Innovation, Sciences et Développement économique Canada
Office de la Propriété Intellectuelle du Canada

Innovation, Science and Economic Development Canada
Canadian Intellectual Property Office

CA 2935435 C 2023/09/19

(11)(21) 2 935 435

(12) BREVET CANADIEN
CANADIAN PATENT

(13) C

(22) Date de dépôt/Filing Date: 2014/03/13

(41) Mise à la disp. pub./Open to Public Insp.: 2014/09/18

(45) Date de délivrance/Issue Date: 2023/09/19

(62) Demande originale/Original Application: 2 905 583

(30) Priorités/Priorities: 2013/03/13 (US61/780,106); 2013/03/22 (US61/804509)

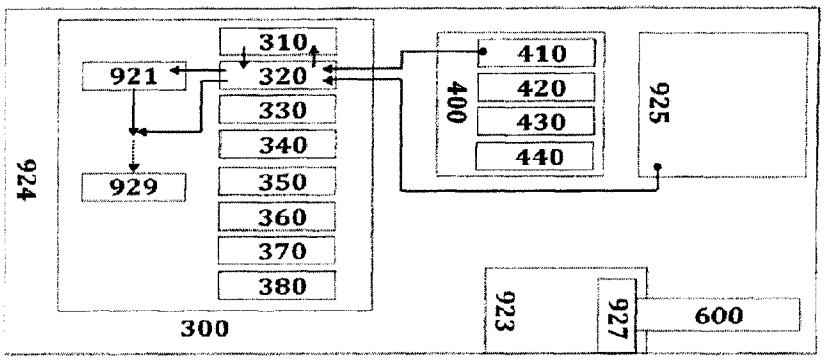
(51) Cl.Int./Int.Cl. *H04L 9/08* (2006.01), *H04L 9/28* (2006.01), *H04L 9/32* (2006.01)

(72) Inventeurs/Inventors:
AMBROZ, ALEXANDER, CA;
PALIR, NEJC, SI

(73) Propriétaire/Owner:
JUMPTO MEDIA INC., CA

(74) Agent: PERRY + CURRIER

(54) Titre : COMMUNICATION SECURISEE EN RESEAU
(54) Title: SECURE NETWORK COMMUNICATION




(57) Abrégé/Abstract:
A client device configured to intercept an outgoing packet. The outgoing packet includes a destination network address. The client device is further configured to use an encryption key to encrypt the outgoing packet to generate an encrypted packet, scatter the encryption key into the encrypted packet according to pattern logic defined by a unique identifier of a routing server, and send the encrypted packet containing the scattered encryption key to the routing server. The routing server is configured to receive the encrypted packet containing the scattered encryption key, extract the encryption key from the encrypted packet using the pattern logic defined by the unique identifier, use the encryption key to decrypt the encrypted packet to obtain the outgoing packet including the destination network address, and send the outgoing packet to the destination network address.

## Abstract of the Disclosure

A client device configured to intercept an outgoing packet. The outgoing packet includes a destination network address. The client device is further configured to use an encryption key to encrypt the outgoing packet to generate an encrypted packet, scatter the encryption key into the encrypted packet according to pattern logic defined by a unique identifier of a routing server, and send the encrypted packet containing the scattered encryption key to the routing server. The routing server is configured to receive the encrypted packet containing the scattered encryption key, extract the encryption key from the encrypted packet using the pattern logic defined by the unique identifier, use the encryption key to decrypt the encrypted packet to obtain the outgoing packet including the destination network address, and send the outgoing packet to the destination network address.

# Secure Network Communication

## Related Applications

[0001] This application claims priority to US provisional applications 61/780,106, filed March 13, 2013, and 61/804,509, filed March 22, 2013.

## Field

[0002] The present invention relates to secure connections and communications between devices and computers on the Internet and in virtual private networks.

## Background

[0003] A variety of methods have been proposed and implemented to provide security and anonymity for device and computer communication over the Internet and in virtual private networks. Conventional communication solutions, including secure virtual private networks, connect a remote device or computer to the target device or computer, while data security is typically tackled using some form of data encryption. These devices and computers in virtual private networks can securely communicate with the exchange of public and private encryption keys or by separately routing (a) the packets of protected data and (b) their encryption keys, from the point of origin - through disparate network paths encompassing multiple devices or computers - to the point of destination. This kind of security is primarily used for corporate virtual private networks that require a broad communication policy thus binding devices and computers to specific networks is preferred.

[0004] These typical virtual private network solutions contain many potential security concerns where there is a need to (a) securely control communication for specific applications and their protocols from devices and computers through foreign networks, (b) ensure that these applications and their protocols can only communicate within the desired foreign networks, and (c) ensure that encryption keys cannot be tracked or traced and that the data cannot be interpreted.

**[0005]** For instance, Larson (U.S. Pat. No. 8,051,181) teaches a technique to establishing a secure communication link between computers of virtual private networks in which one or more data values that vary according to a pseudo-random sequence are inserted into each data packet in order to provide multiple paths to reach the destination. Although Larson provides security enhancements for typical virtual private networks, there remains a need for secure and efficient Internet communication between applications installed on devices.

## Summary

**[0006]** According to one aspect of the present invention, a system for communicating over a network includes a client device configured to intercept an outgoing packet, the outgoing packet including a destination network address. The client device is further configured to use an encryption key to encrypt the outgoing packet to generate an encrypted packet, scatter the encryption key into the encrypted packet according to pattern logic defined by a unique identifier of a routing server, and send the encrypted packet containing the scattered encryption key to the routing server. The system further includes a routing server configured to receive the encrypted packet containing the scattered encryption key, extract the encryption key from the encrypted packet using the pattern logic defined by the unique identifier, use the encryption key to decrypt the encrypted packet to obtain the outgoing packet including the destination network address, and send the outgoing packet to the destination network address.

**[0007]** According to another aspect of the present invention, a client device includes a network interface controller configured to intercept outgoing packets, each outgoing packet including a destination network address. The client device further includes a local proxy server configured to receive the intercepted outgoing packets, control an encryption engine to generate new packets from the outgoing packets, and send the new packets to at least one routing server. The client device further includes the encryption engine configured to use encryption keys to encrypt the intercepted outgoing packets to generate encrypted packets, and scatter the encryption keys into the encrypted packets according to pattern logic defined by at least one unique identifier of the at least one routing server to obtain the new packets.

**[0008]** According to another aspect of the present invention, a routing server includes a network interface controller configured to receive encrypted packets from at least one client device via a network, a protocol agnostic proxy server configured to control an encryption engine to decrypt the received encrypted packets and send packets to at least one destination address via the network, and the encryption engine configured to extract scattered encryption keys from the received encrypted packets using pattern logic defined by a unique identifier of the routing server, and use the encryption keys to decrypt the encrypted packets to obtain the packets to send to the at least one destination address.

**[0009]** According to another aspect of the present invention, a method performed by a client device connected to a network includes intercepting an outgoing packet, the outgoing packet including a destination network address. The method further includes using an encryption key to encrypt the outgoing packet including the destination network address to obtain an encrypted packet, scattering the encryption key into the encrypted packet according to pattern logic defined by a unique identifier of a routing server to obtain a new packet, and sending the new packet to the routing server.

**[0010]** According to another aspect of the present invention, a method performed by a routing server operating in a network includes receiving an encrypted packet via the network from a client device, extracting an encryption key from the received encrypted packet using pattern logic defined by a unique identifier of the routing server, using the encryption key to decrypt the received encrypted packet to obtain an outgoing packet of the client device, the outgoing packet including a destination network address, and sending the outgoing packet to the destination network address.

**[0011]** According to another aspect of the present invention, a method of encrypting a packet includes randomly generating an encryption key from at least user entropy, performing a cryptographic hash on data content of an outgoing packet, using the encryption key to encrypt the outgoing packet to obtain an encrypted packet, performing a cryptographic hash on the encrypted packet, and compressing the encrypted packet.

**[0012]** The method can further include scattering the encryption key into the encrypted packet according to pattern logic defined by a unique identifier of a routing server to obtain a new packet.

**[0013]** The method can be performed independently for each outgoing packet of a plurality of outgoing packets and can use different encryption keys, different cryptographic hashing, and different unique identifiers.

