(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 June 2026 (11.06.2026)

WIPO | PCT

(10) International Publication Number
WO 2026/120226 A1

(51) International Patent Classification:
H04L 9/08 (2006.01)

(21) International Application Number:
PCT/FI2025/060124

(22) International Filing Date:
28 November 2025 (28.11.2025)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

24217184.1	03 December 2024 (03.12.2024)	EP
20255074	31 January 2025 (31.01.2025)	FI
PCT/FI2025/060109	21 November 2025 (21.11.2025)	FI

(71) Applicant: **GURULOGIC MICROSYSTEMS OY** [FI/FI]; Linnankatu 34, 20100 Turku (FI).

(72) Inventor: **KÄRKKÄINEN, Tuomas**; Tikkaajankatu 7, 20400 Turku (FI).

(74) Agent: **MOOSEDOG OY**; Kurjenmäenkatu 10 B 49, 20700 Turku (FI).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH,

(54) Title: ARRANGEMENT AND METHOD FOR SECURELY ENABLING GROUP COMMUNICATION

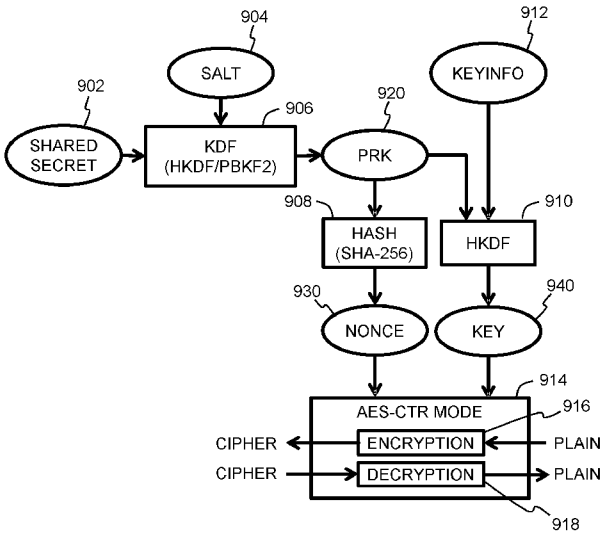

Fig. 9

(57) Abstract: Disclosed is an arrangement (800) and a method (1000) for securely enabling a group communication. The arrangement comprises at least one trust provider (802) and a plurality of clients (804, 806, 808). The trust provider(s) is configured to: receive a request for establishing a session of said communication, from a first client (804); generate a shared secret (902), by executing a first random number generator (810); digitally encapsulate shared secret in a plurality of versions; and distribute a respective version to a corresponding client. Each client is configured to: decapsulate the respective version; obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is used to provide a salt component, derive one or more crypto-products (920, 930, 940), by executing second random number generator (812, 814, 816) utilising the shared secret and the salt component; and employ said crypto-products to encrypt (916) outgoing communication from said client, and decrypt

WO 2026/120226 A1

[Continued on next page]

TJ, TM, TN, TR, TT, TZ, UA, UG, US, UY, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, CV, GH, GM, KE, LR, LS, MU, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *in black and white; the international application as filed contained color or greyscale and is available for download from PATENTSCOPE*

(918) incoming communication at said client.

ARRANGEMENT AND METHOD FOR SECURELY ENABLING GROUP COMMUNICATION

FIELD OF THE INVENTION

The invention concerns generally the technical field of security needed in using digital services among two or more communicating parties. The invention concerns the task of distributing a shared secret between two or more clients for later use in communications or other applications where cryptographic methods are utilized. In particular, the present disclosure relates to arrangements for securely enabling group communications. Furthermore, the present disclosure also relates to methods for securely enabling group communications.

BACKGROUND OF THE INVENTION

Many applications of cryptography require two or more parties to have access to a so-called shared secret. Enabling secure group communications may, for example, have such a requirement for the two or more parties engaging in the group communications. In such applications, there are several challenges associated with delivery of the shared secret to the two or more parties, and with how the shared secret is utilised by the two or more parties for secure applications (such as secure group communications).

A straightforward example is symmetric cryptography, in which a transmitting party uses the shared secret as such as an encryption key and a receiving party uses the same shared secret as a decryption key. Better security against certain attack vectors can be achieved by not using the shared secret as such as a key but making the communicating parties use the shared secret as an input to some further cryptographic operations to derive the actual key or keys. Other examples of using a shared secret include, but are not limited to the use of

bots and/or robots, meaning artificial intelligence powered non-human actors or agents that operate under authorization of a party.

The task of distributing a shared secret among parties for cryptographic applications is not a straightforward one, as it involves a kind of vicious circle: one would need a secure method to communicate the shared secret, but the shared secret itself – or access to it by the parties – is the prerequisite of making the communications secure. Challenges in securely distributing a shared secret have led, for example, to introducing the principle of asymmetric cryptography, in which keys appear in pairs: a party can safely publish its public key and ask everyone to use that public key for encrypting transmissions towards its known holder, as the only possible way to decrypt such encrypted transmissions is with the corresponding private key that the party holds strictly to itself. While asymmetric cryptography removes the need to distribute a shared secret, it has its drawbacks. For example, generating and using pairs of public and private keys that are long enough to adequately resist brute force attacks is computationally intensive to a degree that may make it impossible or at least commercially unattractive for use in simpler digital devices like sensor nodes, digitally controlled household appliances, or the like. Furthermore, asymmetric key exchange protocols are particularly vulnerable in quantum computing era.

There are also challenges associated with storing the shared secrets or digital products generated using the shared secrets, at devices of the parties. These devices usually have a proprietary architecture to protect these secrets, but even if such solutions are certified, they are still audited by a third party which is unknown to the parties. Furthermore, the devices of the parties do not offer complete security against entities that have sufficient technical understanding of the vulnerabilities hidden in the devices and possible backdoors that can be used to break the parties' confidentiality.

Utilizing the shared secret for secure applications also has its challenges. Some challenges may, for example, be that entropy sources (which generated the shared secret) may be unreliable or manipulatable, devices utilizing the shared secret may be faulty or malicious, randomness quality can be affected by external disturbances and environmental changes, and ensuring compliance with standards and certification may be problematic.

Moreover, in some applications, the shared secret is used in random number generators (RNGs), to exploit deterministic randomness. However, in current RNGs, parties rarely have direct control over the process as said RNGs typically operate in the background without user involvement. In such cases, unauthorized parties can attack and exploit RNGs for manipulation purposes, such as predicting outputs of the RNGs, thereby compromising security in applications. For example, in network communications, the RNGs can be triggered by messages originating from external sources.

Furthermore, in current solutions, encryption keys are typically required to be stored in secure environments within the parties' devices. Some examples of such secure environments include Trusted Execution Environment Operating System (TEEOS), Trusted Platform Module (TPM), or Hardware Security Module (HSM). Such storage increases a risk of exposure of the encryption keys to physical attacks or malware on the devices.

Therefore, in light of the foregoing discussion, there exists a need to overcome the aforementioned drawbacks.

SUMMARY

This summary is provided to introduce a selection of concepts in a simplified form that are further described below in the detailed description. This summary is not intended to identify key features or

essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

It is an objective to provide arrangements and methods for securely enabling group communications in a way that a shared secret between parties is processed securely, under the parties' control, to exploit trustworthy deterministic randomness. The arrangements and methods provide a quantum-resilient solution for cryptography. Other objectives are to provide trust providers for securely enabling the group communications, and to provide clients for securely participating in the group communications.

It is also an objective to provide methods and arrangements for distributing shared secrets between parties in a way that only requires reasonable complicatedness and computational capacity but is still secure enough to resist attacks of even the level that will become possible with quantum computers. A further objective is to provide a secure way of using symmetric cryptography on basis of preconditioned exchange of key-related information.

According to a first aspect, there is provided an arrangement for securely enabling a group communication, the arrangement comprising at least one trust provider and a plurality of clients that are communicably coupled to each other and to their respective trust provider. The at least one trust provider is configured to: receive a request for establishing a session of the group communication, from a first client amongst the plurality of clients; generate a shared secret corresponding to the session, by executing a first random number generator; digitally encapsulate the shared secret in a plurality of versions, using respective ephemeral keys corresponding to the plurality of clients, the respective ephemeral keys being available with the at least one trust provider; and distribute a respective version amongst the plurality of versions of the shared secret, to a corresponding client amongst the plurality of clients, prior to a start of the session. Each client amongst the plurality of clients

is configured to: decapsulate the respective version of the shared secret, using their respective private key; obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is to be used to provide a salt component; derive one or more crypto-products, by executing a second random number generator, wherein the shared secret and the salt component are used as inputs to the second random number generator; and employ the one or more crypto-products to encrypt outgoing communication from said client, and decrypt incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients.

It should be appreciated that not all clients need to be communicably coupled to a same trust provider. The plurality of clients can certainly be all connected to a same at least one trust provider or some of the clients can be connected to a first trust provider while others are connected to a second trust provider. In a further example a first client is connected to a first trust provider and a second client is connected to a second trust provider.

According to an embodiment, when deriving the one or more crypto-products, the shared secret is used as a seed component, and wherein each client is configured to: combine the seed component and the salt component using a first derivation function of the second random number generator, to derive a Pseudo-Random Key (PRK).

Further optionally, hash the PRK using a nonce derivation function, to derive a common nonce; and combine the PRK and contextual information associated with encryption, using a second derivation function of the second random number generator, to derive an encryption key, wherein the encryption key and the common nonce are utilised for performing the encryption of the outgoing communication and the decryption of the incoming communication.

Technical effect of calculating PRK as an intermediate key in memory protects the shared secret and crypto products. Since the PRK is calculated and stored into (protected secure) memory, then if this is revealed the attacker cannot go backwards to shared secret. An example of a (first and/or second) derivation function is Key Derivation Function (KDF).

According to an embodiment, the encryption key is dynamically re-computable without requiring its storage at said client.

According to an embodiment, each client is further configured to: receive a start value (nonceStart) for stream nonce; and retrieve a group size value and a member index of said client, the group size value being equal to a number of clients in the plurality of clients,
wherein when encrypting the outgoing communication, said client is configured to: determine a stream sequence value for each data packet, based on the start value for stream nonce, the group size value, the member index, and a packet sequence number; define a stream nonce, based on the common nonce and the stream sequence value; create a cipher block with the encryption key; create an outgoing keystream with the cipher block and the stream nonce; combine each data packet with the outgoing keystream, using an exclusive OR (XOR) operation, to generate encrypted data packets; and add the stream sequence value for each encrypted data packet into a header of said encrypted data packet, prior to transmitting said encrypted data packet.

As an example, when considering IPv4 and IPv6 related packet data communication a stream sequence value could be used. This way shared secret can be distributed between N to N users or clients in IP connection with each other. This way an alternative way to implement for virtual private network (VPN) can be implemented for example by using TLS 1.3.

According to an embodiment, when decrypting the incoming communication, said client is configured to: obtain the stream sequence

value, from the header of each encrypted data packet that is received at said client; define the stream nonce, based on the common nonce and the stream sequence value; create a plaintext block with the encryption key; create an incoming keystream with the plaintext block and the stream nonce, the incoming keystream being same as the outgoing keystream; and combine each encrypted data packet with the incoming keystream, using the XOR operation, to generate decrypted data packets.

According to an embodiment, the at least one trust provider is further configured to: detect when any one of the following occurs: a duration of the session of the group communication exceeds a first predefined threshold, or a number of data packets exchanged during the group communication exceeds a second predefined threshold; generate a new shared secret corresponding to the session upon said detection, by executing the first random number generator; digitally encapsulate the new shared secret in a plurality of new versions, using the respective ephemeral keys corresponding to the plurality of clients; and distribute respective versions amongst the plurality of new versions of the shared secret to corresponding clients amongst the plurality of clients, during the session.

According to any of the preceding claims, wherein each client is further configured to: detect when any one of the following occurs: a duration of the session of the group communication exceeds a first predefined threshold, or a number of data packets exchanged during the group communication exceeds a second predefined threshold; and refreshing at least one of: the shared secret or the salt component.

According to an embodiment, each client amongst the plurality of clients is further configured to obfuscate the shared secret and the one or more crypto-products, upon an end of the session.

According to an embodiment, wherein when a next group communication is to be enabled amongst the plurality of clients, at least one

of: the shared secret, the salt component, is changed and distributed amongst the plurality of clients.

According to an embodiment, upon decapsulation of the respective version of the shared secret at each client, the shared secret is extracted into a programmatically protected memory, the programmatically protected memory being one of: a segregated memory provided by Hardware Security Module (HSM), a secure enclave, a Trusted Platform Module (TPM), a secure element, or a guarded heap memory.

According to an embodiment, the identification credential is obtained by at least one of: personal secret-based authentication, identity card-based authentication, cryptographic authentication, wireless communication-based authentication, token-based authentication, notification-based authentication, code-based authentication, digital identity-based authentication, biometric and/or neural authentication, anti-spoofing authentication.

According to an embodiment, the second random number generator employs a Deterministic Random Bit Generator (DRBG) algorithm when deriving the one or more crypto-products, the DRBG algorithm being one of: a counter mode DRBG (CTR-DRBG) algorithm, a Hash-based Message Authentication Code (HMAC)-based DRBG algorithm, a HASH-DRBG algorithm.

According to an embodiment, the one or more crypto-products comprise at least one of: a symmetric key, an asymmetric key pair, one or more keys for encrypting and/or wrapping other key(s), a nonce value or initialization vector, a secure key from an identification credential, a secure password, a derivative of the secure password, a session key, a one-time key, a pre-shared key (PSK) of a Transport Layer Security (TLS)-PSK model.

According to an embodiment, wherein the group communication is one of: a group audio call, a group conference call, a group video call, a group chat, a group email conversation, a group workspace

communication, a group broadcast or a group network. An additional group communication case is a packet data session. In that regards an embodiment comprises securing data packets in a communication session such as IPv4 or IPv6 based communication session. The communication session can be considered to be any data transfer between two or more entities.

According to a second aspect, there is provided a trust provider for securely enabling a group communication between a plurality of clients, the trust provider being communicably coupled to at least one client of the plurality of clients, wherein the trust provider is configured to: receive a request for establishing a session of the group communication, from a first client amongst the plurality of clients; generate a shared secret corresponding to the session, by executing a first random number generator; digitally encapsulate the shared secret in a plurality of versions, using respective ephemeral keys corresponding to the plurality of clients, one or more of the respective ephemeral keys being available with the trust provider; and distribute a respective version amongst the plurality of versions of the shared secret, to a corresponding client amongst the plurality of clients, prior to a start of the session,

wherein the respective version of the shared secret is decapsulated at each client amongst the plurality of clients, using a respective private key, the shared secret is used at each client to derive one or more crypto-products by executing a second random number generator, and the one or more crypto-products are employed at each client to encrypt outgoing communication from said client, and to decrypt incoming communication at said client. As an example, the trust provider (such as a first trust provider) can be coupled to all of the plurality of clients, or it can serve a subset of the clients. In the latter scenario, rest of the clients are coupled to another trust provider (such as to a second trust provider). In a further example a first client is coupled to a first trust provider and a second client is coupled to a second trust provider.

According to a third aspect, there is provided a client for securely participating in a group communication, the client being communicably coupled to a trust provider and at least one other client. The client is configured to: decapsulate a respective version of a shared secret, using a respective private key, the respective version being received from the trust provider prior to a start of a session of the group communication, wherein the shared secret is generated by the trust provider using a first random number generator; obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is used to provide a salt component; derive one or more crypto-products, by executing a second random number generator, wherein the shared secret and the salt component are used as inputs to the second random number generator; and employ the one or more crypto-products to encrypt outgoing communication from said client, and decrypt incoming communication at said client, the incoming communication being received from at least one other client.

According to a fourth aspect, there is provided a method for securely enabling a group communication, the method being implemented by an arrangement comprising at least one trust provider and a plurality of clients that are communicably coupled to each other and their respective trust provider. The method comprises: receiving, at the at least one trust provider, a request for establishing a session of the group communication, from a first client amongst the plurality of clients; generating a shared secret corresponding to the session, at the at least one trust provider, by executing a first random number generator; digitally encapsulating the shared secret in a plurality of versions, at the at least one trust provider, using respective ephemeral keys corresponding to the plurality of clients, the respective ephemeral keys being available with the at least one trust provider; distributing a respective version amongst the plurality of versions of the shared secret, from the at least one trust provider to a corresponding client amongst the plurality of clients, prior

to a start of the session; decapsulating the respective version of the shared secret, at each client amongst the plurality of clients, using a respective private key; obtaining an identification credential that is indicative of a user's authorization for using the shared secret, at each client, wherein the identification credential is used to provide a salt component; deriving one or more crypto-products at each client, by executing a second random number generator, wherein the shared secret and the salt component are used as inputs to the second random number generator; and employing the one or more crypto-products, at each client, for encrypting outgoing communication from said client, and decrypting incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients.

