptographic address, the receiving node may return to determining **2210** the second input value, and select the next possible second input value in the set of input values and continue. If after all possible second input values are used and none of the generated hash values indicate a non-cryptographic address, the receiving node may take **2220** appropriate action. For example, the receiving node may determine that the sending address may be a CGA or ECGA and initiate the appropriate CGA or ECGA authentication method, such as one of those described above. Alternatively, the receiving node may determine that the address is neither a non-cryptographic address, a CGA, nor an ECGA, and may take appropriate action, such as by dropping the received message, sending an error message, and the like.

[**0149**] It is to be appreciated that the extension to CGAs described above may be implemented in generating and authenticating a non-cryptographic address including a hash of a non-cryptographic value. For example, the first and second input values may be concatenated and hashed by a second hash function. The security parameter or time parameter may be used to select the second hash value. A hash indicator, such as a security parameter Sec, may be encoded into the address bits or communicated in another way to specify the set of permissible values of the second hash function. The non-cryptographic value may include one or more of the routing prefix, the first input value, the second input value, a collision count, and the second hash value. The non-cryptographic value may be hashed by a first hash function, and at least a portion of the result of the first hash function may be inserted into the non-cryptographic address. A similar process may be used to authenticate the non-cryptographic address.

[**0150**] Human Input Security Codes

[**0151**] As noted above, the hash extension technique used to generate an ECGA may be used to generate extended hashes in other applications. For example, a similar hash extension technique may be used to increase the strength of one-way hash functions and/or decrease the number of bits in any situation where some external requirement limits the number of hash bits, and that limit is below what is (or may be in the future) considered secure against brute-force attacks.

[**0152**] For example, to decrease the length of human entered security codes (and maintain the same security), and/or to increase the strength of a human entered security code (and maintain the length of the security code), the security code may be generated and/or authenticated using a process similar to the hash extension of ECGAs. Any security code that is manually communicated (e.g., entered, verified, compared, transmitted, and the like) may be generated using an extended hash method to increase the security and/or limit the length of the security code while maintaining the same level of security. Although a security code to allow communication between a media device and a remote control is used as an example below, it is to be appreciated that any key exchange protocol (e.g., TLS handshake, Internet Key Exchange (IKE), Security Support Provider Interface (SSPI)) may use a security code that is required to be authentic, and not necessarily secret, (e.g., network settings, public keys of devices such as garage doors, configuration settings, and the like) may have extended security and/or decreased length.

[**0153**] **FIG. 23** illustrates a system using a human **2310** as a secure channel to communicate a security code. As shown in **FIG. 23**, a second device **2302** may need to be authenticated by a first device **2304** to allow use of the second device with the first device. The first and second devices may be any suitable devices, as noted above, which may authenticate each other, or verify authorization of each other using a security code, or agree on one or more parameters using a security code. In this example, the second device **2302** is a media device such as a television, and the first device **2304** is a remote control.

[**0154**] The first device and the second device may agree on some data **2305** that is required as a part of a security protocol, such as key distribution. This data may be sent by the first device to the second device, sent by the second device to the first device, or created as a result of a protocol in which both devices send and receive messages, such as a cryptographic key exchange. For example, the data **2305** may contain one or more components such as a device or user identifier, a public key, a Diffie-Hellman exponential, a session key, a cryptographic authenticator, a cryptographic protocol suite identifier, a time or date value, a configuration parameter, an entity identifier, or a security policy description.

[**0155**] In the example of **FIG. 23**, the first device **2302** may be a remote control, the security code may be hash of the first device's public key, and the data **2305** may be the

public key. The second device may be a media device that authenticates the first device using the public key before or after the human entering the security code.