## Brief Description of the Drawings

**[0014]** Embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

**[0015]** FIG. 1 is a block diagram of software components;

**[0016]** FIG. 2 is a block diagram of hardware components;

**[0017]** FIG. 3 is a process diagram of connecting to the server computer;

**[0018]** FIG. 4 is a process diagram of intercepting the outbound network packets and forwarding them to a local proxy server for further processing;

**[0019]** FIG. 5 is a process diagram of generating the random encrypting key and encrypting the network data packet;

**[0020]** FIG. 6 is a process diagram of scattering the encryption key within the encrypted content using the first unique node identification as the pattern for the scattering logic;

**[0021]** FIG. 7 is a process diagram of sending the packet to the server computer and processing it with the protocol agnostic proxy server which decompresses it and forwards it for encryption key rebuilding;

**[0022]** FIG. 8 is a process diagram of rebuilding the encryption content and the encryption key by using the unique node identification as the pattern for the rebuilding logic;

**[0023]** FIG. 9 is a process diagram of sending the decrypted network packet to its target destination and forwarding its reply to the encryption engine for further processing;

**[0024]** FIG. 10 is a process diagram of encrypting the network packet on its return path by using the predefined encryption key;

**[0025]** FIG. 11 is a process diagram of scattering the encryption key within the encrypted content using the second unique node identification as the pattern for the scattering logic;

**[0026]** FIG. 12 is a process diagram of sending the encrypted network packet back to the client destination and forwarding to the local proxy server which decompresses it and forwards it for encryption key rebuilding;

**[0027]** FIG. 13 is a process diagram of rebuilding the encryption content and the encryption key by using the unique node identification as the pattern for the rebuilding logic. The packet is decrypted and its content is returned as a reply to the original network request via the local proxy server; and

**[0028]** FIG. 14 is a process diagram of switching the location of the server computer by changing the global variable.

## Detailed Description

**[0029]** The present invention relates to the field of secure connections and communications between devices and computers on the Internet and in virtual private networks. More particularly, the invention concerns securely connecting and communicating with applications installed on devices and computers to form virtual cloud networks. More particularly, the invention includes application-specific protocol filtering and routing for secure remote communication from within a country of any virtual cloud network.

**[0030]** The present invention can provide secure network communication between applications installed on device without the need to (a) bind devices or computers to substantially the same network, (b) constantly exchange encryption keys, and (c) perform

complex packet routing through disparate paths and through multiple servers to achieve safer communication.

**[0031]** The present invention can also provide increased flexibility to securely and efficiently connect specific applications with specific protocols on dissimilar devices or computers connected on many foreign networks for private, secure and anonymous Internet communication.

**[0032]** The present invention can provide a way of securely and efficiently connecting devices and computers in foreign networks across the Internet, to form secure virtual cloud networks, by employing new security and network routing techniques on the connected devices.

**[0033]** The present invention can provide a secure mechanism for device and computer communication over the Internet that uses (a) a local software firewall that intercepts all outbound traffic on devices and computers, (b) a multi-layer security protection for data packets, including encryption and compression, without exposure of encryption keys, (c) a multi-layer secure proxy server, and (d) a specialized communication routing server. The combined use of these processes in one unified software solution, installed on all connected devices and computers, allows for the creation of secure virtual cloud networks. Once devices and computers are authenticated to a virtual cloud network, all applications installed on connected devices and computers can securely connect, share and communicate with other applications installed on remote devices and computers in disparate networks.

**[0034]** With utilization of aspects of the present invention, cloud-based applications, including browsing and downloading apps, secure file sharing apps, secure e-mail apps, and secure text, voice and video apps, can be securely accessible from any device or computer authenticated to the virtual cloud network; connected devices include computers, tablets, smart phones and smart TVs. Aspects of the invention can provide users secure and anonymous access to their applications and the associated data from any device or computer connected anywhere on the Internet.

**[0035]** Referring now to the invention in more detail, in Fig. 1 and Fig. 2 there are shown plurality of software and hardware components, respectively, which can be used to implement embodiments of the invention:

200 – Processor

202 – Input device

220 – Processor

222 – Memory

300 – Encryption Engine

310 – Generate Random Encryption Key

320 – Encrypt Data

330 – Decrypt Data

340 – Generate Cryptographic Hash

350 – Build Encryption Key

360 – Scatter Encryption Key

370 – Compress Data

380 – Decompress Data

400 – Local Proxy Server

410 – Construct and Send Network Packet

420 – Deconstruct Network Packet

430 – Node Connector

440 – Node Switcher

500 – Secure Cloud Communicator

510 – List of Available Nodes

600 – Local Firewall with Packet Filtering Capabilities

700 – Server Computer

710 – Encryption Engine

711 – Encrypt Data

712 – Decrypt Data

713 – Build Encryption Key

714 – Scatter Encryption Key

715 – Compress Data

716 – Decompress Data

720 – Protocol Agnostic Proxy Server

921 – Encryption Key

922 – Unique Server Node Identification

923 – Network Interface Controller

924 – Device

925 – Network Packet

926 – Packet Destination

927 – Driver

928 – Public Internet

929 – Encrypted Packet

930 – Node IP Variable

931 – Unencrypted Packet

**[0036]** With reference to Fig. 2, the client device 924 can include a processor (e.g., CPU) 200, an input device 202, a network interface controller 923, and an intercepting driver 923 resident in memory (not shown). The routing server 700 can include a processor (e.g., CPU) 220, random-access memory (RAM) 222, and a network interface controller 923.

**[0037]** Referring to Fig. 1 and Fig. 2, embodiments of the invention include a technique for securely connecting devices and computers on the Internet to form secure virtual cloud networks. Initially, a firewall with packet filtering capabilities **600**, containing the intercepting driver, monitors and intercepts all data packets on the network interface controller **923**; it permits users to intercept all network traffic on devices and computers. Data packets are protected with multi-layer security, including encryption and compression. The protected packets contain the hidden and scattered encryption key sequence. They are securely routed through multi-layer proxy servers **400**, which include remote server computers **700** connected to the Internet. Server computers **700** receive and read the protected packets, while the encryption key is not exposed.

**[0038]** In further detail, still referring to Fig. 1 and Fig. 2, the intercepting driver intercepts the outgoing network packets and forwards them to a local proxy server **400** that processes and encrypts the packets. The intercepting driver holds the connection open while it creates a new packet with the encrypted content of the halted packet and sends it out through the local proxy server **400** to the server computer **700**. Server computer **700** decrypts the received content, gathers the data from the original request and returns it to the client device or computer that requested it in the encrypted form. The packet with the open connection remaining open is injected with the response received from the proxy server. Each individual packet is modified, encrypted and compressed. The modified packets contain the hidden and scattered encryption key sequence.

**[0039]** Referring now to Fig. 3, the process of connecting to a node **430** is triggered when the local proxy server process receives the command "connect node" along with the required parameter "unique node identification". In the first step, the process of connecting to a node **430** queries the list of available nodes **510**. The process of fetching and returning the list of all available server nodes **510** on the secure network is triggered by calling the secure cloud communicator process with the "list nodes" command. Node connector **430** communicates with the network server and retrieves the list of all available nodes on the network along with the "unique node identification" numbers and the array of inbound IP addresses available for

connection. The second step returns the constructed list as the result of the process. The local secure cloud communicator **500** is a software component or script, which is installed and running on the user's device. It periodically or manually, on demand, connects to a network server and sets or gets data depending on the command issued. The secure cloud communicator **500** is used to authenticate local user on a device and fetch the list of available nodes. The process of fetching and returning the list of all available server nodes **510** on the secure network is triggered by calling the secure cloud communicator process **500** with the "list nodes" command. Node connector **430** communicates with the network server and retrieves the list of all available nodes on the network along with the "unique node identification" numbers and the array of inbound IP addresses available for connection. The second step returns the constructed list as the result of the process. The list of available nodes process **510** returns an array of available IP addresses to connect to and stores it to a temporary variable "A". A random IP address is chosen from the array in variable "A" and set as the global node IP variable **930**. The global node IP variable **930** is a globally set variable in the software which contains the IP address of the node server that is being used for connection routing. The connection to the node is established.

**[0040]** Referring now to Fig. 4, the network packet **925** is dispatched by the local network and gets intercepted by the local firewall with packet filtering capabilities **600**. The local firewall with packet filtering capabilities **600** requires the installation of a special packet interception driver **927** on the network interface controller **923** in order to intercept the packets and forward them to the local proxy server **400** for further processing. A network interface controller **923** is a computer hardware component that connects a computer to a computer network. The interception driver **927** is a computer program that operates and controls the **927** network interface controller and is customized to communicate with the local proxy server **400** on the application layer. The local proxy server **400** is a software component or script, which is installed and running on the user's device. It listens for commands from processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to route traffic of intercepted outgoing network packets, send and receive network packets, switch the connecting nodes, detect the best

location of the node to route data, and randomize the nodes and/or outgoing IP addresses of connecting nodes. The proxy server connects and forwards traffic to the server node IP address defined in the global node IP variable described in **930**. The global node IP variable **930** is a globally set variable in the software, which contains the IP address of the node server that is being used for connection routing.