According to an embodiment, at the step of deriving the one or more crypto-products at each client, the shared secret is used as a seed component, and wherein the step of deriving the one or more crypto-products comprises: combining the seed component and the salt component using a first derivation function of the second random number generator, to derive a Pseudo-Random Key (PRK).

Further optionally the method comprises hashing the PRK using a nonce derivation function, for deriving a common nonce; and combining the PRK and contextual information associated with encryption, using a second derivation function of the second random number generator, for deriving an encryption key;

wherein the encryption key and the common nonce are utilised for performing the encryption of the outgoing communication and the decryption of the incoming communication.

An example of derivation functions is a key derivation function (KDF).

According to an embodiment, the method further comprising dynamically re-computing the encryption key, without requiring its storage at said client.

According to an embodiment, the step of employing the one or more crypto-products, at each client, for encrypting the outgoing communication from said client comprises: receiving a start value (nonceStart) for stream nonce; retrieving a group size value and a member index of said client, the group size value being equal to a number of clients in the plurality of clients; determining a stream sequence value for each data packet, based on the start value for stream nonce, the group size value, the member index, and a packet sequence number; defining a stream nonce, based on the common nonce and the stream sequence value; creating a cipher block with the encryption key; creating an outgoing keystream with the cipher block and the stream nonce; combining each data packet with the outgoing keystream, using an exclusive OR (XOR) operation, for generating encrypted data packets; and adding the stream sequence value for each encrypted data packet into a header of said encrypted data packet, prior to transmitting said encrypted data packet.

According to an embodiment, the step of employing the one or more crypto-products, at each client, for decrypting the incoming communication at said client comprises: obtaining the stream sequence value, from the header of each encrypted data packet that is received at said client; defining the stream nonce, based on the common nonce and the stream sequence value; creating a plaintext block with the encryption key; creating an incoming keystream with the plaintext block and the stream nonce, the incoming keystream being same as the outgoing keystream; and combining each encrypted data packet with the incoming keystream, using the XOR operation, for generating decrypted data packets.

According to an embodiment, the method further comprises: detecting, at the at least one trust provider, when any one of the following occurs: a duration of the session of the group communication exceeds a first predefined threshold, or a number of data packets exchanged during

the group communication exceeds a second predefined threshold; generating a new shared secret corresponding to the session upon said detection, by executing the first random number generator; digitally encapsulating the new shared secret in a plurality of new versions, using the respective ephemeral keys corresponding to the plurality of clients; and distributing respective versions amongst the plurality of new versions of the shared secret to corresponding clients amongst the plurality of clients, during the session.

According to an embodiment, the method further comprises: detecting, at each client, when any one of the following occurs: a duration of the session of the group communication exceeds a first predefined threshold, or a number of data packets exchanged during the group communication exceeds a second predefined threshold; and refreshing at least one of: the shared secret or the salt component.

According to an embodiment, the method further comprises obfuscating the shared secret and the one or more crypto-products, at each client, upon an end of the session.

According to an embodiment, wherein when a next group communication is to be enabled amongst the plurality of clients, the method further comprises changing and distributing amongst the plurality of clients, at least one of: the shared secret, the salt component.

According to a fifth aspect, there is provided a computer program product comprising at least one non-transitory machine-readable data storage medium having stored thereon one or more sets of one or more machine-executable instructions that are configured to, when executed by one or more processors, cause the execution of a method according to the fourth aspect.

The present disclosure also provides an arrangement for distributing a shared secret among clients. The arrangement is configured to, in response to receiving a first request from a first client, generate a first ephemeral key and one or more further ephemeral keys, and to

digitally encapsulate the shared secret in a first version and one or more further versions using a key encapsulation mechanism. Said first version is the shared secret encapsulated with the first ephemeral key and subject to decapsulation with a first private key, a paired public key of which is known to the arrangement as being a public key of the first client. Each said further version is the shared secret encapsulated with a respective one of said one or more further ephemeral keys and subject to decapsulation with a respective further private key, a paired public key of which is known to the arrangement as being a public key of a respective further client. The arrangement is configured to store said first and said one or more further versions in association with a common identifier, and to respond to receiving a second request from one of said first or further clients identifiable as a holder of one of said first or further private keys, said second request referring said common identifier, by transmitting that of said first or further versions that is subject to decapsulation with the private key, the paired public key of which is known to the arrangement as being a public key of said one of said first or further clients.

According to an embodiment, the arrangement is configured to generate said one or more further ephemeral keys in association with respective client identifiers revealed by said first request. This involves at least the advantage that in the continuation, the encapsulated versions will be identifiable and possible to associate with each respective client.

According to an embodiment, the arrangement comprises a hardened true random number generator subsystem configured to generate said shared secret. This involves at least the advantage that highest possible entropy and security of the shared secret can be ensured.

According to an embodiment, the arrangement is configured to transmit said first version to said first client in a response transmission responding to said first request. This involves at least the advantage that the initiator client can be given large freedom concerning how to handle the shared secret in the continuation.

According to an embodiment, the arrangement is configured to use a value received in said first request as said shared secret. This involves at least the advantage that the operation of the arrangement can be adapted to the handling of shared secrets in a very versatile way.

According to an embodiment, the arrangement is configured to generate said common identifier as a crypto-product derived from said shared secret. This involves at least the advantage that the generation of additional digital data can be kept at minimum.

According to an embodiment, the arrangement is configured to use said shared secret or a crypto-product derived therefrom as an encryption key to encrypt digital data received from the first client, store the encrypted digital data in association with said common identifier, and respond to receiving said second request from said one of said further clients by transmitting the encrypted digital data to said one of said further clients. This involves at least the advantage that the arrangement can be used to securely store, forward, and play back digital data for the needs of clients.

The present disclosure also provides a method for distributing a shared secret among clients. The method comprises, in response to receiving a first request from a first client, generating a first ephemeral key and one or more further ephemeral keys. The method comprises digitally encapsulating the shared secret in a first version and one or more further versions using a key encapsulation mechanism. Said first version is the shared secret encapsulated with the first ephemeral key and subject to decapsulation with a first private key, a paired public key of which is known to the arrangement as being a public key of the first client. Each said further version is the shared secret encapsulated with a respective one of said one or more further ephemeral keys and subject to decapsulation with a respective further private key, a paired public key of which is known to the arrangement as being a public key of a respective further client. The method comprises storing said first and said one or more

further versions in association with a common identifier, and responding to receiving a second request from one of said first or further clients identifiable as a holder of one of said first or further private keys, said second request referring said common identifier, by transmitting that of said first or further versions that is subject to decapsulation with the private key, the paired public key of which is known to the arrangement as being a public key of said one of said first or further clients.

According to an embodiment, the generating of said one or more further ephemeral keys takes place in association with respective client identifiers revealed by said first request. This involves at least the advantage that in the continuation, the encapsulated versions will be identifiable and possible to associate with each respective client.

According to an embodiment, the method comprises generating said shared secret in a hardened true random number generator subsystem. This involves at least the advantage that highest possible entropy and security of the shared secret can be ensured.

According to an embodiment, said first version is transmitted to said first client in a response transmission responding to said first request. This involves at least the advantage that the initiator client can be given large freedom concerning how to handle the shared secret in the continuation.

According to an embodiment, a value received in said first request is used as said shared secret. This involves at least the advantage that the operation of the arrangement can be adapted to the handling of shared secrets in a very versatile way.

According to an embodiment, the method comprised generating said common identifier as a crypto-product derived from said shared secret. This involves at least the advantage that the generation of additional digital data can be kept at minimum.

According to an embodiment, the method comprises using said shared secret or a crypto-product derived therefrom as an encryption key

to encrypt digital data received from the first client, storing the encrypted digital data in association with said common identifier, and responding to receiving said second request from said one of said further clients by transmitting the encrypted digital data to said one of said further clients. This involves at least the advantage that the arrangement can be used to securely store, forward, and play back digital data for the needs of clients.

The present disclosure also provides a computer program product comprising one or more sets of one or more machine-executable instructions that are configured to, when executed by one or more processors, make said one or more processors execute a method of for distributing a shared secret among clients, as described above.

BRIEF DESCRIPTION OF THE DRAWINGS

In the drawings:

Fig. 1 illustrates a sequence of events between two clients and a trusted third party, for exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients,

Fig. 2 illustrates another sequence of events between two clients and a trusted third party, for exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients,

Fig. 3 illustrates a method executed by some functional units at an arrangement operated as the trust provider in a sequence of events like those in figs. 1 and 2,

Fig. 4 illustrates yet another sequence of events between clients and a trusted third party, for exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients,

Fig. 5 illustrates still another sequence of events between clients and a trusted third party, for exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients,

Fig. 6 illustrates yet another sequence of events between clients and a trusted third party, for exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients,

Fig. 7 illustrates still another sequence of events between clients and a trusted third party, for exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients,

Fig. 8 illustrates an arrangement for enabling a group communication,

Fig. 9 illustrates an example of how each client in the group communication may use a shared secret to derive one or more crypto-products, and

Fig. 10 illustrates steps of a method for securely enabling the group communication, in accordance with one or more embodiments of the present disclosure.

DETAILED DESCRIPTION

In the following description, reference is made to the accompanying drawings, which form part of the disclosure, and in which are shown, by way of illustration, specific aspects in which the present disclosure may be placed. It is understood that other aspects may be utilised, and structural or logical changes may be made without departing from the scope of the present disclosure. The following detailed description, therefore, is not to be taken in a limiting sense, as the scope of the present disclosure is defined by the appended claims.

For instance, it is understood that a disclosure in connection with a described method may also hold true for a corresponding device or system configured to perform the method and vice versa. For example, if a specific method step is described, a corresponding device may include a unit to perform the described method step, even if such unit is not explicitly described or illustrated in the figures. On the other hand, for example, if a specific apparatus is described based on functional units, a corresponding method may include a step performing the described functionality, even if such step is not explicitly described or illustrated in the figures. Further, it is understood that the features of the various example aspects described herein may be combined with each other, unless specifically noted otherwise.

Concerning terminology, in this description the term crypto-product (or crypto-product) means the output of a cryptographic operation. Examples of crypto-products include but are not limited to encryption keys, sets of keys, certificates, and digital signatures. Cryptographic operations (or crypto-operations for short) can in many cases be further characterised as either encrypting operations or decrypting operations.

Fig. 1 illustrates the exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients. In Fig. 1 it is assumed that there are at least two clients A and B who would benefit from getting hold of a common shared secret. There may be more clients C, D, … with the same intentions. The parties that are called clients may be client devices that are operated by human users, or may be human users, in which case the operations described in the following take place in respective user devices operated by their human users. Yet further option is that the clients are bots and/or robots, meaning artificial intelligence powered non-human actors or agents that operate under authorization of a human user. In this regard, when any client is described to perform any processing step, it will be appreciated that said processing step is performed by a client device or a user device.

The client device is a device acting as a client in a client-server architecture, meaning that it interacts with a server or a centralized system to request or consume services or data. Examples of the client devices include, but are not limited to, computing devices (such as desktop computers, laptop computers, tablet computers, and similar), smartphones, Internet of Things (IoT) devices, gaming devices, media devices, streaming devices, and industrial and enterprise devices (such as workstations). Additionally, or alternatively, at least some of the clients may be independently operating electronic devices, examples of which include but are not limited to node devices of a communications network, sensor devices of a sensor network, and devices operating as a part of a building automation network. As a difference to user devices, independently operating electronic devices operate without direct user interaction, executing one or more computer programs stored therein and/or at their disposal. As a further example, at least some of the clients may be software processes executing in some execution environment that may be centralized or distributed.

Concerning the following description of Figs 1-7, it is not important what purpose the shared secret is to be used for, but as an illustrative and non-limiting example it could be used as a cryptographic key (or, as a seed for deriving a cryptographic key) for use in further communications protected with symmetric encryption between the clients. While both symmetric and asymmetric encryption involve their respective advantages, symmetric encryption is more preferable for use in devices that only have limited amounts of power and/or computational capacity because it consumes less these resources.

In addition to the clients, fig. 1 illustrates a party called a trust provider. The trust provider is, as the name suggests, a party that can be trusted, for example, to reliably identify clients that communicate with it and to produce random numbers of high entropy. The trust provider is typically a party who maintains and operates a server or an arrangement

of linked servers that have been objectively and impartially evaluated and certified for secure operation. The trust provider may also be any device which can perform computations in a secure manner. For example, devices having Trusted Execution Environments (TEEs), hardware security modules (HSMs), secure enclaves, and similar, may be employed as a trust provider in the embodiments disclosed herein. When the trust provider is described to perform any processing step, it will be appreciated that said processing step is performed by said server or said arrangement of linked servers. In other words, the trust provider can be understood to be at least one server that has been objectively and impartially evaluated and certified for secure operation. The trust provider can be implemented as a single server or as a plurality of servers which are communicably coupled to each other. A digital certificate issued by the trust provider can be relied upon as a proof of the respective piece of digital information being authentic and having unchallenged integrity.

At the first step shown in fig. 1, the trust provider receives a request 101 from the client marked as client A in fig. 1. The request 101 contains some kind of an indication that client A wants to establish a secret that is to be shared between them and at least one other client B, (C, D…). The request 101 may also be assumed to contain identifier(s) of the other client(s) with which the secret is to be shared. This is not necessary in all cases, however. One may also imagine cases where a client A asks the trust provider to establish a secret to be shared with other client(s) B, (C, D…) so routinely that already the reception of the request 101 tells the trust provider, which other client(s) should be involved. A technical advantage of this flexible request-handling mechanism is that it enables dynamic and context-aware establishment of secure group communication sessions, reducing initialization latency and allowing the trust provider to automatically manage participant associations without requiring predefined group configurations or repeated key exchange negotiations.

In response to receiving the request 101, the trust provider generates a shared secret at step 102. Preferably, the trust provider uses a certified source of true randomness for generating the shared secret at step 102. Examples of suitable sources of true randomness include, but are not limited to, specialized hardware known as hardware random number generators (HRNG), true random number generators (TRNG), non-deterministic random bit generators (NRBG), or physical random number generators. Physical phenomena that may serve as sources of randomness include, but are not limited to, electrical noise, chained free-running oscillators, nuclear decay, shot noise, and quantum optical phenomena.

Step 103 in fig. represents digitally encapsulating the shared secret using a key encapsulation mechanism. Here, it is assumed that an intended recipient possesses a key pair consisting of a private key and the corresponding public key. Usually designated with the acronym KEM, according to a general definition a key encapsulation mechanism allows a sender who knows the public key of the intended recipient to simultaneously generate a short random secret key and an encapsulation or ciphertext of the secret key by the KEM's encapsulation algorithm. The recipient who knows the private key corresponding to the public key can recover the same random secret key from the encapsulation by the KEM's decapsulation algorithm. In the present context, the shared secret generated at step 102 is considered as the "random secret key" in the general definition of a KEM.

In particular, at step 103 the trust provider encapsulates the shared secret in a first version and one or more further versions. The first version is the shared secret encapsulated so that it is subject to decapsulation with a private key held by client A. Each further version is the shared secret encapsulated so that it is subject to decapsulation with a respective private key held by one of the other clients B, (C, D,...).

At the next step in fig. 1, the trust provider transmits a response 104 to client A. In this embodiment, the trust provider transmits

the first version mentioned above to client A in the response 104. In other words, the response 104 responds to the request 101 that the trust provider received from client A earlier and delivers the shared secret generated at step 102 to client A in an encapsulated form that client A is able to decapsulate with a private key known to client A.

At the next step in fig. 1, client A sends an invitation 105 to the other client B who they wish to get hold of the same shared secret. If there are several such other clients B, C, D,..., client A sends a respective invitation to each one of them. The purpose of the invitation 105 is to make the other client(s) aware that there is a respective encapsulated version of the shared secret waiting for them at the trust provider. Exact ways in which this information is conveyed to the other client(s) are described in more detail later in this text. In general, we may assume that a common identifier exists that identifies all encapsulated versions of the shared secret at the trust provider.