[0156] The user **2310** may receive a security code **2306** that is associated with the first device. In one example, the first device **2304** may communicate a security code to the user **2310** in any suitable manner. For example, a user interface, such as a display of the first device may communicate the security code to the user. In another example, the security code may be passively communicated to the user by the first device. For example, the security code **2306** may be made available to the user **2310** through a label or sticker that can be referenced by the user when retrieving the security code. For example, the security code **2308** may be on a sticker that is attached to the first device. Additionally or alternatively, the user may discover the security code through a message from a third party. For example, the user may receive a certificate through paper mail which informs the user of the security code. In another example, the user may access the security code through other media, such as a telephone call or through a network communication. In another example, the user may know the security code and not require a communication of the security code from the first device.

[0157] To have the human user **2310** act as a secure channel, the user may communicate the security code **2306** to the second device **2302** using any suitable means. For example, in the case of a media device or other electronic device, the user may directly input the security code through an alpha-numeric keypad or keyboard of the second device. In another example, the user **2310** may communicate the security code to the second device, such as by inputting the security code into a computing device and transmitting the security code through a communications network to the second device. In another example, the user may communicate the security code to the second device through a telephone link. For example, the user may enter the security code through the numeric keypad of the telephone or may speak the security code. It is to be appreciated that any means of communication may be used to communicate the security code from the user to the second device.

[0158] In another example, the second device may communicate the same security code **2306** to the user and the user may compare the security code received from device **2304** to the security code received from device **2302**. If the two security codes are equal, the user may signal acceptance to the second device, for example, by pressing a key on the second device or by entering some other another affirmative input to the second device. In a security protocol, this comparison of two codes may be equal to or alternative to the user entering the code to the second device. The comparison of two codes may have the benefit of reducing the effort required from the user or requiring a simpler input device compared to the user entering security code.

[0159] In addition to the second device authenticating the first device, the first device may authenticate the second device. In one example, the human user may transfer a second security code to the first device, which may be used for the second authentication. In another example, the user may signal the acceptance of the input by the second device to the first device by pressing a key or by entering some other affirmative input to the second device. In yet another example, the user may signal to both devices the successful comparison of two security codes, each of which is output from one of the devices. In these examples, the data **2305** may include parameters of both devices such as the public keys of both devices.

[0160] The authentication of the first device by the second device may authorize the first device to access the first device, or to control the first device, or to use the first device as an input or output device, The authentication of the second device to the first device may likewise authorize such actions.

[0161] In another example, rather than the devices agreeing on data **2305**, the user may agree on a data value **2305** with a single device. For example, with reference to **FIG. 23**, a user **2310** may retrieve a security code **2306** that is not associated with the first device **2304**. As noted above, the user may retrieve the security code in any suitable manner, including for example, through a label or sticker associated with the first device, a certificate from paper mail, a telephone call or through a network communication, from the user's own memory if known to the user, and the like. The second device **2302** may receive the data from a communication network, retrieve the data from a storage device such as its memory, or the data may be entered to the second device by the user. To authenticate or authorize the data **2305** to the second device **2302**, the user may communicate the security code **2306** to the second device. The user may communicate the security code to the second device in any suitable manner, which as noted may include, for example, directly inputting the security code through an alpha-numeric keypad or keyboard of the first device, inputting the security code into a computing device and transmitting the security code through a communications network to the first device, entering the security code through the numeric keypad of the telephone or by speaking the security code to a human who may then transfer the security code or an authorization message to the first device, and the like. Alternatively, the second device may output the security code, which the user then compares with a value already known to the user. The user may then signal the successful comparison of the security codes to the second device using an input such as a key press. In one example, the second device may authenticate or authorize the data before using the data to configure device settings, communication settings or access control settings. In another example, the second device may authenticate or authorize the data before allowing the user to take a desired action, such as access the second device, communicate with the second device, and the like. Authentication of the input security code by the first device is discussed further below.