**[0041]** Referring now to the embodiment of Fig. 5, the process of constructing a network packet **410** is triggered when the local proxy server receives the command "construct" along with the required parameter "packet A" and "destination IP address". The result of constructing a network packet **410** is the creation of a new network packet, "packet B". The "packet B" headers are set to route to the "destination IP address". The "packet A" is kept alive without leaving the device until a response is received and injected in it. The entire packet, which is passed in the parameter "packet A", is encrypted using the encryption engine and the process described in **320** and is stored in the temporary variable "A". The process of encrypting data **320** is triggered when the encryption engine **300** process receives the action command "encrypt data" along with the required parameter "data" and the "unique target identification". The encryption engine **300** is a software component or script, which is installed and is running on users' devices. It listens for commands from the processes described herein, drivers discussed herein, as well as other software, using inter-process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption as well as generate cryptographic hashes. The data parameter is stored in the temporary variable "A" and emptied after successful completion of data encryption. The unique target destination is stored in the temporary variable "B" and emptied after successful completion of data encryption.

**[0042]** In the first step, construct and send network packet process **410** internally communicates with the process **310** and stores the generated encryption key into variable "C". The process of creating a random encryption key **310** is triggered if the encryption engine process receives the action command "generate key". The encryption key is generated by using

Unix Epoch time, a 32-digit random number and mouse entropy. The values are stored and combined into a temporary variable "Z". Variable "Z" gets cryptographically hashed by using the internal process **340**. The process of generating a cryptographic hash **340** is triggered when the encryption engine process **300** receives the command "hash" along with the required parameter "value". The value parameter is stored in a temporary variable "Z". The value of variable "Z" is emptied and deleted from memory after the successful completion of this process. The encryption engine **300** uses one of the irreversible cryptographic hashing methods defined by the global system (SHA-2, SHA-3) to hash the value of variable "Z" and return it as the result of this process. The values of variables are emptied and deleted from memory. The cryptographically hashed value gets returned as the final result.

**[0043]** In the second step, construct and send network packet process **410** encrypts the variable "A" with the encryption key from variable "C" using the system defined encryption algorithm (AES, RSA, Serpent, Two-fish). The encrypted data is returned and stored into variable "D".

**[0044]** In the third step, construct and send network packet process **410** internally communicates with the process **360** and scatters the encryption key within the variable "D" on a predefined pattern, which depends on the unique destination identification. The value of variable "D" is updated.

**[0045]** In the fourth step, construct and send network packet process **410** internally communicates with the process **370** which compresses the scattered value of variable "D" to a smaller value. The value of variable "D" is updated. The compressed variable "D" is returned as the result of the encryption process. The variables are emptied and deleted from system memory. This completes the data encryption process. The encrypted variable "A" is used as the data and/or content parameter of the newly constructed "packet B". The result of this process is the dispatch of the creation of the variables required in the next process of scattering the encryption key **360**.

**[0046]** Referring now to Fig. 6, the process of scattering the encryption **360** key is triggered when the encryption engine process receives the command "scatter key" along with the required parameters "encrypted data" **929**, "encryption key" **921** and "unique target identification" **922**. The encryption key **921** is used to encrypt and decrypt the binary value of the network packet. The encryption key is generated in the **310** process. The unique server node identification **922** is a series of characters, which uniquely identify a server node. The unique server node identification **922** is used as the logic pattern for scattering and building the encryption key within the encryption engine. Scatter encryption key process **360** divides the encryption key and scatters it within encrypted data using the logic pattern from the "unique target identification" parameter. In some examples, the unique server node identification (unique identifier) is only known to the server that it identifies, and therefore only that server can extract the encryption key from the packet. The method, which defines the inter-operation of the unique target identification related to the scattering of the encryption key, is hardcoded into the **300** encryption engine. The encryption engine **300** is a software component or script, which is installed and running on the user's device. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption as well as generate cryptographic hashes. The hardcoded method must be substantially the same within the local encryption engine **300** and within the encryption engine located on the server computer **700**, in order for "encrypt data" and "decrypt data" processes to work and correctly compile and decompile information. In order to create an example, we will use the value of "SCBA1342" as the unique target identification **922**, "XY9876WZ" as the generated encryption key **921** and "C66AB657BF319B38284492AD2E21514E" as the encrypted content **929.** We will use a pattern logic, which scatters the encryption key within the encrypted content, by using numbers and characters from the unique target identification as position marks for each corresponding character in the encryption key **921**. The numbers from 0 – 9 have the value range of 0 – 9. The letters in the English alphabet have the value range of 10 – 35. By knowing the pattern of

scattering the encryption key, we translate the unique target identification value "SCBA1342" to position values. The letter "S" is the first character in the unique target identification, and if converted, has the position value of 28, which means that the first character from the encryption key 921 is injected before the 28th character in the encrypted data 929. The exact position translation for the scattering pattern "SCBA1342" is "28 – 12 – 11 – 10 – 1 – 3 – 4 – 2". After completing the scattering of the encryption key **921**, we end up with the result of "7Z6WC66AB89Y657BF319B382844X92AD2E21514E", which replaces variable 929 as the new encrypted packet. The length of the encryption key lengthens the encrypted data value. The lengthened encrypted data containing the scattered encryption key is returned as the result of process **360**. Note that this pattern example is merely an illustrative example used only to explain the concept of encryption key scattering. Implementation examples vary in the method of implementation, length of encryption keys, length of unique target identification and complexity of the pattern structure. The scattered encryption gets returned to the encrypt data process **320**, which compresses the data using the compression process **370**, for increased network performance, and returns it to the **410** process in the local proxy server **400**. The process of compressing data **370** is triggered when the encryption engine process receives the command "compress data" along with the required parameter "encrypted data". Scatter encryption key process **360** compresses the parameter "encrypted data" using the system defined compression algorithm (for example, LZ4, ZIP, RAR, TAR, or TAR.GZ). The compressed data is returned as the result of this process. The packet in process **410** is flagged with ignore bits that informs the local firewall with packet filtering capabilities **600** to ignore the filtering process during traffic processing. The result of this process is the dispatch of packets from the system to the server computer **700**, identified with the unique target identification. The server computer **700** is a hardware server running on CENTOS, for example, with the encryption engine and with the protocol agnostic proxy server **720** installed on it.

**[0047]** Referring now to Fig. 7, the encrypted packet **929** is sent to the server computer **700** via the public internet **928**. The packet is received and processed by the protocol agnostic proxy server **720**. The protocol agnostic proxy server **720** runs on the server computer and accepts inbound network connections on the specific port, in the range of selected IP

addresses. It is capable of processing network protocols regardless of their type (for example, UDP, TCP/IP). The packet is forwarded to the data decryption process **712** within the encryption engine **710**. The encryption engine **710** is a software component or script, which is installed and running on the server computer. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption as well as generate cryptographic hashes. The process of decrypting data **712** is triggered when the encryption engine **710** process receives the action command "decrypt data" along with the required parameter "encrypted data" and "unique target identification". The encrypted data parameter is stored in the temporary variable "A" and emptied after successful completion of data decryption. The unique target destination is stored in the temporary variable "B" and emptied after successful completion of data encryption.

**[0048]** In the first step, the encrypted packet process **929** internally communicates with the process **716**, which decompresses the received data from variable "B". The process of decompressing data **716** is triggered when the encryption engine **710** process receives the command "decompress data", along with the required parameter "compressed data". The encrypted packet process **929** decompresses the parameter "compressed data" using the system defined compression algorithm (for example, LZ4, ZIP, RAR, TAR, or TAR.GZ). The decompressed data is returned as the result of this process. The value of variable "B" is updated with the new decompressed value. In the second step, the encrypted packet process **929** internally communicates with the process **713**, which builds the scattered encryption key within the variable "B" on a predefined pattern, which is dependent on the unique destination identification.

**[0049]** Referring now to Fig. 8, the process of building the encryption key **713** is triggered when the encryption engine process receives the command "build key", along with the required parameters "encrypted data" **929** and "unique target identification" **922**. Build the encryption key process **713** extracts the encryption key and the encrypted content from the encrypted

data parameter using the logic pattern from the "unique target identification" parameter. The method, which defines the inter-operation of the unique target identification related to the scattering of the encryption key, is hardcoded into the **710** encryption engine. The encryption engine **710** is a software component or script, which is installed and running on a server computer. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption as well as generate cryptographic hashes. The hardcoded method must be substantially the same within the local encryption engine **300** and within the encryption engine located on the server computer **700**, in order for "encrypt data" and "decrypt data" processes to work and correctly compile and decompile information. In order to create an example we use the value of "SCBA1342" as the unique target identification **922**, and "7Z6WC66AB89Y657BF319B382844X92AD2E21514E" as the encrypted content **929**, which contains a secret scattered encryption key**.** We use a pattern logic that rebuilds and divides the scattered encryption key and encrypted content from the encrypted content by using numbers and characters from the unique target identification, as position marks for each corresponding character for the encryption key **921**. The numbers from 0 – 9 have the value range of 0 – 9. The letters in the English alphabet have the value range of 10 – 35. By knowing the pattern of scattering the encryption key, we can translate the unique target identification value "SCBA1342" to position values. The letter "S" is the first character in the unique target identification, and when converted, it has the position value of 28, which means that the first character to form the encryption key **921** is the $28^{th}$ character in the encrypted data **929**. The exact position translation for the scattering pattern "SCBA1342" is "28 – 12 – 11 – 10 – 1 – 3 – 4 – 2". After completing the scattering of our encryption key within the encrypted content, we get two separated variables as the result of this process. We get the encryption key **921** value "**XY9876WZ**" and the encrypted content **929** value "C66AB657BF319B38284492AD2E21514E", as the new data to be decrypted. The extracted encryption key and the extracted content are returned as the result of this process. They are forwarded back to data decryption process **712**