At the next step in fig. 1, client B sends what can be called here a second request 106 to the trust provider. Essentially, with the second request 106 client B asks to get the respective encapsulated version of the shared secret. If other clients C, D,... also received an invitation from client A, they too may send their respective second requests to the trust provider. Every client who sends requests to the trust provider should authenticate themselves to the trust provider and be capable of making reference to the shared secret that is to be distributed. Advantageous ways in which such reference can be made are described in detail later in this text.

At the last step shown in fig. 1, the trust provider responds to the second request 106 with a further response 107, in which the trust provider transmits one of the further versions of the encapsulated shared secret. The trust provider knows who was the client from which it received the second request 106, so the version included in the further response

107 is the one that is subject to decapsulation with the private key held by the originator of the second request 106.

Herein, the shared secret is encapsulated in a client-specific manner using the KEM with ephemeral keys corresponding to each client. Through this approach, the trust provider generates distinct encapsulated versions of the shared secret for every authorized client. Each encapsulated version is uniquely bound to the corresponding client's ephemeral key, ensuring that compromise of one client's encapsulated version does not affect the confidentiality or integrity of the versions distributed to other clients.

Fig. 2 shows another embodiment of the exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients. The parties shown are the same as in fig. 1. As a difference to the embodiment of fig. 1, in fig. 2 the shared secret is not generated by the trust provider but by the client that was called client A in the description of fig. 1. The step of generating the shared secret is shown with reference designator 201 in fig. 2.

While there occurs the transmission of the so-called first request 202 from client A to the trust provider also in fig. 2, there is the difference that in fig. 2 the indication included in the first request 202 that client A wants to establish a secret (that is to be shared) actually contains the shared secret itself. Step 103 is the same as the similarly designated step in fig. 1 and involves the trust provider digitally encapsulating the shared secret in a first version and one or more further versions, using a key encapsulation mechanism. Similar to fig. 1, the first version is subject to decapsulation with a private key held by client A. Each further version is subject to decapsulation with a respective further private key held by respective one of the other client(s) B, (C, D,...).

At the next step in fig. 2, the trust provider transmits a response 203 to client A. In this embodiment, the trust provider may transmit the first version mentioned above to client A in the response 203, but

this is not obligatory as client A knows the shared secret already (because they generated it themselves at step 201). In any case, the response 203 responds to the request 202 that the trust provider received from client A earlier and may also deliver the shared secret generated at step 201 to client A in an encapsulated form that client A is able to decapsulate with a private key known to client A. At least, the response 203 is a confirmation for client A that the further encapsulated versions of the shared secret are ready at the trust provider, waiting for the other client(s) B, (C, D,...).

The invitation(s) 105 transmitted by client A, the request(s) 106 transmitted by the other client(s), and the response(s) 107 by the trust provider are the same as in the embodiment of fig. 1 above, as indicated by the same reference designators.

Fig. 3 illustrates some functional units and their operation at an arrangement operated as the trust provider in a sequence of events like those in figs. 1 and 2. The arrangement may comprise a so-called hardened true random generator subsystem 301 that is configured to generate shared secrets of maximal entropy. An advantage of having a hardened true random generator subsystem 301 in (or at least at the disposal of) the trust provider is that the generation of shared secret is not prone to hacking in some way that may be possible if one had to rely upon a random number generator operating in a user device of unknown (or not completely known) origin. If the arrangement is to be used only for operation in accordance with fig. 2, the hardened true random generator subsystem 301 is naturally not needed, at least not for the purpose described here, as the shared secret will then always come from the client initiating the distribution of the shared secret.

A shared secret generated by the hardened true random generator subsystem 301 is shown with reference designator 302 in fig. 3. The shared secret 302 may be called a strong one due to the guaranteed security of and high entropy associated with the hardened true random

generator subsystem 301. If operation according to fig. 2 is considered, there is only the difference that the arrangement would receive the shared secret in a request from the initiating client and not generate it by itself.

Block 303 in fig. 3 represents the generation of a common identifier, called the DATA-ID 304, using the shared secret 302 as input data. Generating the DATA-ID 304 in block 303 may involve calculating a hash or otherwise generating a crypto-product that has a one-to-one correspondence to the shared secret 302. The method used in block 303 should be cryptographically unidirectional, meaning that from a generated DATA-ID 304 it is impossible to go backwards or find out the shared secret 302. Many widely used hashing methods have this characteristic, for which reason block 303 is marked as a hash in fig. 3.

The arrangement comprises a key generator 305, the purpose of which is to generate the keys to be used in encapsulating the shared secret. As many keys for encapsulating should be generated as there are clients among which the shared secret is to be distributed. Additionally, it is important to generate the keys for encapsulating so that each encapsulated version of the shared secret is subject to decapsulation with a private key held by the client for which that version is meant.

In fig. 3, it is assumed that there are N clients, where N is a positive integer equal to or larger than 2, among which the shared secret is to be distributed. It is further assumed that user identifiers (UID's) 306, 307, and 308 of those clients are known to the key generator 305. Knowing the user identifiers 306, 307, and 308 of the clients enables the arrangement of fig. 3 to find out at least one respective public key of each client. When a public key of a client is known, the arrangement knows that a corresponding paired private kay is held by the same client. The key generator 305 may then generate the keys for use in the encapsulating process so that each encapsulated version of the shared secret will

be subject to decapsulation with the private key, the paired public key of which is known to the arrangement of fig. 3.

The keys for use in the encapsulation are shown with reference designators 309, 310, and 311 in fig. 3. For reasons described in more detail later, these keys are called ephemeral keys 309, 310, and 311. The first ephemeral key 309 is conceptually associated with the first user identifier 306, the second ephemeral key 310 is conceptually associated with the second user identifier 307, and so on.

Reference designator 312 shows the encapsulator functionality in the arrangement. As already indicated above, the encapsulator functionality 312 is configured to digitally encapsulate the shared secret 302 in a first version 313 and one or more further versions 314 and 315, each time with the respective one of the ephemeral keys 309, 310, and 311. As the shared secret may be called by the acronym SS, each i:th encapsulated version thereof may be designated

ENC(SS) #i, where

$i \in [1,\ldots,N]$.

Each i:th encapsulated version 313, 314, and 315 is subject to decapsulation with a private key, a corresponding public key of which is known to the arrangement as being a public key of the respective i:th client.

As shown with reference designator 316, the arrangement is configured to store the first version 313 and the one or more further versions 314 and 315 in association with the common identifier or DATA-ID 304. It is possible but not mandatory that each stored version is also separately identifiable by the associated user identifier 306, 307, or 308. An association to a user identifier of an i:th client means that the i:th stored encapsulated version of the shared secret is known to be subject to decapsulation with a private key, the corresponding paired public key of which is known to be a public key of the i:th client.

Above in the description of figs. 1 and 2 it was explained how the trust provider transmits its response 104 or 203, respectively, to client A. As also explained already, whether this response contains the encapsulated version 313 that is subject decapsulation with a private key held by client A depends on whether the shared secret 302 was generated by the trust provider (as in fig. 1) or by client A themselves (as in fig. 2). The same applies to the common identifier or DATA-ID 304. If the operation proceeded as in fig. 1, also the DATA-ID 304 was only generated for the first time by the trust provider and needs to be transmitted to client A in the response 104. If, however, the operation proceeded as in fig. 2, client A could have generated also the DATA-ID and transmitted it to the trust provider already in the first request 202. Alternatively, if the algorithm used to generate the DATA-ID is known to both client A and the trust provider, they can generate the same DATA-ID 304 independently. In these latter cases, the trust provider does not need to transmit the DATA-ID to client A in the response 203.

When the arrangement then receives the so-called second request 106 from one of the further clients, it may note that the originator of the request is identifiable as a holder of one of the further private keys. Here, the further private keys are those that the arrangement knows to enable their holders to decapsulate a corresponding encapsulated version of the shared secret. Advantageously, the arrangement requires the originator of each such second request to authenticate themselves using a strong and reliable authentication mechanism before responding to the respective second request. Each such second request should also refer to the common identifier or DATA-ID 304. Hence, the arrangement knows to respond to the second request by transmitting that of further versions 314 or 315 that is subject to decapsulation with the private key, the paired public key of which is known to the arrangement as being a public key of the respective one of said further clients.

Above it was pointed out that it is possible but not mandatory that each stored version 313, 314, and 315 of the encapsulated shared secret is also separately identifiable by the associated user identifier 306, 307, or 308. If the stored versions are not identifiable by the associated user identifier, the trust provider cannot know, which of them it should transmit to the originator of the second request. It is possible, however, that the trust provider simply transmits every version in its response to the second request. It remains then the responsibility of the client receiving the bunch of versions to try decapsulating each of them in turn with their appropriate private key. The decapsulation will only succeed in the case of the correct version.

As a special case, the arrangement may receive a request like the one called the second request above also from client A. It may happen, for example, that client A has somehow lost their previously obtained copy of the shared secret and must ask the trust provider to transmit it anew. The arrangement should respond to such an exceptional second request similarly as it responds to the actual second requests, i.e. by transmitting the first encapsulated version 313 that is subject to decapsulation with the private key, the paired public key of which is known to the arrangement as being a public key of client A. What was said above about the trust provider possibly transmitting all versions of the encapsulated shared secret in response to a second request, applies also here.

An obfuscator functionality 317 is shown in fig. 3 as an advantageous further feature of the arrangement. The arrangement may be configured to permanently obfuscate the plaintext form of the shared secret 302, as well as the ephemeral keys 309, 310, and 311 after using them for the generation of the encapsulated versions 313, 314, and 315, respectively. In such a case, even if the arrangement has the encapsulated versions 313, 314, and 315 stored as indicated with reference designator 316, it cannot even itself later find out the plaintext form of the shared secret 302. This provides an additional layer of security in case

the security measures of the trust provider's arrangement were later successfully penetrated by an unauthorized party who tried to gain access to the previously distributed shared secrets of clients. Yet, any of the clients among which the shared secret was originally distributed can download their dedicated encapsulated version of the shared secret also at any later moment, for decapsulating with their respective private key.

Fig. 4 shows another embodiment of the exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients. The parties shown are the same as in figs. 1 and 2, and the shared secret is generated by the trust provider like in fig. 1. As a difference to the embodiments of figs. 1 and 2, in fig. 4 the initiator of communications, i.e. client A, has already some data that they wish to be transmitted to client(s) B, (C, D,...) in a cryptographically protected way. When, how, or where such data actually originated is immaterial to the following description, but for the sake of example fig. 4 shows the generation of the data as step 401.

Similar to figs. 1 and 2, the arrangement of the trust provider receives a first request 402 from client A. The communications connection between client A and the trust provider is assumed to be cryptographically protected (irrespective of the actual method used) and the trust provider is assumed to have required client A to use a strong authentication method, so this initial transaction should involve a reasonable level of security against any unauthorized access to the data by third parties.

The generation of a shared secret by the trust provider at step 102 takes place in the same way as in fig. 1. At step 403, the trust provider uses the shared secret (or a further crypto-product derived therefrom) as an encryption key to encrypt the data it received from client A.

Client A may have also transmitted a common identifier called DATA-ID in the first request 402. Alternatively, the arrangement of the trust provider may have calculated a DATA-ID from the received data already before step 403. In any case, a common identifier DATA-ID may

be either generated or updated at step 404. As a non-limiting example, the DATA-ID formed at step 404 may be a hash or other crypto-product derived from the shared secret that the arrangement generated at step 102.

Encapsulating the shared secret at step 103 takes place in the same way as in figs. 1 and 2 (for details, see fig. 3 and its associated description). As a difference to figs. 1 to 3, the arrangement of the trust provider is configured to store also the encrypted data (and not only the encapsulated versions of the shared secret) in association with the common identifier DATA-ID. Thus, after the completion of steps 102 – 103 in fig. 4, the common identifier DATA-ID may be considered as a kind of pointer in the arrangement of the trust provider, pointing to a dedicated data storage where the encrypted data and the encapsulated versions of the shared secret are waiting to be distributed to the clients that are to communicate with each other. Indeed, the dedicated data storage is a part of trust provider architecture. The dedicated data storage of the trust provider can be used to store all or some of the encrypted data and the encapsulated versions of the shared secret. This way said data and encapsulated versions of the shared secrets can be arranged to be waiting to be distributed to the clients that are to start a communication session with each others.

The response 405 from the trust provider to client A contains at least an acknowledgement that the trust provider has completed steps 102 – 103, as well as the (updated) common identifier DATA-ID. It may also contain the first version of the encapsulated shared secret (i.e. the one subject to decapsulation with a private key of client A) and/or the encrypted data that the arrangement of the trust provider generated at step 403. Transmitting the encrypted data back to client A at this point is not necessary, as client A possesses the data already as a result of having generated it themselves at step 401. Also, transmitting the encapsulated shared secret to client A is only necessary if client A needs the

shared secret for some further use, like for such communications between client(s) B, (C, D,...) that are not shown in fig. 4.

Client A transmits an invitation 406 to client(s) B, (C, D,...). The invitation 406 contains at least the common identifier DATA-ID in the (updated) form in which client A received it from the trust provider in the first response 405. Equipped with knowledge of the common identifier DATA-ID, any of clients B, C, D,... who received the invitation 406 may then approach the trust provider with the so-called second request 407, referring to the common identifier DATA-ID. The arrangement of the trust provider is configured to respond to receiving the second request(s) 407 by transmitting the encrypted data (and the appropriate encapsulated version of the shared secret, like in figs. 1 and 2) in the response 408 to the client B, C, and/or D who sent the second request(s) 407. After successful decapsulation of the shared secret, the client who received the response 408 is able to decrypt the transmitted data at step 409.

Fig. 5 shows another embodiment of the exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients. Steps and actions with reference designators 401, 402, 102, 403, 404, and 103 are the same as in fig. 4. As a difference to fig. 4, the response 501 to client A now mandatorily contains also the encrypted data that the arrangement of the trust provider generated at step 403. When client A then transmits an invitation 502 to client(s) B, (C, D,...) it includes the encrypted data in the invitation. Consequently, the client(s) who received the invitation 502 need not download the encrypted data from the trust provider as in fig. 4. It suffices to transmit the so-called second request 503 to the trust provider to receive, in the response 504, the required version(s) of the encapsulated shared secret. After decapsulating the appropriate version, the client in question may decrypt, at step 409, the encrypted data it received in the invitation 502.

Fig. 6 shows another embodiment of the exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients. Steps and actions with reference designators 101, 102, 103, and 104 are the same as in fig. 1. Similar to figs. 4 and 5, also in fig. 6 it is assumed that client A wishes to securely transmit some data to client(s) B, (C, D,...). When, how, or where such data actually originated is again immaterial to the following description, but for the sake of example fig. 6 shows the generation of the data as step 601. At step 602, client A encrypts the data using the shared secret it received (as the appropriate encapsulated version) from the trust provider in the first response 104. Client A includes the encrypted data in the invitation 603 it transmits to client(s) B, (C, D,...). The actions of the recipient(s) of the invitation 603 in steps 106, 107, and 409 are the same as in figs. 1 and 4, respectively.

Fig. 7 shows another embodiment of the exchange of transmissions and execution of some operations in an arrangement for distributing a shared secret among clients. Steps and actions with reference designators 401, 402, 102, 403, 404, and 103 may be the same as in fig. 4, however with some possible differences. For example, the request 402 does not necessarily contain any identifiers of any other clients. Namely, one possible application of the invention is for individual clients to use the trust provider as a secure storage of data that they might need later but that they do not want to store in their own possession in the meantime, for example because they have no sufficiently secure form of data storage available in their own device(s). A non-limiting example of such data is an encryption key to be used in symmetric cryptography when communicating later with other clients, or a seed to be used in later re-generating such a key.

In conformity with that above, the data generated by client A at step 401 may already contain (also) the shared secret, like at step 201 of fig. 2 earlier. In such a case, step 102 is not needed and the encryption

of the data – if there is also other data than the shared secret itself – at step 403 take place using the shared secret that the trust provider received from client A at step 402.

As illustrated with the use of parentheses, steps 405, 406, 407, and 408, and 409 are optional in fig. 7. They may be included if the purpose is to eventually establish a situation where client A as well as at least one of clients B, C, D,… possess the same shared secret. They are not needed, however, if the purpose is only to ensure that client A can later retrieve data that they generated at step 401 and transmitted to the trust provider at step 402 but obfuscated (or simply lost) thereafter at step 701. Basically, the trust provider does not even need to respond to client A at step 405 if, at that stage, client A does not need any confirmation that the trust provider completed the required ones of steps 102, 403, 404, and 103 as intended. However, if the trust provider updated the so-called common identifier at step 404 (instead of using an identifier that would have come from client A at step 402), it may transmit the updated common identifier to client A in the response transmitted at step 405.