[0162] **FIG. 24** illustrates an example schematic diagram of security code generation and **FIG. 25** illustrates a flowchart of an example method of generating a security code. Like the generation of an ECGA as described above with reference to **FIGS. 3, 4, 5**, and **11-13**, a security code to be input by a user may be generated by using an extended hash. For example, the input to the second hash function **520** may include data **2305** and a modifier **306**. The data may be any suitable information. For example, the data may contain one or more device or user identifiers, public keys, Diffie-Hellman exponentials, session keys, configuration parameters, identifiers of cryptographic algorithm suites, device configuration parameters, communications parameters, and

any security-critical information, The modifier **306** and the data **2305** may be combined in any suitable manner. For example, as shown in **FIG. 24**, the modifier is concatenated with the data. However, it is to be appreciated that the order and combination of inputs to the second hash function **520** may be accomplished in any suitable order and manner.

[**0163**] The second hash function **520** may hash the input value to generate a hash value **500**. The termination condition **2404** may be examined. The termination condition may be a test for the inclusion of the second hash value in a predetermined set of hash values. For example, as noted above, the termination condition may be a security value, such as a number of zero bits or N*Sec. The security value N*Sec may be compared to the number of zero bits in the left-most portion of the hash value **500**. If the hash value **500** does not have an equal or greater number of zero bits (or any other predetermined bit pattern), then the termination condition is not satisfied. Additionally or alternatively, the termination condition may be a time parameter. The time parameter may be compared to a total time to determine the second hash value. If the time parameter is not exceeded, then the generation of the second hash value may continue. Additionally or alternatively, the termination condition may include a probability threshold indicating a probability of generating a more secure hash value **500** than the ones previously found. As noted above, the probability threshold may be compared to a computed probability of generating a more secure hash value. If the probability of generating a more secure hash value is greater than or equal to the probability threshold, the generation of the second hash value may continue.

[**0164**] If the generation of the hash value continues (e.g., the termination condition is not met), then the modifier **306** may be modified **2410** and the new modifier value may be combined (e.g., concatenated) with the data **2305**. The generation of the hash value **500** may continue until the termination condition is met. It should be noted that, in some implementations, the modifier is not necessarily a separate data field but any variable part of the data may act as the modifier.

[**0165**] If the termination condition is met, a second hash value may be selected from the plurality of generated hash values **500**. The second hash value may be selected in any suitable manner. For example, the second hash value may be selected as the 'most secure' hash value (e.g., the hash value **500** which produced the highest Sec value) from the plurality of hash values **500** generated by the second hash function. The modifier that produced the highest found Sec value may be stored to be used for the remainder of the security code generation.

[**0166**] Once the second hash value is selected and the associated modifier stored, the security code **2306** may be generated by determining the first hash value **502** using a first hash function **510**. As noted above with respect to **FIG. 5**, the first and second hash functions may be any suitable hash function, and may be the same or different hash functions. The input to the first hash function may be determined in any suitable manner. For example, as shown in **FIG. 24**, the input to the first hash function may include a combination of the modifier **306** and data **2305**. In some cases, the data **2305** input to the first hash function may not be the same as the data input to the second hash function, or the data may be only partially the same. In some cases, one or more optional additional data values **2412** may be included in the input to the first hash function.

[**0167**] To prevent the first hash value from being equal to the second hash value, one or more optional parameters may be modified to ensure that the hash inputs are different, e.g., the input to the first hash function may be different from the input to the second hash function, a static variable may be appended to one or more of the inputs, and the like. In other cases, the first hash function may be different from the second hash function. In another example, the second hash value may be a predetermined portion of a larger hash value, and the first hash value may be another portion of the larger hash value.

[**0168**] The result of the first hash function **510** is the first hash value **502**. At least a portion of the first hash value **502** may be used to form at least a portion of the security code **2306**. For example, as shown in **FIG. 24**, at least a portion of the first hash value **502** may form the entire security code **2306**. Any suitable portion of the first hash value may be used, such as the leftmost portion of the first hash value and having a length of the correct number of bits for the security code. In other cases, the at least a portion of the first hash value may be combined, such as concatenated with one or more code parameters. The one or more code parameters (not shown) may include, for example, a manufacturer code, a date indicator, a location indicator, a network indicator, and the like. It is to be appreciated that any suitable predetermined code parameter may be used to form the security code **2306**.