for data decryption. The process of decrypting data **712** is triggered when the encryption engine **710** process receives the action command "decrypt data", along with the required parameter "encrypted data" and "encryption key". The encryption key is stored into a temporary variable "C" and a session permanent variable that is used during encryption during the creation of the reply network package. The encrypted content is stored into a temporary variable "D". Build the encryption key process **713** decrypts the variable "D" with the built encryption key from variable "C" using the system defined encryption algorithm (AES, RSA, Serpent, Two-fish). The decrypted data is returned as the result of this process. The variables are emptied and deleted from system memory. The decryption of data is complete. The unencrypted packet **931** is forwarded to the protocol agnostic proxy server **720** for further processing. The process of the protocol agnostic proxy server **720** runs on the server computer and accepts inbound network connections on the specific port in the range of selected IP addresses. The proxy server processes and decrypts the received encrypted requests and makes a new request to the originally intended destination. The destination is read from the header of the unencrypted network packet **931** and sent out via the network controller interface **923.**

**[0050]** Referring now to Fig. 9, the unencrypted network packet **931** is sent by the server computer **700** to the packet destination server **926**. The **926** packet destination server is a computer server, which was addressed in the originally intercepted network packet in FIG. 4. The server computer **700** is a hardware server running on CENTOS, for example, with the encryption engine and with the protocol agnostic proxy server **720** installed on it. The request from the unencrypted network packet **931** gets processed by the packet destination server **926**, which returns an unencrypted packet back to the protocol agnostic proxy server **720**, as the processed reply to the sent request. The process of the protocol agnostic proxy server **720** runs on the server computer and accepts inbound network connections on the specific port in the range of selected IP addresses. Once the packet is received by the protocol agnostic proxy server **720**, it gets forwarded to the data encryption **711** process within the encryption engine **710** and prepares and encrypts the packet for a structured reply to the connected device, which triggered the original request.

**[0051]** Referring now to Fig. 10, the unencrypted packet **931** gets forwarded to the encryption engine **710** for data encryption within the data encryption process **711.** The encryption engine **710** is a software component or script, which is installed and running on server computer. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption and to generate cryptographic hashes. The process of encrypting data **711** is triggered when the encryption engine **710** process receives the action command "encrypt data" along with the required parameter "data". The data parameter is stored in the temporary variable "A" and emptied after successful completion of data encryption. The encryption key **921**, used to encrypt the data, is pulled from the session permanent variable that was populated and set as described in FIG. 8. Unencrypted packet process **931** encrypts the variable "A" with the encryption key **921**, using the system defined encryption algorithm (AES, RSA, Serpent, Two-fish). The encrypted data is returned and stored into variable "D". The variables are emptied and deleted from system memory. This completes the data encryption process. An encrypted network packet **929** is returned as the result of this process.

**[0052]** Referring now to Fig. 11, the process of scattering the encryption key **714** is triggered when the encryption engine process receives the command "scatter key", along with the required parameters "encrypted data" **929**, the "encryption key" **921** and the "unique target identification" **922**. The encryption key **921** is used to encrypt and decrypt the binary value of the network packet. The unique server node identification **922** is a series of characters, which uniquely identify a server node. The unique server node identification **922** is used as the logic pattern for scattering and building the encryption key within the encryption engine. Scatter encryption key process **714** divides the encryption key and scatters it within encrypted data using the logic pattern from the "unique target identification" parameter. In some examples, the unique server node identification (unique identifier) is only known to the server that it identifies, and therefore only that server can extract the encryption key from the packet. The method, which defines the inter-operation of the unique target identification

related to the scattering of the encryption key, is hardcoded into the **710** encryption engine. The encryption engine **710** is a software component or script, which is installed and is running on server computer. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption as well as generate cryptographic hashes. The hardcoded method must be substantially the same within the local encryption engine **300** and within the encryption engine located on the server computer **700**, in order for "encrypt data" and "decrypt data" processes to work and correctly compile and decompile information. In order to create an example we use the value of "SCBA1342" as the unique target identification **922**, "XY9876WZ" as the previously stored encryption key **921** and "756EF53A416236BD2D3BA331E3D367C5" as the encrypted content **929.** We use a pattern logic, which scatters the encryption key within the encrypted content, by using numbers and characters from the unique target identification as position marks for each corresponding character in the encryption key **921**. The numbers from 0 – 9 have the value range of 0 – 9. The letters in the English alphabet have the value range of 10 – 35. By knowing the pattern of scattering the encryption key we can translate the unique target identification value "SCBA1342" to position values. The letter "S" is the first character in the unique target identification, and when converted, it has the position value of 28, which means that the first character from the encryption key **921** will is injected before the $28^{th}$ character in the encrypted data **929**. The exact position translation for the scattering pattern "SCBA1342" is "28 – 12 – 11 – 10 – 1 – 3 – 4 – 2". After completing the scattering of our encryption key within the encrypted content, we get the result of "**7Z6W**756EF**89Y**53A416236BD2D3B**X**A331E3D367C5", which replaces variable **929** as the new encrypted packet. The length of the encryption key lengthens the encrypted data value. The lengthened encrypted data containing the scattered encryption key is returned as the result of process **714**. Note that this pattern example is merely an illustrative example used only to explain the concept of encryption key scattering. Implementation examples vary in the method of implementation, length of encryption keys, length of unique target identification and

complexity of the pattern structure. The scattered encryption gets returned to the encrypt data process **711**, which compresses the data using the compression process **715** for increased network performance and returns it to the protocol agnostic proxy server **710** process for further processing. The process of compressing data **715** is triggered when the encryption engine process receives the command "compress data" along with the required parameter "encrypted data". Scatter encryption key process **714** compresses the parameter "encrypted data" using the system defined compression algorithm (for example, LZ4, ZIP, RAR, TAR, TAR.GZ). The compressed data is returned as the result of this process. The result of this process is the dispatch of the packet from the server computer **700** back to the requesting client device via the network interface controller **923**. The server computer **700** is a hardware server running on CENTOS, for example, with the encryption engine and the protocol agnostic proxy server **720** installed on it.

**[0053]** Referring now to Fig. 12, the local proxy server **400** receives the encrypted packet **929** from the server computer **700** and triggers the process of deconstructing a network packet **420.** The process of deconstructing a network packet **420** is triggered on the response from sending the packet in process **410.** The received packets' data and content are stored in a variable "A" and decrypted by the encryption engine with the decrypt data process **330**. The decrypted content of the received packet is the response, which gets injected to the "packet A" in process **410** as the response. The local proxy server **400** is a software component or script, which is installed and running on the user's device. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to route traffic of intercepted outgoing network packets, send and receive network packets, switch the connecting nodes, detect the best location of the node to route data, and to randomize the nodes and/or outgoing IP addresses of connecting nodes. The proxy server connects and forwards traffic to the server node IP address defined in the global node IP variable, as described in **930**. The global node IP variable **930** is a globally set variable in the software, which contains the IP address of the node server that is being used for connection routing. The server computer **700** is a hardware server running on CENTOS, for example, with the encryption engine and protocol agnostic proxy server **720** installed on it. The

process of decrypting data **330** is triggered when the encryption engine process **300** receives the action command "decrypt data", along with the required parameter "encrypted data" and the "unique target identification". The encrypted data parameter is stored in the temporary variable "A" and emptied after successful completion of data decryption. The unique target destination is stored in the temporary variable "B" and emptied after successful completion of data encryption.

**[0054]** In the first step, local proxy server **400** internally communicates with the process **380** which decompresses the received data from variable "B". The value of variable "B" is updated with the new decompressed value. The process of decompressing data **380** is triggered if the encryption engine process receives the command "decompress data" along with the required parameter "compressed data". Local proxy server process **400** decompresses the parameter "compressed data" using the system defined compression algorithm (for example, LZ4, ZIP, RAR, TAR, TAR.GZ). The decompressed data is returned as the result of this process.

**[0055]** In the second step, local proxy server **400** internally communicates with the process **350**, which builds the scattered encryption key within the variable "B" on a predefined pattern, which is dependent on the unique destination identification.