Notable is that if the trust provider is to act only as a secure data storage for client A, without any need to ever respond to any of clients B, C, D,… in association with this particular stored data, the encapsulation at step 103 only needs to take place using one ephemeral key. Comparing to fig. 3 earlier, there would be only the first user identifier 306, only the first ephemeral key 309, and only the first version 313 of the encapsulates shared secret.

Later, at step 702, client A may send to the trust provider a second request that may be quite like those "second" requests that other clients may transmit and that are shown with reference designator 407 in fig. 7. The trust provider will then note that the request of step 702 comes from the first client A, identifiable as a holder of the key that was called the first private key earlier in this text. The trust provider will also

note that the request refers to the so-called common identifier mentioned above. The trust provider will then respond by transmitting, at step 703, (both the encrypted data and) that version of the encapsulated version of the shared secret that is subject to decapsulation with the private key, the paired public key of which it knows to be a public key of the first client A. At step 704, client A decapsulates the encapsulated shared secret using their private key. If there was also other data that came encrypted in the response 703, client A decrypts the received encrypted data at step 704.

The methods and arrangements described above can advantageously be applied when one wishes to use communications and/or information distribution channels that already exist between users to also exchange confidential information. Such already existing channels include, but are not limited to, emails, instant messaging, VoIP (Voice over IP) communications, and packet data networks. The possibility of using previously existing channels is a significant advantage, as it mitigates any need to establish new infrastructure and brings secure digital communications within reach of a vast number of existing devices with only minimal needs to update existing software. Securely distributing shared secrets according to the foregoing description allows the communicating parties to maintain high levels of security against even the most advanced attacking attempts while requiring only moderate amounts of processing capacity. This makes the methods and arrangements described above readily applicable to, for example, IoT (Internet of Things) solutions where the nature and physical implementation of at least some of the devices involved may set strict limitations on critical resources such as power consumption, computing capacity, and/or available memory.

According to one embodiment, an implementation of so-called disposable devices (such as disposable phones, disposable tablets, disposable personal digital assistants, disposable wearable devices, disposable IoT devices, and similar) can be envisioned, for example. We may

assume that a user with a smartphone or some other kind of digital communications device wants to use their device repeatedly for an application that requires a high level of digital security, like making encrypted phone calls. Yet, between calls the device should be wiped clean of any traces that such phone calls ever took place. One or more of the methods explained above may be used to generate and/or distribute a shared secret for use only for the exact duration of a call. For example, the setup phase of the call may involve the device automatically transmitting the first request 101 of Fig. 1 or the first request 202 of Fig. 2, so that before the call actually proceeds, a shared secret has been generated and delivered to not only the device in question but also to a device of the intended recipient of the call. The shared secret can then be used for cryptographic purposes, like encrypting transmitted communications and decrypting received communications between the parties of the call. If the setup phase of the call involved the device transmitting the second request 106 of any of Figs. 1 or 2, for example if the intended recipient of the call should not be bothered to download their version of the shared secret a new, the initiating device just downloads a previously generated shared secret, and the call may then proceed using the previously generated shared secret for the cryptographic purposes. When the call ends, the device may use a built-in obfuscator functionality to permanently obfuscate all traces that the call took place, including the shared secret and any possible cryptographic derivatives thereof.

Such applications of cryptography are enabled by the aspects of the present disclosure, which will now be specifically discussed in detail. Fig. 8 illustrates an arrangement 800 for enabling a group communication. The arrangement 800 comprises at least one trust provider (depicted as a trust provider 802) and a plurality of clients (depicted, for example, as clients 804, 806, and 808) that are communicably coupled to each other and to the trust provider 802.

It will be appreciated that the at least one trust provider could be implemented as a single trust provider (such as the trust provider 802) or a plurality of trust providers that serve as linked vaults, namely trust providers. It should be noted that the trust provider may also be called a vault, emphasizing the assumption that it represents an institution with exceptionally high level of digital security. In this regard, the plurality of clients need not necessarily be linked to a same trust provider. Furthermore, the plurality of clients can include N clients, where N is a positive integer equal to or larger than 2. Let us consider, for example only, that the client 804 initiates the group communication and is thus a first client (or the initiator client).

The "*group communication*" involves 2 or more members (i.e., 2 or more clients). In an embodiment, the group communication is one of: a group audio call, a group video call, a group chat, a group email conversation, a group workspace communication, a group conference call, a group broadcast, a group network, a packet data session. Such various types of group communications can be securely enabled using the arrangement 800. It will be appreciated that one or more sessions of the group communication may be enabled between the same clients. The arrangement 800 provides security in such group communications, to ensure privacy, protect sensitive information, and maintain trust amongst the plurality of clients 804, 806, and 808. A technical effect of these features is to provide a secure, user authorization-controlled, and client-specific mechanism for enabling the group communication, wherein each stage of the process contributes to enhanced confidentiality, integrity, and access control.

The trust provider 802 is configured to: receive a request for establishing a session of the group communication, from the first client 804 amongst the clients 804, 806, and 808; generate (at the step 102) a shared secret corresponding to the session, by executing a first random number generator 810; digitally encapsulate (at the step 103) the shared

secret in a plurality of versions (such as the encapsulated versions 313, 314, and 315), using respective ephemeral keys (such as the ephemeral keys 309, 310, and 311) corresponding to the clients 804, 806, and 808, the respective ephemeral keys being available with the trust provider 802; and distribute a respective version amongst the plurality of versions 313, 314, and 315 of the shared secret, to a corresponding client amongst the clients 804, 806, and 808, prior to a start of the session. In this regard, for example, the version 313 may be distributed to the client 804, the version 314 may be distributed to the client 806, and the version 315 may be distributed to the client 808. Notably, each client receives a single respective version of the shared secret, in the encapsulated form.

In the above regard, the processing steps performed by the trust provider 802 relate primarily to establishing the group communication in a secure manner. The distribution of shared secret is essential for enabling each client 804, 806, and 808 to join and participate in the group communication, and the digital encapsulation of the shared secret ensures that the secret is properly secured before being shared, in a manner that only the intended client is able to decapsulate it, thus providing security when establishing the group communication. The shared randomness (i.e., the shared secret) is strongly protected and verified, ensuring prevention of their manipulation or misuse. The processing steps performed by the trust provider 802 have been discussed in detail earlier in the description.

The request received from the first client 804 may be the first request 101, the first request 202, or the first request 402. The first random number generator (RNG) 810 may be the hardened true random generator subsystem (TRNG) 301, or may be a Hardware random number generator (HRNG). The first RNG 810 serves as a first phase of a Deterministic Trust Random Number Generator (DTRNG), which is a two-phase hybrid RNG. Herein, the hardened TRNG refers to a specialized hardware or cryptographic component configured to generate truly unpredictable

numbers from a physical entropy source, with enhanced security features that protect against both external attacks and internal failures or manipulation. The HRNG represents a hardware-based implementation of a random number generator that similarly produces random values derived from a physical entropy source, though without necessarily incorporating additional protective or redundancy mechanisms found in the hardened TRNG. Both the hardened TRNG and the HRNG are, by definition according to the present disclosure, hardware-based random number generators relying on physical entropy sources for generating high-entropy shared secrets. It will be appreciated that the use of such hardware-based entropy sources ensures reliable, tamper-resistant, and manipulation-resistant generation of shared secrets required for secure group communication. A distributed architecture of the plurality of trust providers is implementable if the first RNG 810 can be guaranteed to operate in a trusted environment.

In the arrangement 800, the shared secret has high entropy, which ensures reliability, protection against manipulation, provides resistance to physical attacks and replay attacks, resists malicious clients, protects against external disturbances and environmental changes by securely generating high-quality entropy in a trusted entropy service (i.e., the trust provider). This high-quality entropy can be short or long term and is sent as encrypted to the clients, ensuring that both data authentication and high-entropy's trustworthiness and integrity are maintained. The first random number generator can be implemented as any hardware-based random number generator configured to produce the shared secret with high entropy. The use of such a hardware-based random number generator ensures that the generated shared secret exhibits a high degree of unpredictability and statistical randomness derived from physical entropy sources. It will be appreciated that the shared secret with high-entropy enhances overall security and robustness of the group

communication by ensuring that cryptographic materials generated by the trust provider and utilized by the clients remain secure.

Each client 804, 806, and 808 is configured to: decapsulate the respective version of the shared secret, using a respective private key; obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is used to provide a salt component; derive one or more crypto-products, by executing a second random number generator 812, 814, 816, respectively, wherein the shared secret and the salt component are used as inputs to the second random number generator 812, 814, and 816; and employ the one or more crypto-products to encrypt outgoing communication from said client, and decrypt incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients. In this regard, for example, the client 804 derives the one or more crypto-products by executing the second RNG 812, the client 806 derives the one or more crypto-products by executing the second RNG 814, and the client 808 derives the one or more crypto-products by executing the second RNG 816. It will be appreciated that this configuration enables secure and authorization-bound utilization of the shared secret, thereby ensuring that decryption and subsequent key derivation can only be performed by verified clients while maintaining consistent and synchronized keying material across the plurality of clients participating in the group communication.

The arrangement further ensures a precise cryptographic correspondence between the keys used for encapsulation and decapsulation operations. In particular, the private key maintained at each client 804, 806, and 808 corresponds to the ephemeral key used by the trust provider when encapsulating the respective version of the shared secret for that client. This correspondence forms a one-to-one mapping between the encapsulation and decapsulation keys, thereby ensuring that only the intended client possessing the matching private key can successfully

decapsulate its version of the shared secret. This correspondence provides a fundamental security mechanism that prevents any unauthorized entity from accessing the distributed shared secret.

The identification credential is used, in providing a salt component, as an access/controlling means to get the salt from the trust provider or from the party which initiated the communication session, or it can be formulated by trust provider using proprietary algorithms. Herein, the "*proprietary algorithms*" refers to deterministic computational processes implemented within the trust provider for securely generating or formulating the salt component. The proprietary algorithms operate on parameters such as identification credentials, session identifiers, and entropy values generated by the first random number generator, to produce a consistent salt component that is identical for each authorized client participating in a given communication session. The proprietary nature of these algorithms resides in their internal implementation and controlled execution within the trust provider, ensuring authorization-bound, reproducible, and confidential generation of the salt component.

In other words, the proprietary algorithms could refer to non-standard public algorithms, internally implemented computational processes executed by the trust provider for securely generating or formulating the salt component. The proprietary algorithms differ from publicly standardized cryptographic algorithms, for example, such as SHA-256, HKDF, or crystals-Kyber, in that said proprietary algorithms are custom-designed to fulfill specific design requirements, including data minimization, reproducible identity binding, client uniqueness or session uniqueness, and enhancement of entropy through multi-attribute input. In the present disclosure, such proprietary algorithms can operate deterministically to derive or regenerate a consistent salt component based on one or more parameters associated with the communication session. Herein, the one or more parameters may include, but are not limited to, identification credentials, user authentication inputs (such as a personal

identification number (PIN), and the like), Vault-issued identity credentials (such as a User Registration Token (URT), a User Authentication Token (UAT), and the like), contextual session identifiers (such as a User Identifier (UID), a Transaction Identifier (TRID) and the like), and cryptographic material (such as a Vault Archive Key (VAK) or a User Secret Seed (USS)). It will be appreciated that the purpose of these proprietary algorithms is to ensure that multiple authorized clients (or sessions) deriving the same salt component from the trust provider obtain an identical and consistent value, thereby enabling deterministic generation of symmetric session keys generation, zero-knowledge authentication tokens, or other crypto-products required for group communication. This configuration further allows the trust provider to securely provide reproducible salt components without exposing the underlying derivation inputs or internal logic, thereby maintaining confidentiality, minimizing data disclosure, and preserving secure reuse of the salt component across authorized sessions. Accordingly, the proprietary algorithms implemented by the trust provider can be regarded as internal, authorization-bound derivation mechanisms that ensure deterministic identity or session binding in a secure and privacy-preserving manner.

This way same salt component can be securely provided to each client 804, 806, and 808 amongst the plurality of clients. To facilitate secure and consistent generation of the one or more crypto-products among all clients, the arrangement employs an identification credential associated with each authorized user. Herein, the identification credential acts as an access-control parameter that enables retrieval or formulation of the salt component, i.e. authorization-tied salt. The salt component could be obtained from the trust provider, from the initiating party of the communication session, or can be formulated by the trust provider using one or more internal or proprietary algorithms, resulting in the same salt component being available to every authorized client participating in the group communication. In this way, all clients receive an identical salt

component even though the retrieval process is authorization-bound through each client's unique identification credential. The use of the identification credential as an access mechanism ensures that the credential itself does not directly alter the salt value but only governs its acquisition from a centralized and trusted source. When each client employs its identification credential to obtain the same salt component and combines it with the identical shared secret within the second random number generator, the derivation of crypto-products becomes deterministic and uniform across clients. This configuration eliminates the need for any further negotiation or exchange of keying material between clients during session establishment, since all clients generate identical crypto-products locally.

Moreover, in the present disclosure, the salt component functions as a cryptographic component that is explicitly tied to a user's authorization to utilize the shared secret. The identification credential provides access to the salt component, thereby binding the generation and use of derived cryptographic products to verified authorization. This configuration ensures that the shared secret and the resulting crypto-products can only be accessed and employed by authorized clients, thereby enhancing security through explicit access control and preventing unauthorized derivation or usage of the shared secret and the crypto-products.

In the above regard, the processing steps performed by each client 804, 806, and 808 relate primarily to maintaining the group communication in a secure manner. These processing steps will now be discussed in more detail.

The respective private key (which is used for decapsulation of the respective version of the shared secret, at each client), corresponds to the respective ephemeral key which is employed for encapsulating the shared secret for said client. This ensures that only an intended client is able to decapsulate the shared secret. In an embodiment, upon decapsulation of the respective version of the shared secret at each client 804, 806, and 808, the shared secret is extracted into a programmatically

protected memory, the programmatically protected memory being one of: a segregated memory provided by Hardware Security Module (HSM), a secure enclave, a Trusted Platform Module (TPM), a secure element, a guarded heap memory. This programmatically protected memory works in a memory area that protects the high-entropy randomness (i.e., the shared secret) from being exposed to various attacks and buffer overflows during execution. The programmatically protected memory enables safe extraction and storage of the shared secret, at each client 804, 806, and 808. Optionally, the shared secret is retrieved from the programmatically protected memory, when the user provides the identification credential to the client device. In this way, the shared secret is prevented from being accessed and/or used without the user's explicit consent, giving the user full control over the process of using the shared secret. This solution provides a new layer of security, which is not present in conventional systems. Such provision of the identification credential by the user signifies the user's explicit authorization (i.e., consent) to use the shared secret.

In an embodiment, the identification credential is at each client 804, 806, and 808 an input provided by a corresponding user of said client. Optionally, the shared secret is used at any client 804, 806, and 808 only after the identification credential is obtained. Providing the identification credential is a way of enabling users to authorize usage of the second RNGs for enabling controlled generation of the one or more crypto-products at runtime, which can prevent unauthorized entities from exploiting the second RNGs for manipulation purposes. The requirement for the identification credential for using the shared secret prevents deterministic random number generator attack reproduction attempts. The identification credential protects against modeling (backtracking resistance) and forward prediction (prediction resistance), to provide an additional layer of security in enabling the group communication. Therefore, even though the shared secret is used outside of the trust provider

802, security in its usage is ensured reliably. Moreover, the presence of the identification credential enables controlled activation of the second random number generator, which is responsible for generating one or more crypto-products at runtime. This control mechanism protects against exploitation of the second random number generator by unauthorized entities. The use of the identification credential for authorizing the generation of crypto-products also improves resilience against deterministic random number generator attacks, thereby providing both backtracking resistance and prediction resistance. These measures collectively ensure that the group communication remains secure even in dynamic or partially trusted environments.

Herein, each client 804, 806, and 808 employs the derived crypto-products for encrypting outgoing data and decrypting incoming data exchanged with other clients in the group. The shared secret and the salt component serve as common deterministic inputs to the second random number generator at every authorized client, ensuring that identical crypto-products (such as encryption keys or authentication codes) are produced for all clients within same communication session. Consequently, each client can perform encryption and decryption of communication data without any additional negotiations or exchange of keying once the session has been initialized. This deterministic derivation process maintains synchronization of encryption parameters and preserves end-to-end confidentiality and integrity.