[**0169**] Optionally, a hash indicator **550**, such as Sec, may be combined with at least a portion of the first hash value to form the security code **2306**. This may be useful, for example, if the termination condition is based on a time parameter. However, it may not be necessary to encode the hash indicator into the security code if the hash indicator value is fixed, is known from some other communication, or is known from the protocol context.

[**0170**] In some cases, the generated security code **2306** may be compared to other known or discoverable security codes. If there is no collision, the security code **2306** may be stored associated with the first device. In some cases, the security code containing a portion of the first hash value may be encoded to allow easier understanding and/or transmission by a human user. For example, the security code may be encoded into a series of alpha-numeric characters. As noted above, the security code may be prepared to be communicated to the user **2310**, such as printed on a label, coded into the first device for display to the user, printed into a certificate for transmission or paper mail to the user, and the like.

[**0171**] If there is a collision with any existing security code and the application or protocol requires the security codes to be unique, one or more input values to the first hash function may be modified. In one example, the modifier may be adjusted, and the security code generation may return to generating the hash values **500** using the second hash function **510**. In another example, a collision count (which may be initialized to zero) may be incremented **420**, as shown in **FIG. 4**. The collision count may be one of the additional data values **2412** as input to the first hash function **510**. In this manner, a new first hash value **502** may be

determined using the incremented collision count with the other original input parameters (e.g., modifier **306** and data **2305**). The collision count may also be an optional input value input to the second hash function **520**. However, the collision count value may be initialized to and remain zero in generating the second hash value.

[**0172**] If the application or protocol requires the security codes to be unique, the resulting security code from the newly generated first hash value **502** may be compared to known and/or discoverable security codes to determine if a collision occurs. The process may repeat until no collision occurs.

[**0173**] **FIG. 25** illustrates a flowchart of an example method of generating a security code which is similar to the method of generating an ECGA of **FIG. 13**. However, it is to be appreciated that the security code may be generated in a manner similar to any of the above described extensible hashes described above. As shown in **FIG. 25**, the data **2305** may be determined **2502**. As noted above, the data may be any suitable information on which two devices, or one device and the user, want to agree. The modifier may be initialized **2504**. For example, the modifier may be initialized to a random value, which in some cases may be **16** random octets. If a collision evaluation is to be done, a collision count may be initialized to zero or other suitable value.

[**0174**] The termination condition may be determined **2506**. As noted above, the termination condition may be a predetermined bit pattern in the second hash value. For example, at least a number of N*Sec zero bits may be desired in the left-most bits of the second hash value. Additionally or alternatively, generation of the second hash value may accept the 'most-secure' hash value generated during a given time parameter. As noted above, the time parameter may be modified over time to adjust for computational expectations in the future based on the computational speed and capabilities of the computing device generating the security code. Additionally or alternatively, the termination condition may include a probability threshold indicating the probability of generating a more secure hash value result of the second hash function in a given period of time, which may be the remaining time that is allocated to the security code generation. The probability threshold may be set to any suitable parameter depending on the computation time desired by the computing system generating the second hash value, the security desired in the security code, and any other suitable parameters.

[**0175**] To form the input to the second hash function, the data may be combined **2508** with the modifier in any suitable manner. For example, the data may be concatenated with the modifier. The combined values may then be hashed by the second hash function to generate **2510** a hash value. Based on the result of the second hash function, a hash indicator may be determined **2512**. For example, the number of zeros in the left-most position of the hash value may be counted and an associated Security value (Sec) may be determined as the hash indicator as described above.

[**0176**] The second hash value may be selected **2516** in any suitable manner, such as by selecting the most 'secure' of the plurality of hash values generated by the second hash function. For example, the second hash value may be set to the hash value having the highest found Security value (e.g., number of left-most zero bits). More particularly the hash indicator may be the Security value and may be compared to a prior determined hash indicator. The hash value associated with the highest Security value (hash indicator) may be selected as the current second hash value. The modifier and the hash indicator associated with the selected second hash value may be stored in any suitable manner, such as in a data store. The second hash value may be updated as subsequent results of the second hash function are generated until the termination condition is met.