**[0056]** Referring now to Fig. 13, the process of building the encryption key **350** and rebuilding the encrypted content is triggered when the encryption engine process **300** receives the command "build key", along with the required parameters "encrypted data" and the "unique target identification". The encryption engine **300** is a software component or script, which is installed and running on users' device. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to encrypt the content of outgoing packets, decrypt the data of incoming packets, generate the random encryption key, scatter the encryption key and rebuild it for dynamic encryption as well as generate cryptographic hashes. Build the encryption key process **350** extracts the encryption key and the encrypted content from the encrypted data parameter using the logic pattern from the "unique target identification" parameter. The method, which defines the inter-operation of the unique target identification related to the

scattering of the encryption key, is hardcoded into the encryption engine **300**. The hardcoded method must be substantially the same within the local encryption engine **300** and within the encryption engine located on the server computer **700**, in order for "encrypt data" and "decrypt data" processes to work and correctly compile and decompile information. In order to create an example we use the value of "SCBA1342" as the unique target identification **922**, and "7Z6W756EF89Y53A416236BD2D3BXA331E3D367C5" as the encrypted content **929**, which contains a secret scattered encryption key. We use a pattern logic that rebuilds and divides the scattered encryption key and encrypted content from the encrypted content, by using numbers and characters, from the unique target identification as position marks for each corresponding character for the encryption key **921**. The numbers from 0 – 9 have the value range of 0 – 9. The letters in the English alphabet have the value range of 10 – 35. By knowing the pattern of scattering the encryption key we can translate the unique target identification value "SCBA1342" to position values. The letter "S" is the first character in the unique target identification and if converted has the position value of 28, which means that the first character to form the encryption key **921** is the 28$^{th}$ character in the encrypted data **929**. The exact position translation for the scattering pattern "SCBA1342" is "28 – 12 – 11 – 10 – 1 – 3 – 4 – 2". After completing the scattering of our encryption key within the encrypted content we get two separated variables as the result of this process. We get the encryption key **921** value "**XY9876WZ**" and the encrypted content **929** value "756EF53A416236BD2D3BA331E3D367C5", as the new data to be decrypted by the decryption engine **330**. The process **350** returns the two variables: the dynamic encryption key **921** and the encrypted content **929**. The encryption key is stored into a temporary variable "C" while the encrypted content is stored into a temporary variable "D". The third step uses the decryption process **300** to decrypt the variable "D" with the built encryption key from variable "C" using the system defined encryption algorithm (for example, AES, RSA, Serpent, Two-fish). The decrypted data is returned as the result of this process. The variables are emptied and deleted from system memory. This completes data decryption. The decrypted packet **931** is returned and injected to the network packet **925** as the network packet containing the reply to its original network request.

**[0057]** Referring now to Fig. 14, the node switcher process **440** address without breaking the node connection and temporarily disabling and compromising network security is triggered when the local proxy server **400** process receives the command "switch node", along with the required parameter "unique node identification". The local proxy server **400** is a software component or script, which is installed and running on the user's device. It listens for commands from the processes described herein, drivers discussed herein, as well as other software using inter-process communication. It is configured to route traffic of intercepted outgoing network packets, send and receive network packets, switch the connecting nodes, detect the best location of the node to route data, and to randomize the nodes and outgoing IP addresses of connecting nodes. The proxy server connects and forwards traffic to the server node IP address defined in the global node IP variable described in **930**. The global node IP variable **930** is a globally set variable in the software, which contains the IP address of the node server that is being used for connection routing. Node switcher process **440** queries the list of available nodes **510**. The list of available nodes process **510** returns an array of available IP addresses to connect to and stores it to a temporary variable "A". A random IP address is chosen from the array in variable "A" and set as the global node IP variable **510**. The connection to the node is established automatically on the next request triggered by an outgoing network packet.

**[0058]** As apparent from the above, several client devices can securely communicate with each other through the routing server, where each client's packets are encrypted differently such that only the intermediating routing server can decrypt them. In such examples, outgoing packets from each client device identify the other client device by destination address.

**[0059]** *The term "server" as used herein is not limiting and denotes any electronic device* capable of performing the functions described herein, such as servers, computers, mobile devices, smartphones, and the like.

**[0060]** While the foregoing provides certain non-limiting example embodiments, it should be understood that combinations, subsets, and variations of the foregoing are contemplated. The monopoly sought is defined by the claims.

**What is claimed is:**

1. A method of encrypting a packet, the method comprising:

randomly generating an encryption key from at least user entropy;

performing a cryptographic hash on data content of an outgoing packet;

using the encryption key to encrypt the outgoing packet to obtain an encrypted packet;

performing a cryptographic hash on the encrypted packet; and

compressing the encrypted packet.

2. The method of claim 1, further comprising scattering the encryption key into the encrypted packet according to pattern logic defined by a unique identifier of a routing server to obtain a new packet.

3. The method of claim 2, wherein the method is performed independently for each outgoing packet of a plurality of outgoing packets.

4. The method of claim 2, wherein the method is performed independently for each outgoing packet of a plurality of outgoing packets using different encryption keys, different cryptographic hashing, and different unique identifiers.