In an embodiment, the identification credential can be for example one or more of: personal secret-based authentication, identity card-based authentication, cryptographic authentication, wireless communication-based authentication, token-based authentication, notification-based authentication, user identifier, code-based authentication, digital identity-based authentication, biometric and/or neural authentication, anti-spoofing authentication. The personal secret-based authentication may be in the form of a password, a PIN code (for example personal

secret), a pattern, biometric data, private key, recovery phrase, one-time password (OTP), security key, two-factor authentication (2FA), certificate, an application password, or similar. The identity card-based authentication may utilize identity cards readable via NFC technology, radio technology or similar, such as identity cards, passport chips, smartcards, or similar for convenient and high-security authorization. The cryptographic authentication may include Fast Identity Online (FIDO) standards authentication. The wireless communication-based authentication may, for example, be Bluetooth Low Energy (BLE)-based, Wi-Fi-based, or similar. It provides an additional layer of security, particularly for IoT and mobile devices. The token-based authentication may, for example, be USB security key-based. The notification-based authentication may, for example, employ push notifications. The code-based authentication may, for example, employ Quick Response (QR) codes. The digital identity-based authentication may be based on digital wallets, digital profiles, or similar. The biometric and/or neural authentication may enable authorization for example by fingerprint, facial or iris recognition or through a brain-computer interface, where the user's thoughts or neurological signals act as a personal secret. The anti-spoofing authentication may employ liveness detection, based on video and audio, for enhancing security by ensuring the user is physically present and alive during the authentication process.

The one or more crypto-products are derived at each client 804, 806, and 808, by executing the second RNG 812, 814, and 816 corresponding to said client. The second RNGs 812, 814, and 816 serve as a second phase of the DTRNG, which is performed in untrusted environment to ensure the trustworthiness of randomness by repeating the high-quality randomness generated from the first phase using user authorization. Revealing the shared secret does not compromise the derived randomness without an additional secret, the salt, being provided by the user. The salt is obtained using the user’s identification credential, before

using the second RNGs 812, 814, and 816. Each of the second RNGs 812, 814, and 816 may, for example, be a Pseudo random number generator (PRNG), a Deterministic Random Bit Generator (DRBG), or similar. The second RNG exploits trustworthy deterministic randomness, by using the shared secret. The second RNGs 812, 814, and 816 are optionally implemented as a set of one or more modules. Optionally, each module of the set is specialized for generating specific types of random values, corresponding to specific types of crypto-products. This modular approach allows for providing flexibility, security, and performance optimization, in cryptographic systems such as the arrangement 800 for enabling the group communication.

In the DTRNG, two opposite RNG components are combined, meaning that the first RNG 810 generates the shared secret (using a trusted service such as the trust provider 802), which is used deterministically in the second RNGs 812, 814, and 816. The first RNG 810 and the second RNGs 812, 814, and 816 meet the requirements set for randomness generators, for example, according to standards such as FIPS, NIST, ENISA, or similar. The DTRNG provides deterministically reproducible numbers that appear as if they are randomly generated. This is achieved by using the first RNG 810 as the source of true randomness to generate high entropy shared secret, which is then utilized in deterministic algorithms of the second RNGs 812, 814, and 816 to produce pseudorandom numbers deterministically in a controlled environment. The DTRNG provides user-specific random numbers which can be used widely. The DTRNG employs centralized security for decentralized use (for example, in European Digital Identity (EUDI) solutions, decentralized identity (DID) solutions, self-sovereign identity (SSI) solutions, or similar, for self-sufficient identity management). Such identity solutions benefit from the use of distributed randomness, especially in key generation, DID identifiers creation, verifiable credentials management, zero-knowledge proofs solutions and secure communication, because they can utilize

reliable shared sources of randomness to ensure the security and reliability of an identity system. The decentralized or distributed use enables cost-effective distributed service architecture solutions, wherever feasible technologically. The DTRNG has various applications such as cryptography, AI systems, communication protocols, authentication and identity management, information security systems, lotteries and gaming applications, arts, simulations, and scientific experiments, IoT and closed systems, cloud and distribution systems, and financial applications. In addition, the DTRNG method can be additionally applied in 3GPP 5G & LTE solutions, since distributed randomness plays an important role in cryptographic operations such as key exchange, authentication and encryption, the security of which largely depends on the quality of randomness. The method enables distributing encryption keys for each participants in the group call, e.g. using SRTP-DTLS protocol for VoIP.

In an embodiment, the second random number generator 812, 814, and 816 employs a Deterministic Random Bit Generator (DRBG) algorithm when deriving the one or more crypto-products, the DRBG algorithm being one of: a counter mode DRBG (CTR-DRBG) algorithm, a Hash-based Message Authentication Code (HMAC)-based DRBG algorithm, a HASH-DRBG algorithm. These DRBG algorithms use a key derivation function to combine the shared secret (i.e., seed) and the salt component (which is obtained or provided by utilizing respective user's identification credential) in a way that preserves the randomness of the seed and ensures that the final result meets the requirements of standards. These DRBG algorithms can maintain quantum durability with a security level of 128-bits. This means that as the seed comes from a trusted source (i.e., the trust provider 802), the initial state of the DRBG is as random as possible, due to which these methods enable 128-bit security against quantum attacks.

According to one embodiment, the DRBG algorithm is the CTR-DRBG algorithm. This implementation is based on a symmetric block

cipher, which is used in counter mode. The symmetric block cipher may, for example, be an AES block cipher, which encrypts consecutive counters and produces random bit strings. The CTR-DRBG algorithm is very secure if the security of the block cipher is reliable. The CTR-DRBG algorithm generates randomness quickly and is easily scalable, and can be used in performance-critical applications that require high performance, such as cryptographic libraries and cryptographic tools.

According to another embodiment, the DRBG algorithm is the HMAC-based DRBG algorithm. In this implementation, the HMAC-based DRBG algorithm is used to generate deterministic random data using HMAC. HMAC is a combination of a hash function (for example, such as SHA-256) and a symmetric key. The HMAC-based DRBG algorithm is also very secure, especially in applications where key-protected deterministic randomness is required. HMAC is resistant to various attacks because it combines the hash function and the secret of the key. The HMAC-based DRBG algorithm can be used, for example in software-based implementations such as cryptographic libraries, where a simple and secure random number generator is needed and is suitable for deterministic generation of encryption keys and cryptographic identifiers.

According to yet another embodiment, the DRBG algorithm is the HASH-DRBG algorithm. This implementation directly uses a cryptographic hash function (for example, SHA-256) to generate random bit strings. The shared secret and the provided salt component are processed through a hash function that produces randomness. The HASH-DRBG algorithm is very secure, especially if the hash function used is secure and unknown to attacks. However, the HASH-DRBG algorithm is slightly simpler than the HMAC-based DRBG algorithm because it does not contain key material and is used for applications where simplicity and easy implementation are important. For this reason, it is suitable for situations where a deterministic random number is needed without the need for a symmetric key (e.g. testing and validation).

Pseudocodes for each of the aforementioned DRBG algorithms are provided in the APPENDIX herein later. The DTRNG described herein is based on symmetric cryptography and can be used far into the future by updating the hash algorithms. For example, the SHA3 algorithm can already be used instead of the SHA2 algorithm, which can be used to increase the 256-bit output to 512-bit.

In an embodiment, the one or more crypto-products comprise at least one of: a symmetric key, an asymmetric key pair, one or more keys for encrypting and/or wrapping other key(s), a nonce value or initialization vector, a secure key from an identification credential, a secure password, a derivative of the secure password, a session key, a one-time key, a pre-shared key (PSK) of a Transport Layer Security (TLS)-PSK model. Such crypto-products can be utilized in one or more cryptographic applications. Pseudocodes for generating the one or more crypto-products are provided in the APPENDIX herein later.

The asymmetric key pair comprises a public key and a private key. Optionally, the asymmetric key pair may be generated using a Rivest-Shamir-Adleman (RSA) algorithm, an Elliptic Curve Digital Signature Algorithm (ECDSA), a Curve25519 algorithm, an Ed25519 algorithm, a Kyber1024 algorithm, a Dilithium5 algorithm, or similar. The one or more keys for encrypting and/or wrapping other key(s) enable secure key sharing. The nonce value or initialization vector may be needed in cryptographic algorithms (for example, such as AES-GCM, ChaCha20 or AES-CBC. The secure password or its derivative may be used in cryptographic applications such as password managers. The session key may be used to create secure sessions, using cryptographic protocols (for example, such as TLS). Optionally, the one-time key is a part of a secure messaging protocol. The secure messaging protocol may, for example, be a Signal Protocol, a Matrix Protocol, or similar.

The PSK for the TLS-PSK model provides several security advantages, and it mitigates certain vulnerabilities compared to a traditional

certificate-based TLS model. As a first example, the traditional certificate-based TLS model relies on asymmetric cryptography, which is vulnerable to quantum computer attacks. Using the PSK for the TLS-PSK model ensures quantum resilience by generating a sufficiently large encryption key through a one-way hash from the shared secret. As a second example, the traditional certificate-based TLS model places trust in certificate authorities (CAs) that issue certificates. If any certificate authority is compromised or a client accepts an invalid certificate, an attacker can create unauthorized certificates, breaking the trust chain e.g., through MitM (Man in the middle) attacks with certificates. As an example of such attacks, SSL strip downgrades HTTPS connections to HTTP, enabling attackers to intercept and manipulate traffic while bypassing secure certificate validation (due to the shared secret being insecurely protected). As a third example, the PSK is provided based directly on trust relationship between the at least one trust provider and clients, so there is no need of relying on third-party certificate services to verify revoking or compromise of certificates. As a fourth example, the traditional certificate-based TLS model relies on certificate stores on local systems to maintain accepted certificates and CA roots. If a certificate store is compromised, an attacker can add malicious CAs or invalid certificates. the TLS-PSK model does not use certificate stores, eliminating this attack vector. As a fifth example, the TLS-PSK model avoids certificate exchange or verification (as it does not utilize the CA infrastructure), thereby simplifying and speeding up connection process of the group communication.

Using the method pursuant to the disclosure, the PSK for the TLS-PSK model can be leveraged to enable a new layer of protection for group communications over the internet, achieving so called “Internet 2.0”. For example, protocols for such communication may be strengthened for providing a quantum-resilient solution while ensuring default mutual authentication. For example, such protocols may be HTTP Secure (HTTPS), Transport Layer Security (TLS), Secure Shell (SSH), File

Transfer Protocol Secure (FTPS), Simple Mail Transfer Protocol Secure (SMTPS), Internet Message Access Protocol Secure (IMAPS), Post Office Protocol Secure (POP3S), Lightweight Directory Access Protocol Secure (LDAPS), Message Queue Telemetry Transport Secure (MQTTS), and similar.

The TLS-PSK model enables real-time authentication of clients, resembling an Identity and Access Management (IAM) model. This is because interaction with the at least one trust provider always requires strong user authentication, ensuring the confidentiality of clients 804, 806, and 808 in the arrangement 800. For example, retrieving the shared secret always requires strong user authentication and verification based on encryption keys and signed attributes (i.e., strong digital identifiers). As a result, clients can trust that the usage of the shared seed - namely the secret - is limited to authorized clients only. Moreover, the at least one trust provider 802 can utilize strong electronic identification compliant with requisite standards (such as the eIDAS standard). The implementation of the at least one trust provider 802 can be based on a Public Key Authentication (PKA) model, where the user's digital identity is verified as belonging to the user logging into a trusted service.

The one or more crypto-products are employed to secure communications between the plurality of clients 804, 806, and 808. Each client 804, 806, and 808 secures the outgoing communication therefrom, by encrypting said communication using the one or more crypto-products. Furthermore, each client 804, 806, and 808 receives incoming communication in a secure encrypted form from at least one other client who is a sender of said communication, and decrypts the incoming communication using the one or more crypto-products. In this way, secure communications are exchanged between the plurality of clients 804, 806, and 808 engaging in the group communication.

It will be appreciated that the present disclosure provides a novel architecture that employs corresponding ephemeral–private key

pairs in conjunction with centralized distribution of the salt component, thereby enabling secure encapsulation of the shared secret and deterministic generation of consistent crypto-products across the plurality of clients. In the arrangement of the present disclosure, session-fresh secret material (i.e., the shared secret) is generated by the at least one trust provider's first random number generator rather than reusing a static group key. The shared secret is encapsulated per client using a Key Encapsulation Mechanism, producing client-specific versions. Yet further, the arrangement also incorporates an authorization-bound salt component (derived from URT/UAT identification credentials) as an input to the client-side second RNG/KDF and implements a dual-RNG workflow to produce layered crypto-products.

Fig. 9 shows an example of how each client 804, 806, and 808 may use a shared secret 902 to derive one or more crypto-products 920, 930 and 940, in this example for use in symmetric encryption in group communications with at least one other client. The shared secret 902 may be a shared secret that came to the knowledge of each client 804, 806, and 808 as a result of executing the processes that are described elsewhere in this text. However, it should be noted that Fig. 9 is equally applicable in any case where a shared secret exists at the disposal of each client 804, 806, and 808.

In an embodiment, when deriving the one or more crypto-products 920, 930 and 940, the shared secret 902 is used as a seed component. Each client 804, 806, and 808 is configured to combine the seed component and the salt component 904 using a first Key Derivation Function (KDF) 906 of the second random number generator 812, 814, and 816, respectively, to derive a Pseudo-Random Key (PRK) 920. Examples of functions that may be used in the first KDF 906 include, but are not limited to, the known PBKDF2 (Password-Based Key Derivation Function 2) and HKDF (HMAC-based Key Derivation Function) functions. Such key derivation functions ensure that the shared secret 902 and the

salt component 904 are combined securely, and that the PRK 920 is resistant to attacks. The output of the first KDF 906 is the PRK 920, which is unique combination of the shared secret 902 and the salt component 904. Repeatedly using same inputs, such as the same shared secret 902 and the same salt component 904, will always produce the same PRK 920 for group. However, the PRK 920 is unique for each session.

If the salt 403 is used in the process, and if the key (and/or other crypto-product) that is to be derived with the method of fig. 3 should yield the same result for a known group of two or more clients, like a key for use in symmetric encryption for example, the clients in the group must have a way of sharing the salt 403 among them. If the method of fig. 4 is executed by client(s) B, (C, D,…) as shown in figs. 1 and 2, one possible way of sharing the salt involves client A transmitting the salt to client(s) B, (C, D,…) in encrypted form in the invitation 105. Another possibility is that the salt is included in what client(s) B, (C, D,…) get from the trust provider in the response 107. Further examples of conveying a salt and/or other data from client A to client(s) B, (C, D,…) are described later in this text.

Each client 804, 806, and 808 is configured to hash the PRK 920 using a nonce derivation function 908, to derive a common nonce 930. In the process shown in fig. 9, the PRK 920 is not used as such as an encryption or decryption key. It is an intermediate crypto-product that is used as an input to the nonce derivation function 908 (an output of which is the common nonce 930) as well as to a second key derivation function 910. The common nonce 930 may be derived by using SHA-256 hashing, for example. The common nonce 930 ensures every encryption operation, for example, in AES-CTR mode, is unique and unpredictable. The common nonce 930 could also be called the initial nonce, as an advantageous way of using it is to combine it with a streamNonce value during actual use. The advantage of such practice is that it makes each encryption operation secure and unpredictable.

Each client 804, 806, and 808 is configured to combine the PRK 920 and contextual information 912 associated with encryption, using the second KDF 910 of the second random number generator 812, 814, and 816, respectively, to derive an encryption key 940. In addition to the PRK 920, another possible input to the second key derivation function 910 is the contextual information 912 (which is so-called Keyinfo). If used, Keyinfo may contain contextual information associated with the encryption, like a message identifier and/or information about the type of encoded information (audio, video, control, etc.). The contextual information may enable any client to differentiate between different encryption sessions/different sessions of the group communication. The output of the second key derivation function 910 is the encryption key 940 that the client may then use for selected cryptographic operations. For example, the PRK 920 is expanded using the second KDF 910, to produce the encryption key 940. The encryption key 940 and the common nonce 930 are utilised for performing the encryption 916 of the outgoing communication and the decryption 918 of the incoming communication, as depicted by block 914.

In an embodiment, the encryption key 940 is dynamically re-computable without requiring its storage at said client. An important advantage of using a method like that in fig. 9 is that the encryption key 940 is derived dynamically and deterministically, hence, it does not need to remain stored in the memory of any client 804, 806, and 808 any longer than what is needed for an individual session of communications (or other kind of utilization of the encryption key 940). The encryption key 940 can thus be re-computed deterministically, as required, using the identification credential. This simplifies security requirements related to key management. By avoiding the storage of the encryption key 940, the risk of exposure, whether from physical attacks or malware on the client, is reduced. This capability to re-compute keys dynamically enhances both the flexibility and security of the provided solution. It will be

appreciated that this configuration minimizes vulnerability associated with key storage by enabling transient use of dynamically derived encryption keys, thereby reducing attack surface and improving overall security and robustness of the group communication.