[**0177**] The termination condition may be evaluated **2518**. For example, the time parameter may be compared to a run time of generating multiple hash value results from the second hash function. The termination condition may additionally or alternatively determine whether the second hash value belongs to a set of acceptable hash values defined by the hash indicator. For example, the termination condition may compare the hash indicator to a desired pattern of bits in a portion of the second hash value. For a more specific example, the hash indicator may indicate the number of 0 bits in the left-most portion of the hash value, and the generation of the second hash value may terminate when the hash indicator equals or exceeds the security parameter. Additionally or alternatively, the termination condition may include determining a probability of generating a more secure second hash value as compared to the existing selected second hash value. If the probability is lower than or equal to a probability threshold, the generation of the second hash function may terminate.

[**0178**] If the termination condition is not met, the modifier may be modified **2520** in any suitable manner, such as incremented. It should be noted that the modifier **306** need not be a separate data field but may be a variable part of the data **2305**. The process of generating the second hash value may return to combining the modifier with the data to continue generating results from the second hash function as potential second hash values.

[**0179**] If the termination condition is met, the modifier of the selected second hash value may be combined **2522** with the data **2305** and optionally with additional data **2418**. The additional data may be any suitable information that two devices, or one device and a user, want to agree on. The reason for including specific information in the additional data **2418** but not in the data **2305** may be that the additional data changes more often than the data, or that the additional data is not available when the second hash is generated and the modifier determined. The data or additional data may also include one or more optional parameters, such as a collision count. The components of the input to the first hash function may be combined in any suitable manner, such as concatenated. The combination may then be hashed by the first hash function to generate **2524** a first hash value.

[**0180**] The security code may then be generated **2526** from at least a portion of the first hash value. For example, a portion of the first hash value, such as the correct number of bits from the left-most portion of the first hash value, may be selected to form the security code. The portion of the first hash value may be combined with other information in any suitable manner, such as concatenated. For example, one or more of a manufacturer identifier, a date indicator, the hash indicator, the modifier, and the like may be concatenated with the portion of the first hash value to generate the security code.

[0181] If the application or protocol requires the security codes to be unique, the generated security code may be compared to other known or discoverable security values to determine if there is a collision. If no collision is found, then the generated security value may be returned **2536**. It is to be appreciated that any suitable computing device may generate the security code. For example, the first device may compute its own security code, or alternatively, a third party, such as the manufacturer of the first device may compute the security code and store the security code and its associated modifier and/or hash indicator in a data store of the frits device. In another example, the user may compute the security code (using the first device or another device) and communicate the security code to the firs device. In yet another example, the second hash value may be computed by the manufacturer and the first hash value may be computed by the device while including some information that is not available to the manufacture in the additional data.

[0182] If there is a collision, an input parameter to either or both of the first hash function and the second hash function may be modified. For example, a collision count may be incremented in any suitable manner, such as incremented by one. The collision count may be compared to a collision threshold. If the collision count equals or exceeds the collision threshold, appropriate action may be taken. For example, the process of generating a security code may be halted and/or an error message may be sent. If the collision threshold is not exceeded, then the process of generating the security code may return to combining **2522** the modifier, collision count as a second hash parameter, the data value and optional additional data values until a suitable security value is determined or the process is halted.

[0183] If a security parameter is generated, the security parameter may be encoded in any suitable manner for use. For example, the security parameter may be encoded into an alpha-numeric combination. In some cases, the encoding may select letters and/or numerals that are readily distinguished from each other and/or tend not to be prone to transcription errors. For example, the numbers zero and one, and/or the letters 'I' and 'O' may be avoided to reduce errors in transcription. Those skilled in the art may appreciate that other letters or numerals may be avoided or added as suitable, such as avoiding sequential repetitions of a single numeral or letter. It is to be appreciated that alternative embodiments are not limited to Roman characters and Arabic numerals. For example, symbols and other alphabets and numeric systems may be used.