Date Reçue/Date Received 2023-01-27

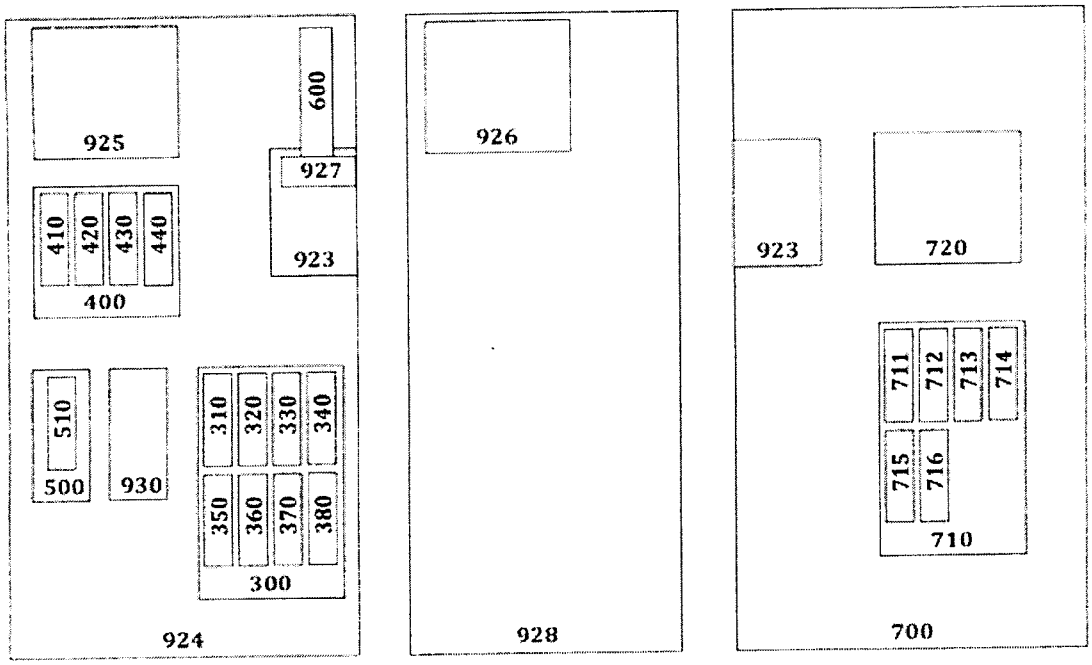



FIGURE 1

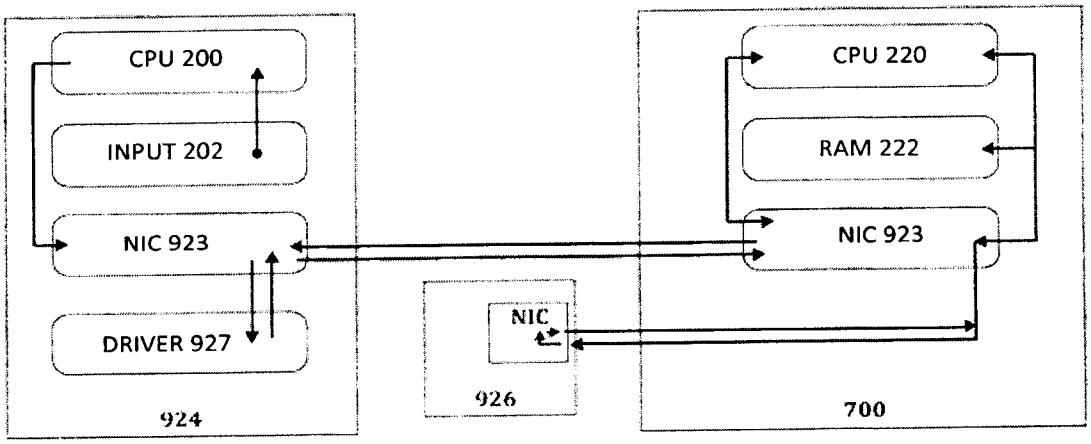



FIGURE 2

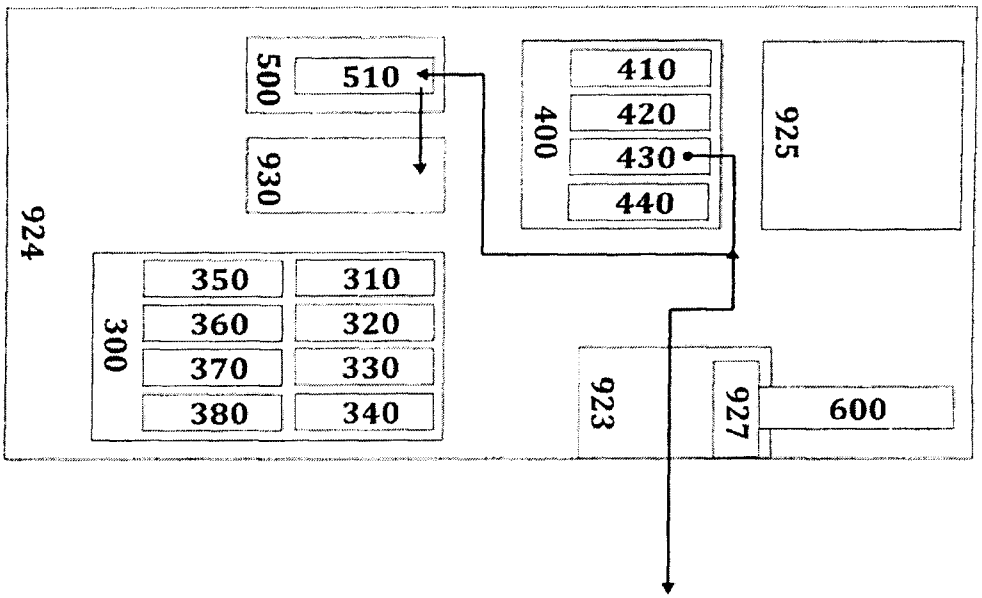



FIGURE 3

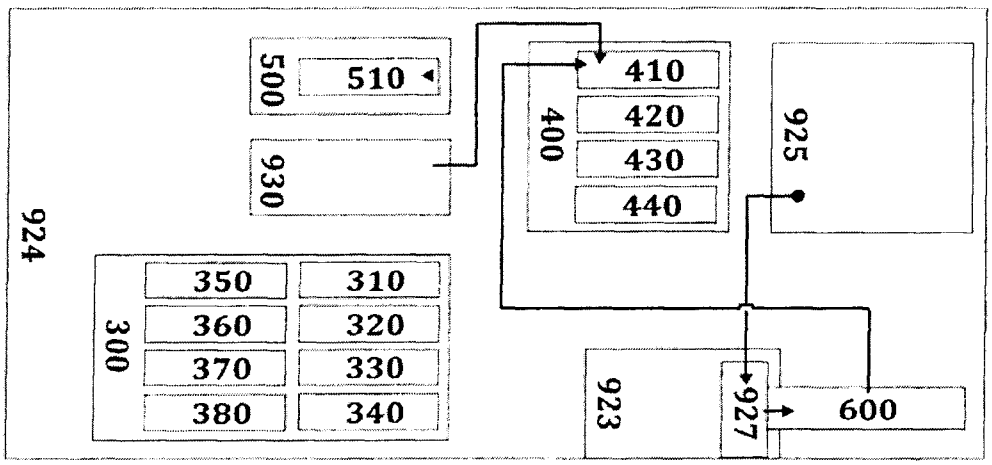



FIGURE 4

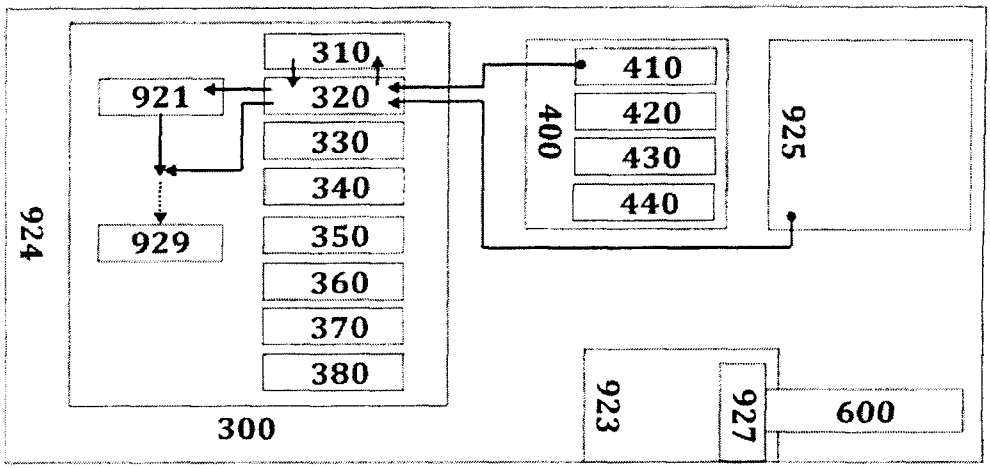



FIGURE 5