In an embodiment, each client 804, 806, and 808 is further configured to: receive a start value (nonceStart) for stream nonce; and retrieve a group size value and a member index of said client, the group size value being equal to a number of clients in the plurality of clients 804, 806, and 808,

wherein when encrypting 916 the outgoing communication, said client is configured to: determine a stream sequence value for each data packet, based on the start value for stream nonce, the group size value, the member index, and a packet sequence number; define a stream nonce, based on the common nonce 930 and the stream sequence value; create a cipher block with the encryption key; create an outgoing keystream with the cipher block and the stream nonce; combine each data packet with the outgoing keystream, using an exclusive OR (XOR) operation, to generate encrypted data packets; and add the stream sequence value for each encrypted data packet into a header of said encrypted data packet, prior to transmitting said encrypted data packet.

In this regard, the start value for stream nonce is received when the group communication is established. The group size value and the member index of said client, may also be then received at said client. For example, when 3 clients 804, 806, and 808 participate in the group communication, the group size value is 3. The group size value may be received from an invitation to join to group communication, the invitation being shared by the trust provider 802 or the first client 804. The stream sequence value for each data packet to be sent by a client is built to be unique, as it is based on the member index (which is unique for each client) and on the packet sequence number (which is also unique for each packet). The XOR operation is computationally efficient and reversible,

which allows it to beneficially be used for both encryption and decryption. Furthermore, the outgoing keystream is created using the encryption key 940, wherein the encryption key 940 serves as an input to a cryptographic algorithm or stream cipher which produces the outgoing keystream. XORing the outgoing keystream with each data packet (i.e., plaintext) produces the encrypted data packets (i.e., ciphertext).

In an embodiment, when decrypting 918 the incoming communication, said client 804, 806, and 808 is configured to: obtain the stream sequence value, from the header of each encrypted data packet that is received at said client; define the stream nonce, based on the common nonce 930 and the stream sequence value; create a plaintext block with the encryption key 940; create an incoming keystream with the plaintext block and the stream nonce, the incoming keystream being same as the outgoing keystream; and combine each encrypted data packet with the incoming keystream, using the XOR operation, to generate decrypted data packets.

In this regard, the decryption happens correspondingly to the encryption. Upon decryption, the data packets received from the at least one other client are readable at said client. The use of symmetric cryptography and the shared secret provides a quantum-resilient solution. Symmetric cryptography removes attack vectors associated with key exchange. In particular, the DTRNG provides a secure way to use symmetric cryptography, by employing the shared secret and the salt component for providing security and user-control, thus eliminating the need for asymmetric key exchange protocols, which are particularly vulnerable in the quantum computing era.

The term "data packet" refers to a chunk of data which can be transmitted as a single unit, for transmission over a communication network. The plurality of clients may exchange data packets with each other during the group communication. A given data packet can include one or more types of data. Some examples of data packets are audio packets,

video packets, text/message packets, file packets, IoT device/sensor data packets, gaming packets, health data, transaction related data, tokens and the like.

As an example, in fig. 9 the client may operate in an AES-CTR (Advanced Encryption Standard – CounTeR) mode for communications as shown with block 914. The encryption key 940, the (initial) common nonce 930, and the possible streamNonce value derived from the above-mentioned encryption process as part of the AES-CTR mode is used for encryption 916 of plaintext information that is to be transmitted as ciphertext. The key 940 is also used for decryption 918 of received ciphertext into plaintext information.

In pseudocode phrases, the operations shown in fig. 9 may be described as follows:

block 920: PRK = First KDF(Salt component, PSS)

where PSS means Pre-Shared Seed and is equal to the shared secret 902 in fig. 9.

block 930: nonce(common) = head(16).sha256(PRK)

block 940: EncryptionKey = Second KDF(PRK,Keyinfo)

where KeySize = 32 bytes.

block 916:

define unique streamSequence based on nonceStart, group size, member index, and packet sequence number

streamNonce = head(8),nonce(common)|streamSequence

AES_block = cipher(EncryptionKey)

AES_CTR_keystream = cipher(AES_block, streamNonce)

encrypt stream packet with AES_CTR_keystream (XOR)

encryptedpacket.header["streamSequence"] = streamSequence.

block 918:

take streamSequence from received encryptedpacket.header

streamNonce = head(8),nonce(common)|streamSequence

AES_block = cipher(EncryptionKey)

AES_CTR_keystream = cipher(AES_block, streamNonce)

decrypt streamed data packets with AES_CTR_keystream (XOR).

The nonceStart value is an initialization value which the communicating parties may have received from the trust provider, for example. Alternatively, the communicating parties may preparatorily agree upon the nonceStart value using some other method and/or channel, as will be described in more detail later in this text. Member index is a value related to the number and order of members in the group of clients that are to communicate with each other: each client (i.e., a member of the group communication) 804, 806, and 808 has a unique member index corresponding to them in the list of members of the group communication. The member indices may be defined by the first client 804 who is the initiator of the group communication, at the time when the first client 804 sends the first request for establishing the group communication to the trust provider 802, or by the trust provider 802 at the time when it generates the client-specific encapsulated versions of the shared secret. Additionally, or alternatively, the members 804, 806, and 808 of the group communication may know their respective member index from other sources, like from previous group communications that have been practiced among the same members of the group, or from a data repository where member indices are pre-stored, or similar. A value streamSequence is an integer calculated as a sum of nonceStart + member_index + n*(group size value), where n=0,1,2,… is increased by one for each packet in the stream. As a consequence, every member has always a different streamSequence value for each packet. The value streamNonce consist of the first 8 bytes of nonce(common)||streamSequence, making streamNonce always unique. Specifically, the first 8 bytes of the common nonce 930 are concatenated with the stream sequence value. The stream nonce is unique for each client, as it is based on stream sequence value which is determined based on client-specific data (like the

member index). In transmitting, the device adds the streamSequence value into the packet header.

In fig. 9, the PRK is derived by combining the shared secret and the salt component using the first KDF of the second random number generator. The PRK then serves as an intermediate value to derive both the common nonce ensuring uniqueness and unpredictability for each encryption operation and an encryption key via the second KDF (of the second RNG). This second RNG stage propagates cryptographic freshness into the derivation process and allows generation of single or multiple distinct crypto-products for different purposes. A technical advantage of this two-stage derivation process, namely dual RNG workflow, is that it provides enhanced entropy propagation and runtime re-keying capability, thereby ensuring that each encryption operation within the group communication remains unique, resistant to replay or key-compromise attacks, and independent from previously derived keys.

In an embodiment, when a next group communication is to be enabled amongst the plurality of clients 804, 806, and 808, at least one of: the shared secret 902, the salt component 904, is changed and distributed amongst the plurality of clients 804, 806, and 808. In this regard, when the shared secret 902 is to be changed, the trust provider 802 generates a new shared secret and distributes corresponding encapsulated versions of the new shared secret to the plurality of clients 804, 806, and 808. Furthermore, when the salt component 904 is to be changed, each of the plurality of clients 804, 806, and 808 receive a new salt component. The new salt component may be received using a same authentication system which was utilized earlier at the time of receiving it. It will be appreciated that changing the at least one of: the shared secret 902, the salt component 904, for the next group communication is a simple, easy to implement, and reliable way of providing security in subsequent group communications. If the clients use a method like that in fig. 9 for cryptographically protected communications, they can

arrange for the distribution of a new shared secret for each call or other individual session of communications. Additionally, or alternatively, they may arrange for the distribution of a new common salt 904 for each call or other individual session of communications. Using the shared secret 902 and salt component 904 guarantees that the PRK 920 will be unique crypto-product described above, capable of being used for cryptographically protecting any communications session.

Also, as in the preferable embodiment the seed (i.e. the shared secret 902) comes from the first RNG 810 (such as the hardened true random number generator) of the trust provider 802, the described method is immune to so-called RNG manipulation, namely a security breach based on an unauthorized party being able to access and/or manipulate the second random number generator 812, 814, and 816 of any client 804, 806, and 808. Further, use of sufficiently long values as shared secrets makes the method even quantum resistant, meaning that brute force attacks are not likely to succeed even if attempted with quantum computers of the kind envisioned at the time of writing this text. The length of the shared secret 902 can be defined essentially freely by using a suitably programmed, hardened true random generator as the first RNG 810.

In an embodiment, each client amongst the plurality of clients 804, 806, and 808 is further configured to obfuscate the shared secret 902 and the one or more crypto-products 920, 930 and 940, upon an end of the session. This provides an additional layer of security in case the security measures of any client 804, 806, or 808 were later successfully penetrated by an unauthorized party who tried to gain access to the shared secret and the one or more crypto-products. This feature enables the plurality of clients 804, 806, and 808 to be implemented as disposable devices.

In an embodiment, the at least one trust provider 802 is further configured to:

- detect when any one of the following occurs:

- a duration of the session of the group communication exceeds a first predefined threshold, or

- a number of data packets exchanged during the group communication exceeds a second predefined threshold;

- generate a new shared secret corresponding to the session upon said detection, by executing the first random number generator 810;

- digitally encapsulate the new shared secret in a plurality of new versions, using the respective ephemeral keys corresponding to the plurality of clients 804, 806, and 808; and

- distribute respective versions amongst the plurality of new versions of the shared secret to corresponding clients amongst the plurality of clients 804, 806, and 808, during the session.

In this regard, detection of any of such conditions indicates that the high-entropy shared secret 902 has been utilized sufficiently and that it should be refreshed to generate the new shared secret (also having high entropy), to maintain security in the group communication. The digital encapsulation and distribution of the new shared secret is performed in a similar manner as that for the (previous) shared secret. The new shared secret may be utilized for the remaining duration of the session or for exchanging remaining data packets, or the new shared secret may be further refreshed to generate one or more newer shared secrets subsequently.

Additionally, or alternatively, in an embodiment, each client is further configured to:

- detect when any one of the following occurs:

- a duration of the session of the group communication exceeds a first predefined threshold, or

- a number of data packets exchanged during the group communication exceeds a second predefined threshold;

and refreshing at least one of: the shared secret or the salt component.

Optionally, the first predefined threshold can be configured based on the expected session duration. If the duration of the group communication session exceeds this threshold, it signifies that the session has been active long enough to generate an adequate level of randomness for the session. Further optionally the second predefined threshold can be set based on the expected volume of data exchanged, for example if the number of data packets exchanged during the group communication surpasses this threshold, it indicates that a substantial amount of communication has occurred within the session.

Fig. 10 illustrates steps of a method 1000 for securely enabling the group communication. The method 1000 is implemented by the arrangement 800 comprising at least one trust provider 802 and the plurality of clients 804, 806, and 808 that are communicably coupled to each other and to the at least one trust provider 802. At step 1002, there is received, at the at least one trust provider 802, a request for establishing a session of the group communication, from a first client 804 amongst the plurality of clients 804, 806, and 808. At step 1004, there is generated a shared secret 902 corresponding to the session, at the at least one trust provider 802, by executing the first random number generator 810. At step 1006, the shared secret 902 is digitally encapsulated in the plurality of versions 313, 314, and 315, at the at least one trust provider 802, using respective ephemeral keys 309, 310, and 311 corresponding to the plurality of clients 804, 806, and 808, the respective ephemeral keys being available with the at least one trust provider 802. According to an example the encapsulated keys are available only in that trust provider which created those. Other trust providers may request them against respective group member user's public encryption key from their respective trust provider. At step 1008, a respective version amongst the plurality of versions of the shared secret 902 is distributed from the at least one trust provider 802 to a corresponding client amongst the plurality of clients 804, 806, and 808, prior to a start of the session. At step

1010, the respective version of the shared secret 902 is decapsulated at each client amongst the plurality of clients 804, 806, and 808, using a respective private key. At step 1012, the identification credential that is indicative of the user's authorization for using the shared secret 902, is obtained at each client, wherein the identification credential is used to provide the salt component 904. At step 1014, the one or more crypto-products 920, 930, and 940 are derived at each client, by executing the second random number generator 812, 814, and 816, respectively, wherein the shared secret 902 and the salt component 904 are used as inputs to the second random number generator. At step 1016, the one or more crypto-products 920, 930, and 940 are employed, at each client, for encrypting outgoing communication from said client, and decrypting incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients 804, 806, and 808.

The aforementioned steps are only illustrative and other alternatives can also be provided where one or more steps are added, one or more steps are removed, or one or more steps are provided in a different sequence without departing from the scope of the claims.

Optionally, the present disclosure provides an arrangement for securely enabling a group communication, the arrangement comprising at least one trust provider and a plurality of clients that are communicably coupled to each other and to their respective trust provider,

wherein the at least one trust provider is configured to:

- receive a request for establishing a session of the group communication, from a first client amongst the plurality of clients;
- generate a shared secret corresponding to the session, by executing a first random number generator;
- digitally encapsulate the shared secret in a plurality of versions, using respective ephemeral keys corresponding to the plurality of clients, the

respective ephemeral keys being available with the at least one trust provider; and

- distribute a respective version amongst the plurality of versions of the shared secret, to a corresponding client amongst the plurality of clients, prior to a start of the session; and

wherein each client amongst the plurality of clients is configured to:

- decapsulate the respective version of the shared secret, using their respective private key, wherein the respective private key used for decapsulation of the respective version of the shared secret at a client corresponds to the respective ephemeral key used for encapsulation of the shared secret for said client;

- obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is used to provide a salt component, as an access/controlling means to get a salt from the trust provider or from the party which initiated the communication session, or it can be formulated by trust provider using proprietary algorithms, resulting into the same salt component provided to each client amongst the plurality of clients;

- derive one or more crypto-products, by executing a second random number generator, wherein the shared secret and the salt component are used as inputs to the second random number generator; and

- employ the one or more crypto-products to

- encrypt outgoing communication from said client, and

- decrypt incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients.

Optionally, the first random number generator is used as a source of true randomness to generate a high-entropy shared secret.

Furthermore, optionally, the present disclosure also provides a trust provider for securely enabling a group communication between a plurality of clients, the trust provider being communicably coupled to at least one client of the plurality of clients, wherein the trust provider is configured to execute respective steps of the aforesaid arrangement.

Furthermore, optionally, the present disclosure also provides a client for securely participating in a group communication, the client being communicably coupled to a trust provider and at least one other client, wherein the client is configured to execute respective steps of the aforesaid arrangement.