[0184] The generated security code, communicated to either the first and/or the second device by a human user may be compared with data previously or subsequently known by the second device.

[0185] **FIG. 26** illustrates a flowchart of another example method **2600** of comparing data with a security code similar to that described above with reference to **FIG. 8**. For the second device to authenticate the first device and/or the user, the second device may receive **2602** the security code as communicated by the user. If the security code as entered is encoded, the first device may decode the security code. The first device may retrieve **2604** one or more hash parameters. The hash parameters may include the hash indicator, such as the security parameter, the modifier, the data, the additional data. The data or the additional data may include the collision count. For example, the first device may retrieve one or more of the hash parameters from memory, may parse the hash parameter value from the security code (e.g., the value is concatenated with the portion of the first hash value to form the security code), and the like. It is to be appreciated that different hash parameters may be retrieved in different manners.

[0186] If the data or additional data includes a collision count, the collision count may be compared **2608** to a predetermined threshold. If the collision count exceeds the predetermined threshold, the authentication of the security code may fail, and appropriate action may be taken **2606**. For example, appropriate action may include sending an error message to the user, stop attempting to authenticate the security code, denying access to the first device, and the like.

[0187] If there is no collision count, or if the collision count does not exceed the threshold, the one or more retrieved parameters may be combined **2610** and hashed **2612** by the first hash function. A predetermined portion of the result of the first hash function may be compared **2614** to a portion of the input security code. If the values differ, the authentication may fail and appropriate action may be taken **2606**.

[0188] If the values match, the hash parameters, such as a modifier value and a data value, and additional data values, may be combined **2616** in a predetermined manner to form the input to the second hash function. The combination may be hashed **2618** by the second hash function. The hash indicator for the calculated second hash value may be determined **2620**, for example, by decoding it from the security code or by using a fixed value or a value specified by the protocol context. It may be determined whether the second hash value belongs to a set of acceptable hash values that is defined by the hash indicator. For example, the number of zero bits in the left-most portion of the second hash value may be counted. The hash indicator for the calculated second hash value may be compared **2622** to the retrieved hash indicator for the received security code. If the value of the second hash value does not belong to the set of acceptable hash values (e.g., if the number if zero bits is smaller than what is required by the hash indicator), the authentication may fail and appropriate action may be taken **2606**. If the second hash value belongs to the set of acceptable hash values (e.g., if the number of zero bits is greater than or equal to what is required by the hash indicator), the security code may be considered verified and appropriate action may be taken **2634**. For example, the second device may process the data **2305** or a success message may be communicated, access may be allowed, and the like.

[0189] Using a hash extension in generating a security code, the strength of the security code may be increased and/or the amount of entropy transferred by the human user can be kept reasonably small. As computing power increases over time, the time parameter and/or security value may be increased to accommodate the need for stronger security codes. For example, the number of zero bits (e.g., the length of the hash extension) can be increased. For example, hash extensions of 32 bits may be trivial to compute with current computing devices. However, using the hash extension technique, the human input security code may have 30 bits of entropy, with an increased strength against brute-force attacks of 30 bits+32 bits of hash extension=62 bits, which

may be too difficult for a typical attacker to break during a real-time man-in-the-middle attack.

[0190] Those skilled in the art will realize that storage devices utilized to store program instructions can be distributed across a network. For example, a remote computer may store an example of the process described as software. A local or terminal computer may access the remote computer 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 by executing some software instructions at the local terminal and some at the remote computer (or computer network). 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 circuit, such as a DSP, programmable logic array, or the like.

[0191] Having described and illustrated the principles of our invention with reference to the illustrated embodiments, it will be recognized that certain modifications, permutations, additions, and sub-combinations thereof of the illustrated embodiments can be made without departing from such principles. It is therefore intended that the following appended claims and claims hereafter introduced are interpreted to include all such modifications, permutations, additions, sub-combinations, and their equivalents as are within their true sprit and scope.