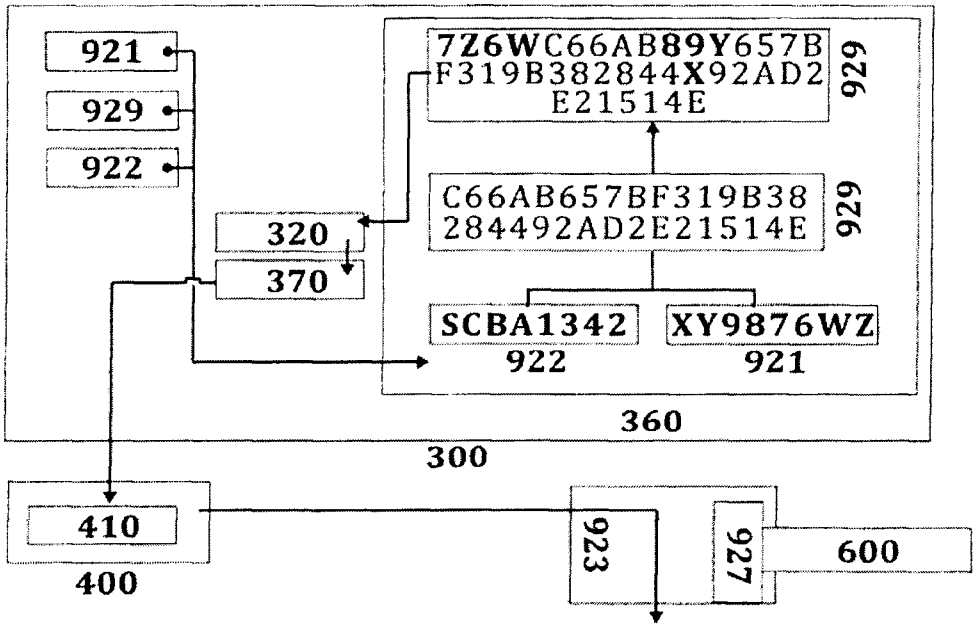



FIGURE 6

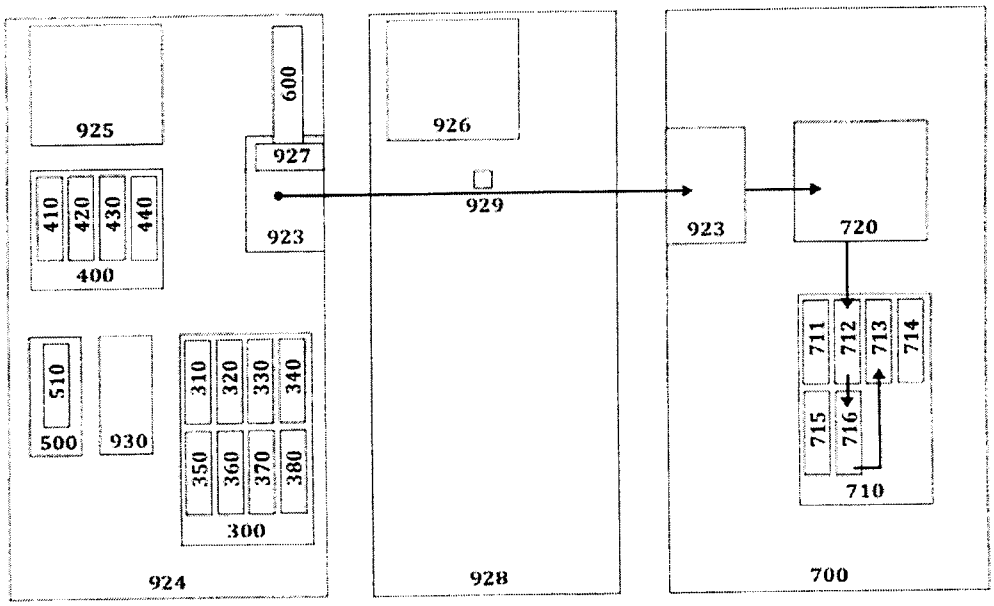



FIGURE 7

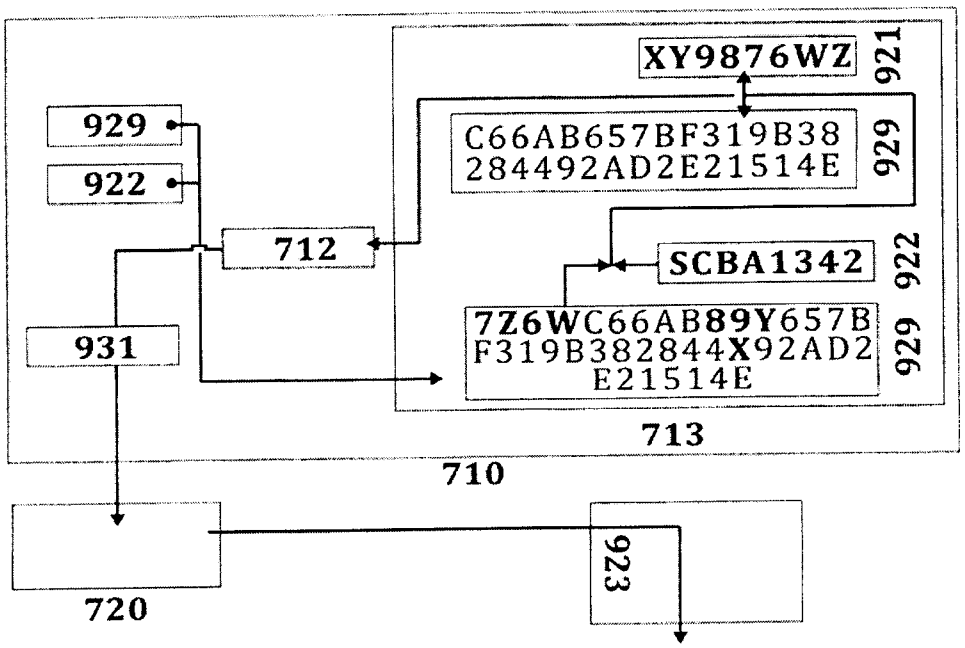



FIGURE 8

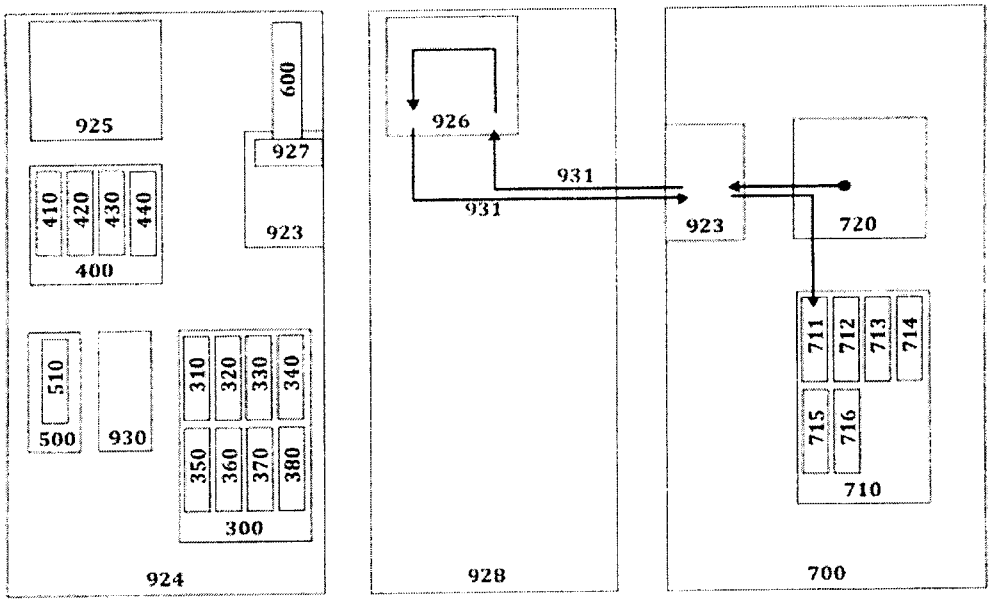



FIGURE 9

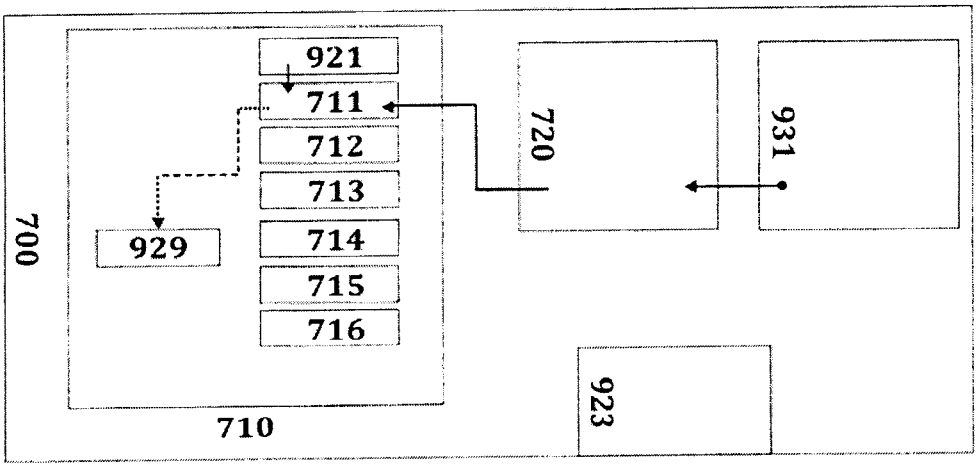



FIGURE 10

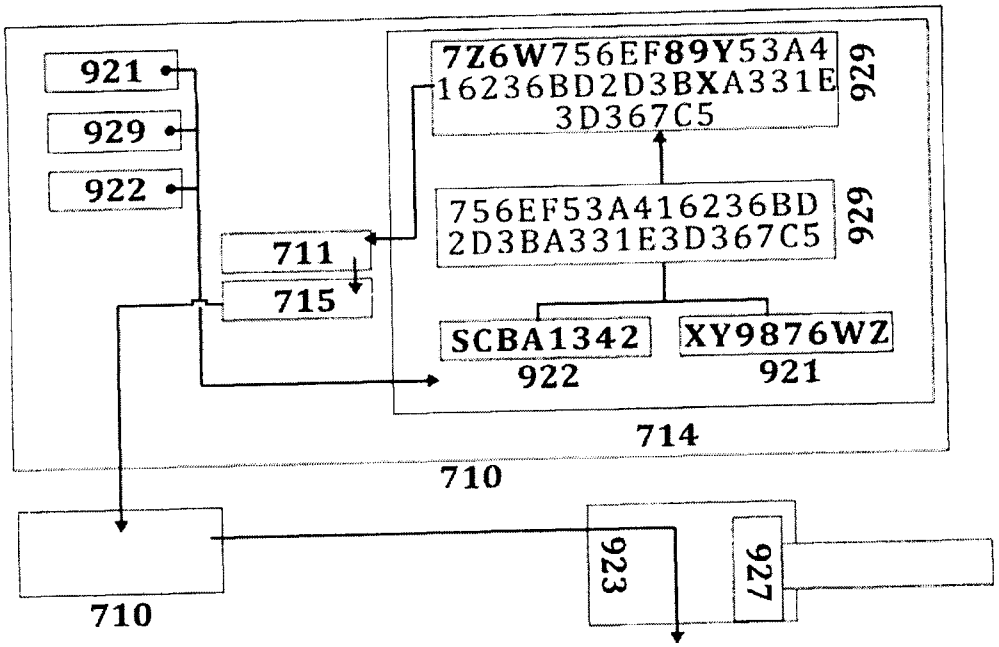