APPENDIX

1. Hereinbelow, there is provided an example pseudocode for a CTR-DRBG algorithm.

```
FUNCTION CTR_DRBG (seed, salt component):
    # Step 1: Combine the seed and salt component using a first key derivation function (KDF), seed is the shared secret
    key = First KDF (seed, salt component)
    # Step 2: Define the counter (nonce)
    counter = GenerateRandomCounter()
    # Step 3: Initialize AES block cipher with the key
    aes_cipher = InitializeAES(key)
    # Step 4: Generate the random bit stream using AES in counter mode
    random_output = AES_CTR_Mode (aes_cipher, counter)
    # Return the random bit stream
    RETURN random output
END FUNCTION
```

2. Hereinbelow, there is provided an example pseudocode for a HMAC-based DRBG algorithm.

```
FUNCTION HMAC_DRBG (seed, salt component):
    # Step 1: Combine the seed and salt component using a first key derivation function (KDF), seed is the shared secret
    key = First KDF(seed, salt component)
    # Step 2: Initialize HMAC with the key
    hmac_instance = InitializeHMAC(key)
    # Step 3: Input the original seed into the HMAC function
    hmac_instance.Update(seed)
    # Step 4: Generate the random bit stream from HMAC
    random_output = hmac_instance.Generate()
    # Return the random bit stream
    RETURN random_output
END FUNCTION
```

3. Hereinbelow, there is provided an example pseudocode for a HASH-DRBG algorithm.

```
FUNCTION HASH_DRBG (seed, salt component):
    # Step 1: Combine the seed and salt component using a first key derivation function (KDF), seed is the shared secret
    combined_input = First KDF (seed, salt component)
    # Step 2: Initialize the hash function (e.g., SHA256)
    hash_instance = InitializeHashFunction(SHA256)
    # Step 3: Input the combined seed into the hash function
    hash_instance.Update(combined_input)
    # Step 4: Generate the random bit stream from the hash function
    random_output = hash_instance.Generate()
    # Return the random bit stream
    RETURN random_output
END FUNCTION
```

4. Hereinbelow, there is provided an example pseudocode for deriving a symmetric key.

```
FUNCTION GenerateSymmetricKeyUsingDRBG(shared secret, salt component):
    # Step 1: Generate random bytes using a DRBG (e.g., CTR-DRBG, HMAC-DRBG, or HASH-DRBG)
    random_bytes = DRBG(shared secret, salt component)
    # Step 2: Truncate or use the generated bytes as the symmetric key
    symmetric_key = random_bytes[:key_length]
    # Return the symmetric key
    RETURN symmetric_key
END FUNCTION
```

5. Hereinbelow, there is provided an example pseudocode for deriving an asymmetric key pair.

```
FUNCTION GenerateKeyPairUsingDRBG(shared secret, salt component):
    # Step 1: Generate random bytes using a DRBG for the private key seed
    private_key_seed = DRBG(shared secret, salt component)
    # Step 2: Derive the private key from the shared secret (e.g., Curve25519, Ed25519, ECDSA, RSA, Kyber1024, or Dilithium5)
    private_key = DerivePrivateKey(private_key_seed)
    # Step 3: Derive the corresponding public key from the private key
    public_key = DerivePublicKey(private_key)
    # Return the key pair
    RETURN (public_key, private_key)
END FUNCTION
```

6. Hereinbelow, there is provided an example pseudocode for deriving one or more keys for encrypting and/or wrapping other key(s).

```
FUNCTION GenerateKeyWrappingKey(shared secret, salt component):
    # Step 1: Generate random bytes using DRBG for the wrapping key
    wrapping_key = DRBG(shared secret, salt component)
    # Step 2: Use the wrapping key to encrypt another key
    wrapped_key = EncryptKey(wrapping_key, key_to_wrap)
    # Return the wrapped key
    RETURN wrapped_key
END FUNCTION
```

7. Hereinbelow, there is provided an example pseudocode for deriving a nonce value or initialization vector.

```
FUNCTION GenerateNonceUsingDRBG(shared secret, salt component):
    # Step 1: Generate random bytes using DRBG for the nonce/IV
    nonce = DRBG(shared secret, salt component)
    # Return the nonce
    RETURN nonce
END FUNCTION
```

8. Hereinbelow, there is provided an example pseudocode for deriving a secure key from an identification credential

```
FUNCTION DeriveKeyUsingDRBG(shared secret, identification credential):
    # Step 1: Combine the shared secret and the identification credential with a KDF
    derived_key = KDF(DRBG(shared secret, identification credential))
    # Return the derived key
```

```
    RETURN derived_key
END FUNCTION
```

9. Hereinbelow, there is provided an example pseudocode for deriving a secure password or a derivative of the secure password.

```
FUNCTION GeneratePasswordUsingDRBG(shared secret, salt component):
    # Step 1: Generate random bytes using DRBG
    password_bytes = DRBG(shared secret, salt component)
    # Step 2: Optionally encode the bytes into a user-friendly format (e.g., base64 or hex)
    password = EncodeToReadableFormat(password_bytes)
    # Return the password
    RETURN password
END FUNCTION
```

10. Hereinbelow, there is provided an example pseudocode for deriving a session key.

```
FUNCTION GenerateSessionKeyUsingDRBG(shared secret, salt component):
    # Step 1: Generate random bytes using DRBG for session key
    session_key = DRBG(shared secret, salt component)
    # Return the session key
    RETURN session_key
END FUNCTION
```

11. Hereinbelow, there is provided an example pseudocode for deriving a one-time key.

```
FUNCTION GenerateOneTimeKeyUsingDRBG(shared secret, salt component):
    # Step 1: Generate random bytes using DRBG for one-time key
    one_time_key = DRBG(shared secret, salt component)
```

```
    # Return the one time key
    RETURN one_time_key
END FUNCTION
```

5

CLAIMS

1. An arrangement (800) for securely enabling a group communication, the arrangement comprising at least one trust provider (802) and a plurality of clients (804, 806, 808) that are communicably coupled to each other and to their respective trust provider
wherein the at least one trust provider is configured to:

- receive a request for establishing a session of the group communication, from a first client (804) amongst the plurality of clients;
- generate a shared secret (902) corresponding to the session, by executing a first random number generator (810);
- digitally encapsulate the shared secret in a plurality of versions, using respective ephemeral keys corresponding to the plurality of clients, the respective ephemeral keys being available with the at least one trust provider; and
- distribute a respective version amongst the plurality of versions of the shared secret, to a corresponding client amongst the plurality of clients, prior to a start of the session; and

wherein each client amongst the plurality of clients is configured to:

- decapsulate the respective version of the shared secret, using their respective private key;
- obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is used to provide a salt component (904);
- derive one or more crypto-products (920, 930, 940), by executing a second random number generator (812, 814, 816), wherein the shared secret and the salt component are used as inputs to the second random number generator; and
- employ the one or more crypto-products to
    - encrypt (916) outgoing communication from said client, and

- decrypt (918) incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients.

2. An arrangement (800) according to claim 1, wherein when deriving the one or more crypto-products (920, 930, 940), the shared secret (902) is used as a seed component, and wherein each client (804, 806, 808) is configured to:

- combine the seed component and the salt component (904) using: a first derivation function (906) of the second random number generator (812, 814, 816), to derive a Pseudo-Random Key (PRK) (920).

3. An arrangement according to claim 2, wherein each client is further configured to

- hash the PRK using a nonce derivation function (908), to derive a common nonce (930); and

- combine the PRK and contextual information (912) associated with encryption, using a second derivation function (910) of the second random number generator, to derive an encryption key (940);

wherein the encryption key and the common nonce are utilised for performing the encryption (916) of the outgoing communication and the decryption (918) of the incoming communication.

4. An arrangement (800) according to claim 3, wherein the encryption key (940) is dynamically re-computable without requiring its storage at said client (804, 806, 808).

5. An arrangement (800) according to claim 3 or 4, wherein each client (804, 806, 808) is further configured to:

- receive a start value (nonceStart) for stream nonce; and

- retrieve a group size value and a member index of said client, the group size value being equal to a number of clients in the plurality of clients;

wherein when encrypting (916) the outgoing communication, said client is configured to:

- determine a stream sequence value for each data packet, based on the start value for stream nonce, the group size value, the member index, and a packet sequence number;

- define a stream nonce, based on the common nonce (930) and the stream sequence value;

- create a cipher block with the encryption key (940);

- create an outgoing keystream with the cipher block and the stream nonce;

- combine each data packet with the outgoing keystream, using an exclusive OR (XOR) operation, to generate encrypted data packets; and

- add the stream sequence value for each encrypted data packet into a header of said encrypted data packet, prior to transmitting said encrypted data packet.

6. An arrangement (800) according to claim 5, wherein when decrypting (918) the incoming communication, said client (804, 806, 808) is configured to:

- obtain the stream sequence value, from the header of each encrypted data packet that is received at said client;

- define the stream nonce, based on the common nonce (930) and the stream sequence value;

- create a plaintext block with the encryption key (940);

- create an incoming keystream with the plaintext block and the stream nonce, the incoming keystream being same as the outgoing keystream; and

- combine each encrypted data packet with the incoming keystream, using the XOR operation, to generate decrypted data packets.

7. An arrangement (800) according to any of the preceding claims, wherein the at least one trust provider (802) is further configured to:

- detect when any one of the following occurs:
    - a duration of the session of the group communication exceeds a first predefined threshold, or
    - a number of data packets exchanged during the group communication exceeds a second predefined threshold;
- generate a new shared secret corresponding to the session upon said detection, by executing the first random number generator (810);
- digitally encapsulate the new shared secret in a plurality of new versions, using the respective ephemeral keys corresponding to the plurality of clients (804, 806, 808); and
- distribute respective versions amongst the plurality of new versions of the shared secret to corresponding clients amongst the plurality of clients, during the session.

8. An arrangement (800) according to any of the preceding claims, wherein each client (804, 806, 808) is further configured to:
- detect when any one of the following occurs:
    - a duration of the session of the group communication exceeds a first predefined threshold, or
    - a number of data packets exchanged during the group communication exceeds a second predefined threshold;
    and
- refresh at least one of: the shared secret or the salt component.

9. An arrangement (800) according to any of the preceding claims, wherein each client amongst the plurality of clients (804, 806, 808) is further configured to obfuscate the shared secret (902) and the one or more crypto-products (920, 930, 940), upon an end of the session.

10. An arrangement (800) according to any of the preceding claims, wherein when a next group communication is to be enabled amongst the plurality of clients (804, 806, 808), at least one of: the shared secret

(902), the salt component (904), is changed and distributed amongst the plurality of clients.

11. An arrangement (800) according to any of the preceding claims, wherein upon decapsulation of the respective version of the shared secret (902) at each client, the shared secret is extracted into a programmatically protected memory, the programmatically protected memory being one of: a segregated memory provided by Hardware Security Module (HSM), a secure enclave, a secure element, a Trusted Platform Module (TPM), a guarded heap memory.

12. An arrangement (800) according to any of the preceding claims, wherein the identification credential is at least one of: personal secret-based authentication, identity card-based authentication, cryptographic authentication, wireless communication-based authentication, token-based authentication, notification-based authentication, user identifier, code-based authentication, digital identity-based authentication, biometric and/or neural authentication, anti-spoofing authentication.

13. An arrangement (800) according to any of the preceding claims, wherein the second random number generator (812, 814, 816) employs a Deterministic Random Bit Generator (DRBG) algorithm when deriving the one or more crypto-products (920, 930, 940), the DRBG algorithm being one of: a counter mode DRBG (CTR-DRBG) algorithm, a Hash-based Message Authentication Code (HMAC)-based DRBG algorithm, a HASH-DRBG algorithm.

14. An arrangement (800) according to any of the preceding claims, wherein the one or more crypto-products (920, 930, 940) comprise at least one of: a symmetric key (940), an asymmetric key pair, one or more keys for encrypting and/or wrapping other key(s), a nonce value (930) or initialization vector, a secure key from an identification credential, a secure password, a derivative of the secure password, a session

key, a one-time key, a pre-shared key (PSK) of a Transport Layer Security (TLS)-PSK model.

15. An arrangement (800) according to any of the preceding claims, wherein the group communication is one of: a group audio call, a group video call, a group chat, a group email conversation, a group workspace communication, a group conference call, a group broadcast, a group network, a packet data session.

16. A trust provider (802) for securely enabling a group communication between a plurality of clients, the trust provider being communicably coupled to at least one client (804, 806, 808) of the plurality of clients, wherein the trust provider is configured to:

- receive a request for establishing a session of the group communication, from a first client (804) amongst the plurality of clients;
- generate a shared secret (902) corresponding to the session, by executing a first random number generator (810);
- digitally encapsulate the shared secret in a plurality of versions, using respective ephemeral keys corresponding to the plurality of clients, one or more of the respective ephemeral keys being available with the trust provider; and
- distribute a respective version amongst the plurality of versions of the shared secret, to a corresponding client amongst the plurality of clients, prior to a start of the session,

wherein the respective version of the shared secret is decapsulated at each client amongst the plurality of clients, using a respective private key, the shared secret is used at each client to derive one or more crypto-products (920, 930, 940) by executing a second random number generator (812, 814, 816), and the one or more crypto-products are employed at each client to encrypt (916) outgoing communication from said client, and to decrypt (918) incoming communication at said client.

17. A client (804, 806, 808) for securely participating in a group communication, the client being communicably coupled to a trust provider (802) and at least one other client, wherein the client is configured to:

- decapsulate a respective version of a shared secret (902), using a respective private key, the respective version being received from the trust provider prior to a start of a session of the group communication, wherein the shared secret is generated by the trust provider using a first random number generator (810);
- obtain an identification credential that is indicative of a user's authorization for using the shared secret, wherein the identification credential is used to provide a salt component (904);
- derive one or more crypto-products (920, 930, 940), by executing a second random number generator (812, 814, 816), wherein the shared secret and the salt component are used as inputs to the second random number generator; and
- employ the one or more crypto-products to
    - encrypt (916) outgoing communication from said client, and
    - decrypt (918) incoming communication at said client, the incoming communication being received from the at least one other client.

18. A method (1000) for securely enabling a group communication, the method being implemented by an arrangement (800) comprising at least one trust provider (802) and a plurality of clients (804, 806, 808) that are communicably coupled to each other and to their respective trust provider, wherein the method comprises:

- receiving, at the at least one trust provider, a request for establishing a session of the group communication, from a first client (804) amongst the plurality of clients;
- generating a shared secret (902) corresponding to the session, at the at least one trust provider, by executing a first random number generator (810);

- digitally encapsulating the shared secret in a plurality of versions, at the at least one trust provider, using respective ephemeral keys corresponding to the plurality of clients, the respective ephemeral keys being available with the at least one trust provider;
- distributing a respective version amongst the plurality of versions of the shared secret, from the at least one trust provider to a corresponding client amongst the plurality of clients, prior to a start of the session;
- decapsulating the respective version of the shared secret, at each client amongst the plurality of clients, using a respective private key;
- obtaining an identification credential that is indicative of a user's authorization for using the shared secret, at each client, wherein the identification credential is used provide a salt component (904);
- deriving one or more crypto-products (920, 930, 940) at each client, by executing a second random number generator (812, 814, 816), wherein the shared secret and the salt component are used as inputs to the second random number generator; and
- employing the one or more crypto-products, at each client, for
    - encrypting (916) outgoing communication from said client, and
    - decrypting (918) incoming communication at said client, the incoming communication being received from at least one other client amongst the plurality of clients.

19. A method according to claim 18, wherein at the step of deriving the one or more crypto-products (920, 930, 940) at each client (804, 806, 808), the shared secret (902) is used as a seed component, and wherein the step of deriving the one or more crypto-products comprises:
- combining the seed component and the salt component (904) using a first derivation function (906) of the second random number generator (812, 814, 816), to derive a Pseudo-Random Key (PRK) (920).

20. A method according to claim 19, wherein deriving the one or more crypto-products further comprises:

- hashing the PRK using a nonce derivation function (908), for deriving a common nonce (930); and
- combining the PRK and contextual information (912) associated with encryption, using a second derivation function (910) of the second random number generator, for deriving an encryption key (940);
wherein the encryption key and the common nonce are utilised for performing the encryption (916) of the outgoing communication and the decryption (918) of the incoming communication.

21. A method according to claim 20, further comprising dynamically recomputing the encryption key (940), without requiring its storage at said client (804, 806, 808).

22. A method according to claim 18 to 21, wherein the step of employing the one or more crypto-products (920, 930, 940), at each client (804, 806, 808), for encrypting (916) the outgoing communication from said client comprises:
- receiving a start value (nonceStart) for stream nonce;
- retrieving a group size value and a member index of said client, the group size value being equal to a number of clients in the plurality of clients;
- determining a stream sequence value for each data packet, based on the start value for stream nonce, the group size value, the member index, and a packet sequence number;
- defining a stream nonce, based on the common nonce (930) and the stream sequence value;
- creating a cipher block with the encryption key (940);
- creating an outgoing keystream with the cipher block and the stream nonce;
- combining each data packet with the outgoing keystream, using an exclusive OR (XOR) operation, for generating encrypted data packets; and

- adding the stream sequence value for each encrypted data packet into a header of said encrypted data packet, prior to transmitting said encrypted data packet.

23. A method according to claim 22, wherein the step of employing the one or more crypto-products (920, 930, 940), at each client (804, 806, 808), for decrypting (918) the incoming communication at said client comprises:

- obtaining the stream sequence value, from the header of each encrypted data packet that is received at said client;
- defining the stream nonce, based on the common nonce (930) and the stream sequence value;
- creating a plaintext block with the encryption key (940);
- creating an incoming keystream with the plaintext block and the stream nonce, the incoming keystream being same as the outgoing keystream; and
- combining each encrypted data packet with the incoming keystream, using the XOR operation, for generating decrypted data packets.

24. A method according to any of claims 18-23, further comprising:

- detecting, at the at least one trust provider (802), when any one of the following occurs:
    - a duration of the session of the group communication exceeds a first predefined threshold, or
    - a number of data packets exchanged during the group communication exceeds a second predefined threshold;
- generating a new shared secret corresponding to the session upon said detection, by executing the first random number generator (810);
- digitally encapsulating the new shared secret in a plurality of new versions, using the respective ephemeral keys corresponding to the plurality of clients (804, 806, 808); and

- distributing respective versions amongst the plurality of new versions of the shared secret to corresponding clients amongst the plurality of clients, during the session.