[0192] In view of the many possible embodiments to which the principles of this invention may be applied, it should be recognized that the embodiments described herein with respect to the drawing figures are meant to be illustrative only and should not be taken as limiting the scope of invention. For example, for performance reasons the cryptographic operations may be implemented in hardware, such as on a network card, rather than in software. Therefore, the invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

What is claimed is:

1. One or more computer readable media having computer executable instructions, that when implemented, perform a method for generating a security code for communication by a human user to a device, the method comprising:

a) generating a hash value based on a combination of a modifier and a first data value;

b) modifying the modifier;

c) repeating the operations of generating a hash value and modifying the modifier until a termination condition;

d) selecting a second hash value from the plurality of hash values;

e) generating a first hash value based on a combination of the modifier associated with the selected second hash value and a second data value; and

f) generating a security code for communication by the human user of the first device, the security code containing at least a portion of the first hash value.

2. The one or more computer readable media of claim 1, wherein the termination condition includes determining whether the hash value belongs to a set of acceptable hash values.

3. The one or more computer readable media of claim 2, wherein generating the security code includes combining a hash indicator with the portion of the first hash value, the hash indicator being associated with the set of acceptable hash values.

4. The one or more computer readable media of claim 3, wherein the hash indicator includes a count of bits that have a predetermined value in a selected portion of the second hash value.

5. The one or more computer readable media of claim 1, wherein the first data value is the same as the second data value.

6. The one or more computer readable media of claim 1, wherein the second data value includes the first data value and one or more additional data values.

7. The one or more computer readable media of claim 6, further comprising detecting a collision between the security code and a security code, and incrementing a collision count parameter in one of the additional data values.

8. The one or more computer readable media of claim 1, wherein the termination condition includes a time parameter indicating a time to generate the second hash value.

9. The one or more computer readable media of claim 1, wherein the termination condition includes comparing with a specified threshold a probability of generating another hash value that would be selected within a specified period of time instead of a currently generated hash value.

10. A method comprising:

a) computing a plurality of hash values using a second hash function until a termination condition, an input for each hash value in the plurality of hash values including a data value and a modifier;

b) selecting a second hash value from the plurality of hash values;

c) computing a first hash value using a first hash function, the input to the first hash function including a concatenation of the data value and the modifier of the selected second hash value, the first hash value being different from the second hash value; and

d) setting at least a portion of a security code to a portion of the first hash value, the security code being usable by a human user to authenticate the data value to a device.

11. The method of claim 10, further comprising:

e) a human user retrieving the security code associated with a first device;

f) the human user communicating the security code to a second device;

g) the first device and the second device communicating over a communications network to agree on a data value;

h) the second device using the security code to authenticate the data value.

12. The method of claim 11, further comprising:

i) the second device using the data value to authenticate the first device.

13. The method of claim 11, wherein the data value includes a public key of the first device.

14. The method of claim 11, wherein communicating the security code to the second device includes using an alphanumeric keypad to enter the security code, and communicating the security code to the second device.

15. The method of claim 11, wherein retrieving the security code includes receiving a certificate containing the security code or examining a label associated with the first device.

16. The method of claim 10, wherein the termination condition includes determining whether a hash value belongs to a set of acceptable second hash values.

17. The method of claim 10, further comprising:

e) generating a hash indicator indicating a set of acceptable second hash values;

f) encoding the hash indicator into at least a portion of the security code.

18. The method of claim 10, wherein selecting the second hash value includes selecting a hash value in the plurality of hash values having the greatest number of bits with a predetermined value in a predetermined portion of the hash value.

19. A method of authenticating a data value, the method comprising:

a) obtaining the data value;

b) receiving a security code as human input;

c) computing a first hash value of the data value;

d) comparing at least a portion of the first hash value with at least a portion of the security code;

e) computing a second hash value of the data value; and

f) checking that the second hash value belongs to a set of acceptable hash values.

20. The method of claim **26**, further comprising:

g) decoding a hash indicator from a portion of the security code;

h) determining the set of acceptable hash values from the hash indicator.

* * * * *