FIGURE 11

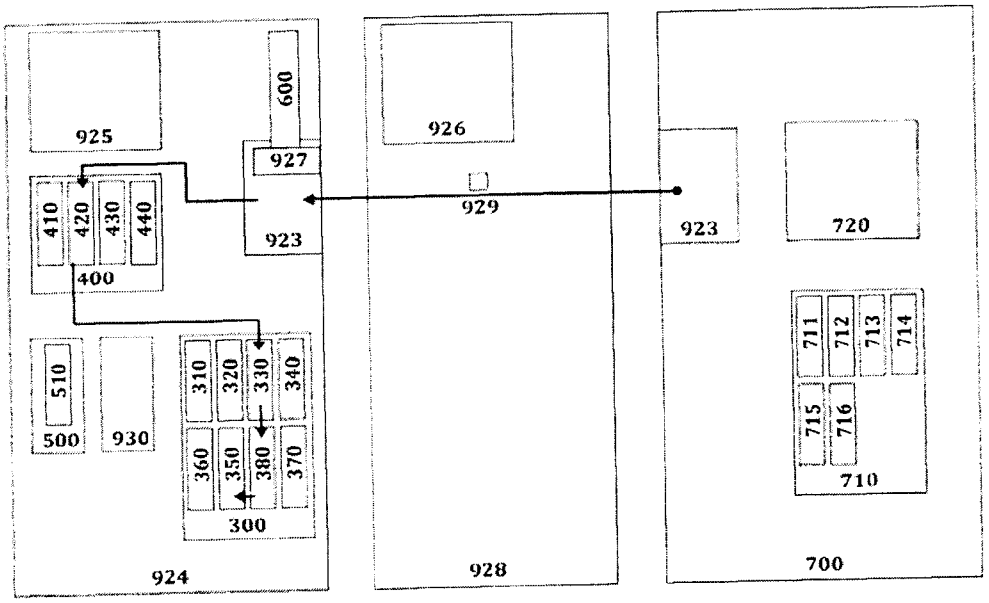



FIGURE 12

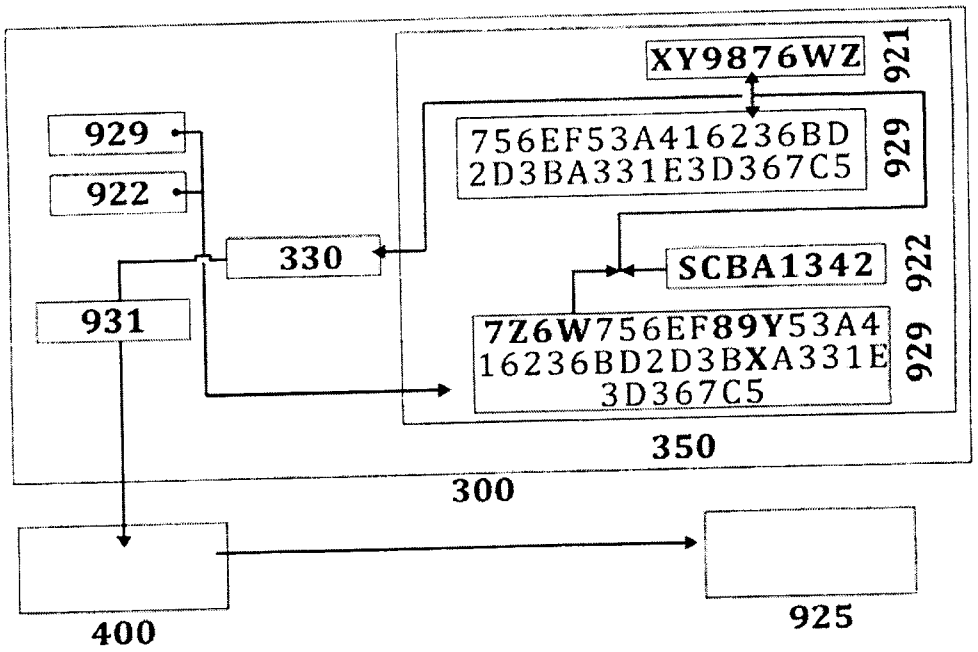



FIGURE 13

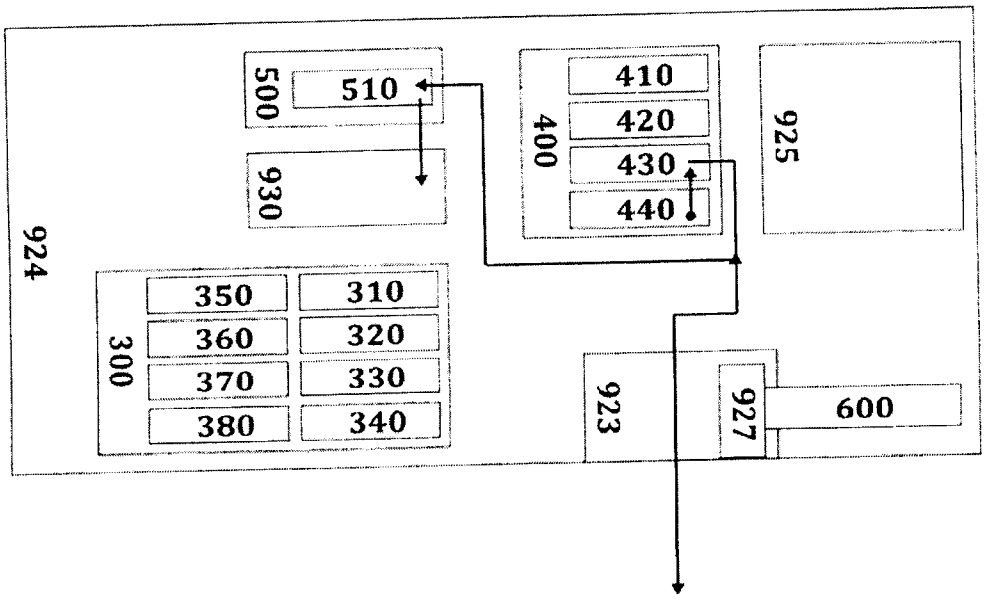
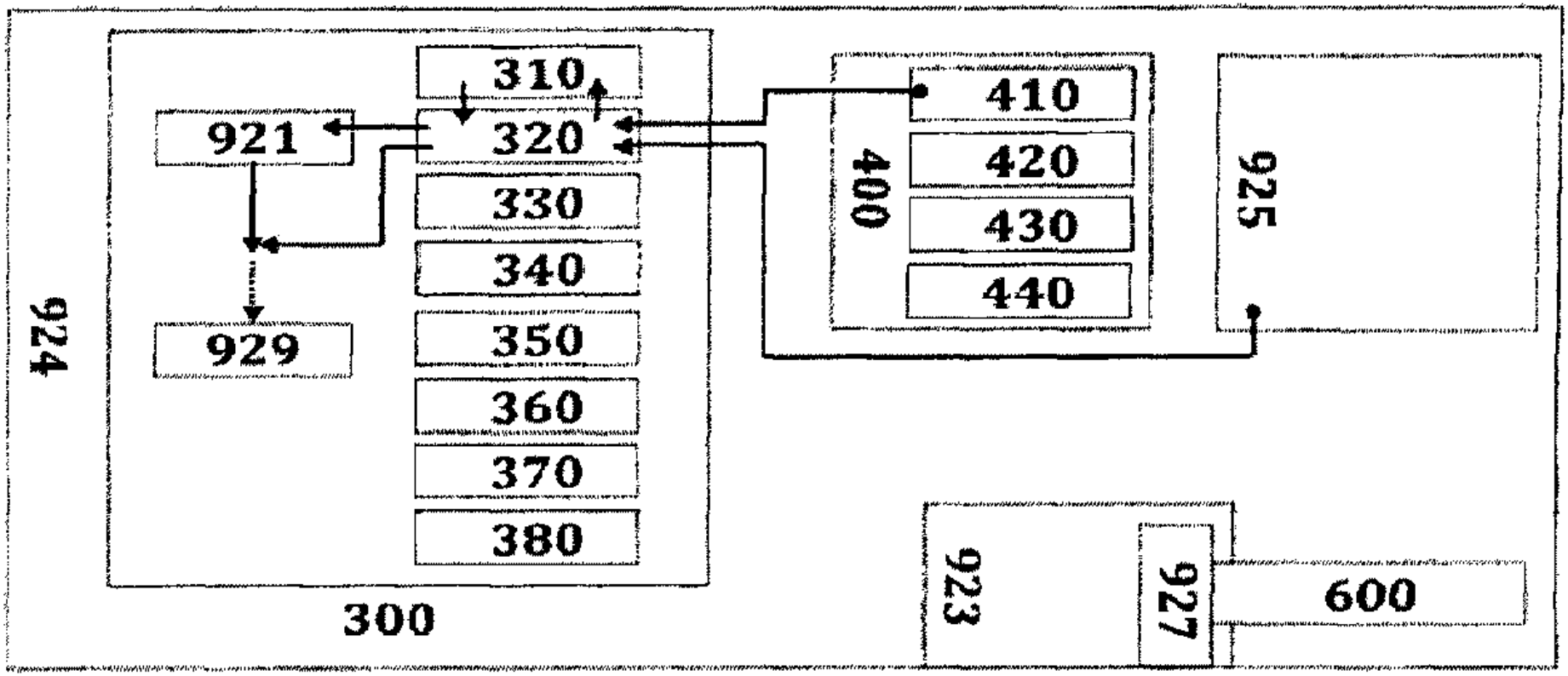



FIGURE 14


924
310
320
330
340
350
360
370
380
921
929
300
400
410
420
430
440
925
923
927
600