25. A method according to any of claims 18-24, further comprising:

- detecting, at each client (804, 806, 808), when any one of the following occurs:
  - a duration of the session of the group communication exceeds a first predefined threshold, or
  - a number of data packets exchanged during the group communication exceeds a second predefined threshold;
  and
- refreshing at least one of: the shared secret or the salt component.

26. A method according to any of claims 18-25, further comprising obfuscating the shared secret (902) and the one or more crypto-products (920, 930, 940), at each client (804, 806, 808), upon an end of the session.

27. A method according to any of claims 18-26, wherein when a next group communication is to be enabled amongst the plurality of clients (804, 806, 808), the method further comprises changing and distributing amongst the plurality of clients, at least one of: the shared secret (902), the salt component (904).

28. A computer program product comprising at least one non-transitory machine-readable data storage medium having stored thereon one or more sets of one or more machine-executable instructions that are configured to, when executed by one or more processors, cause the execution of a method according to any of claims 18-27.

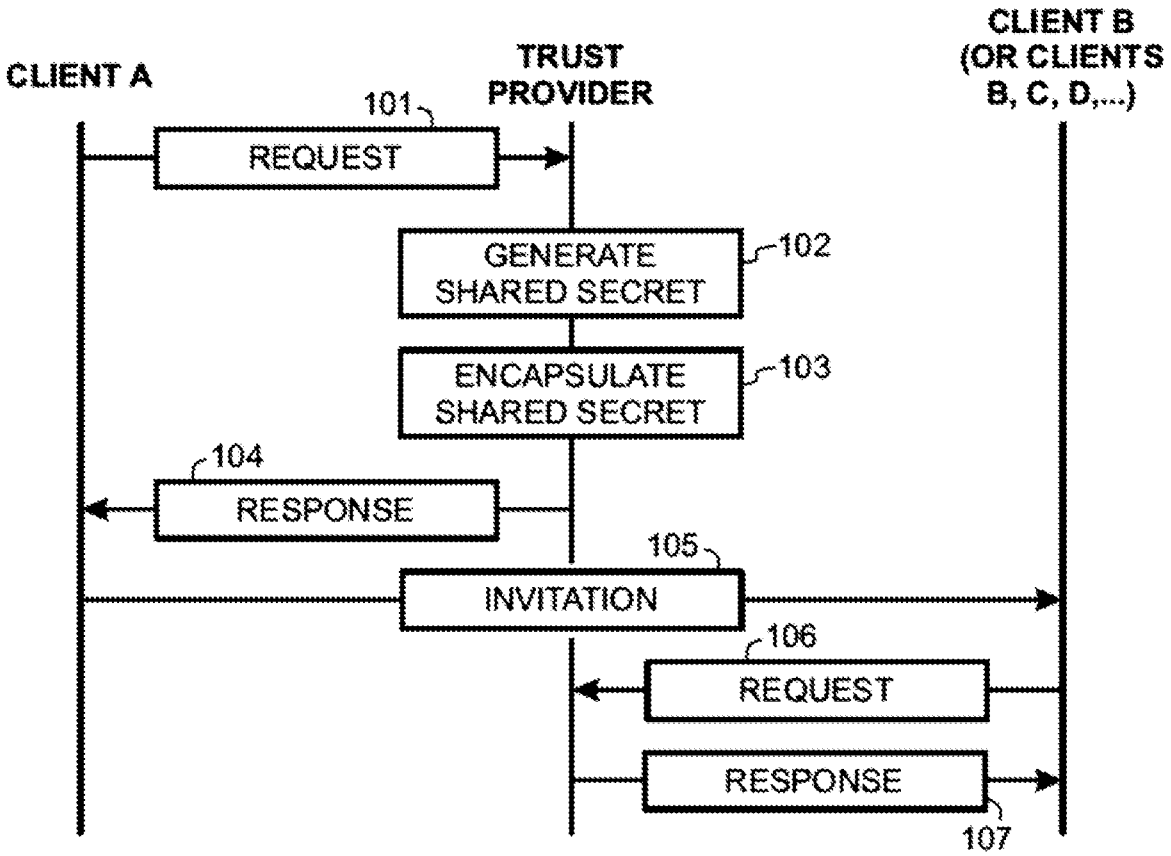
CLIENT A
TRUST PROVIDER
CLIENT B (OR CLIENTS B, C, D,...)
101
REQUEST
GENERATE SHARED SECRET
102
ENCAPSULATE SHARED SECRET
103
104
RESPONSE
105
INVITATION
106
REQUEST
RESPONSE
107

Fig. 1

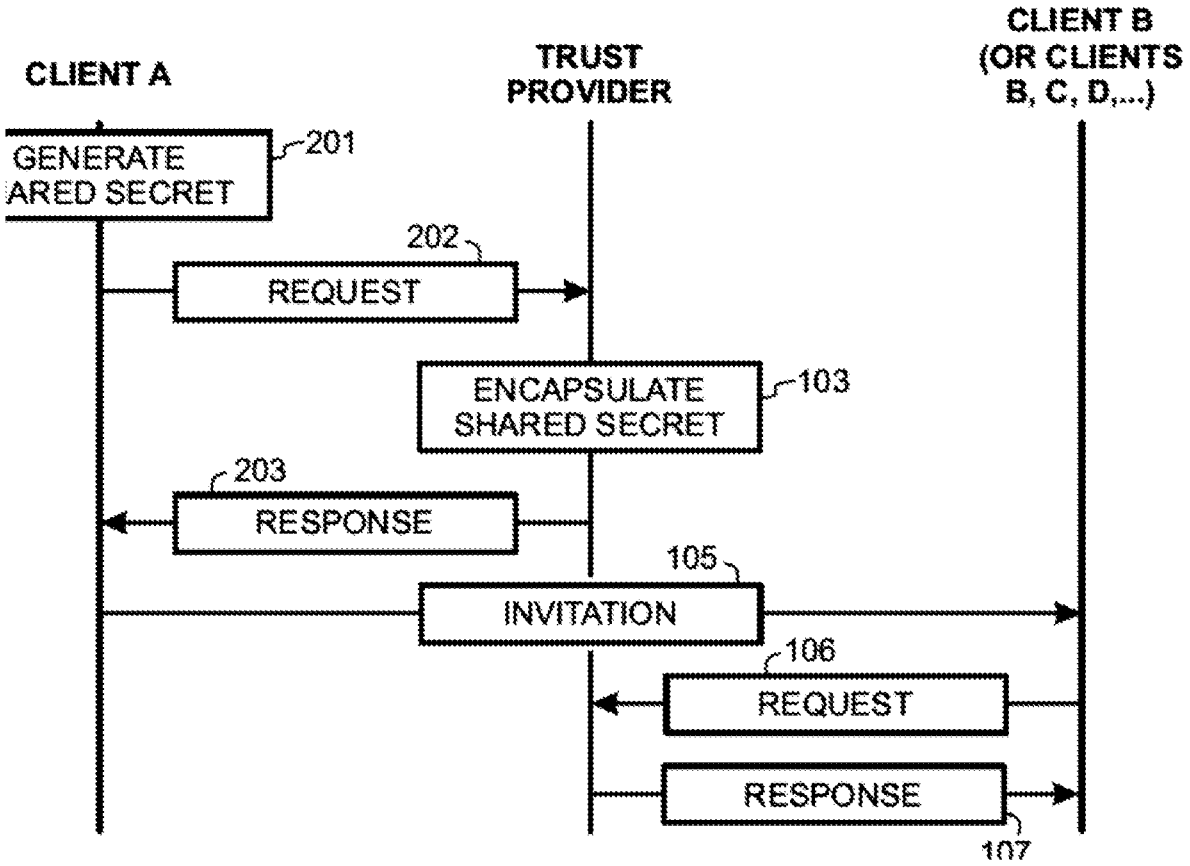
CLIENT A
TRUST PROVIDER
CLIENT B (OR CLIENTS B, C, D,...)
GENERATE ARED SECRET
201
202
REQUEST
ENCAPSULATE SHARED SECRET
103
203
RESPONSE
105
INVITATION
106
REQUEST
RESPONSE
107

Fig. 2

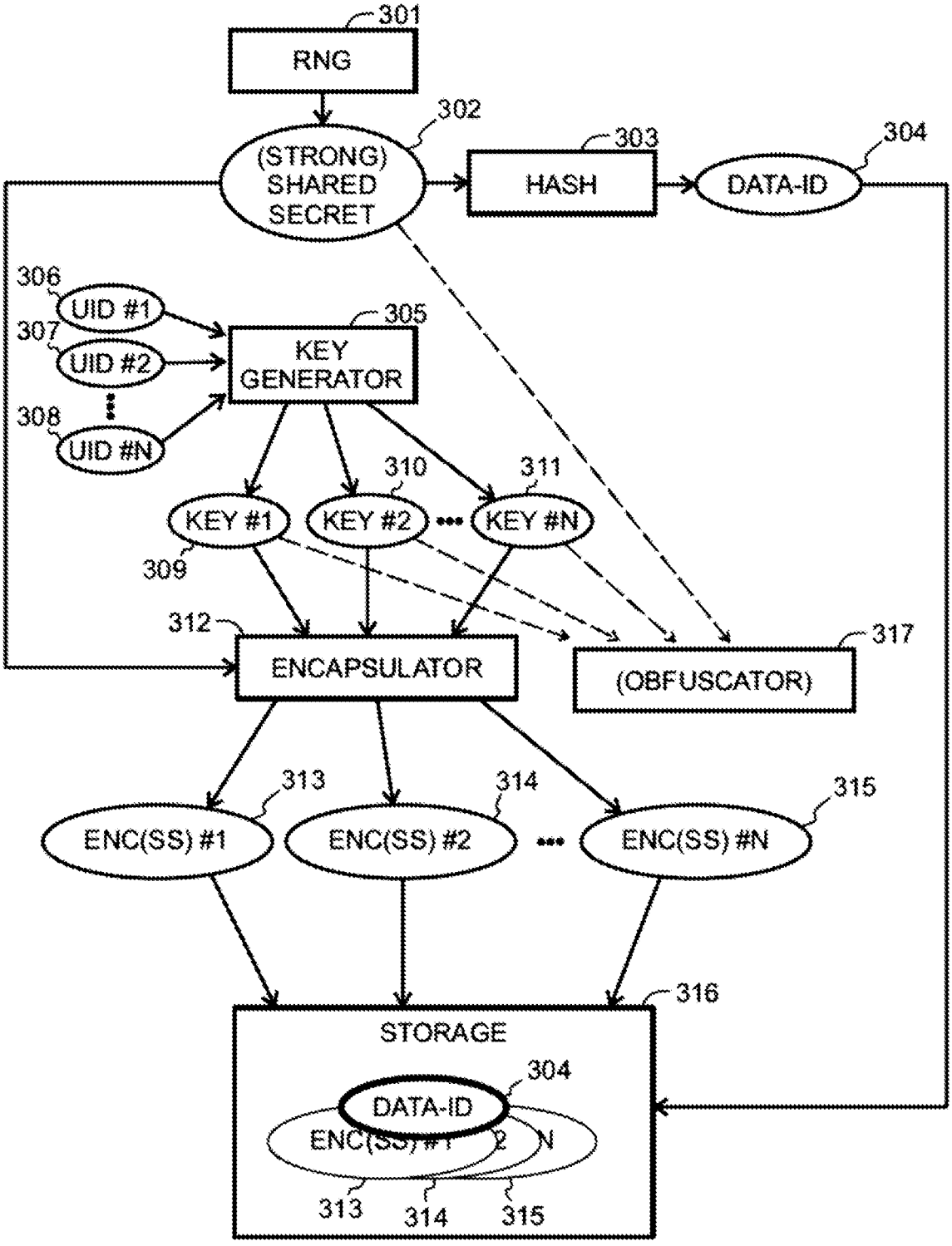
301
RNG
302
(STRONG) SHARED SECRET
303
HASH
304
DATA-ID
306
UID #1
307
UID #2
308
UID #N
305
KEY GENERATOR
310
311
KEY #1
KEY #2
KEY #N
309
312
ENCAPSULATOR
317
(OBFUSCATOR)
313
314
315
ENC(SS) #1
ENC(SS) #2
ENC(SS) #N
316
STORAGE
304
DATA-ID
313
314
315

Fig. 3

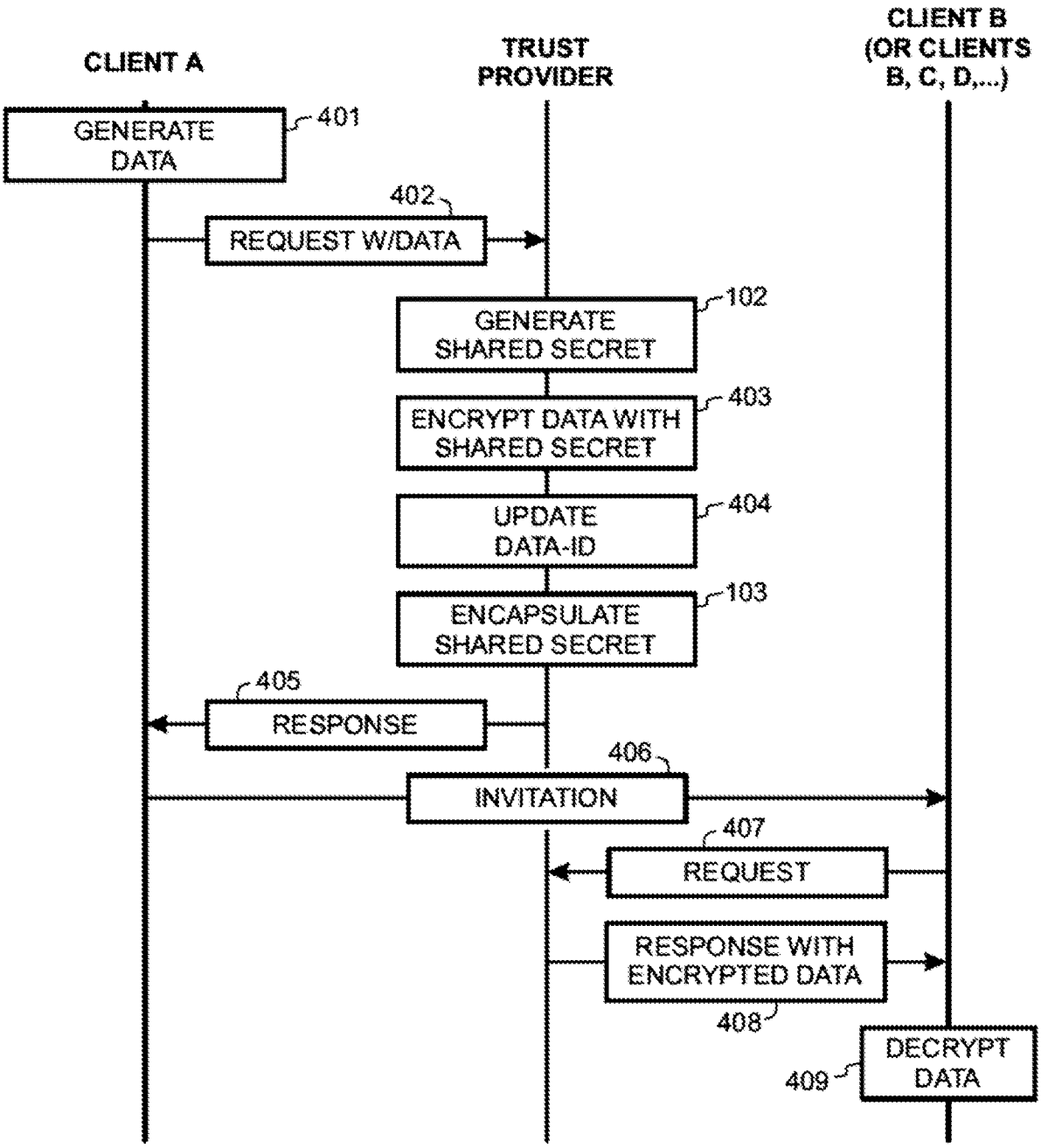
CLIENT A
TRUST PROVIDER
CLIENT B (OR CLIENTS B, C, D,...)
GENERATE DATA
401
402
REQUEST W/DATA
GENERATE SHARED SECRET
102
ENCRYPT DATA WITH SHARED SECRET
403
UPDATE DATA-ID
404
ENCAPSULATE SHARED SECRET
103
405
RESPONSE
406
INVITATION
407
REQUEST
RESPONSE WITH ENCRYPTED DATA
408
DECRYPT DATA
409

Fig. 4

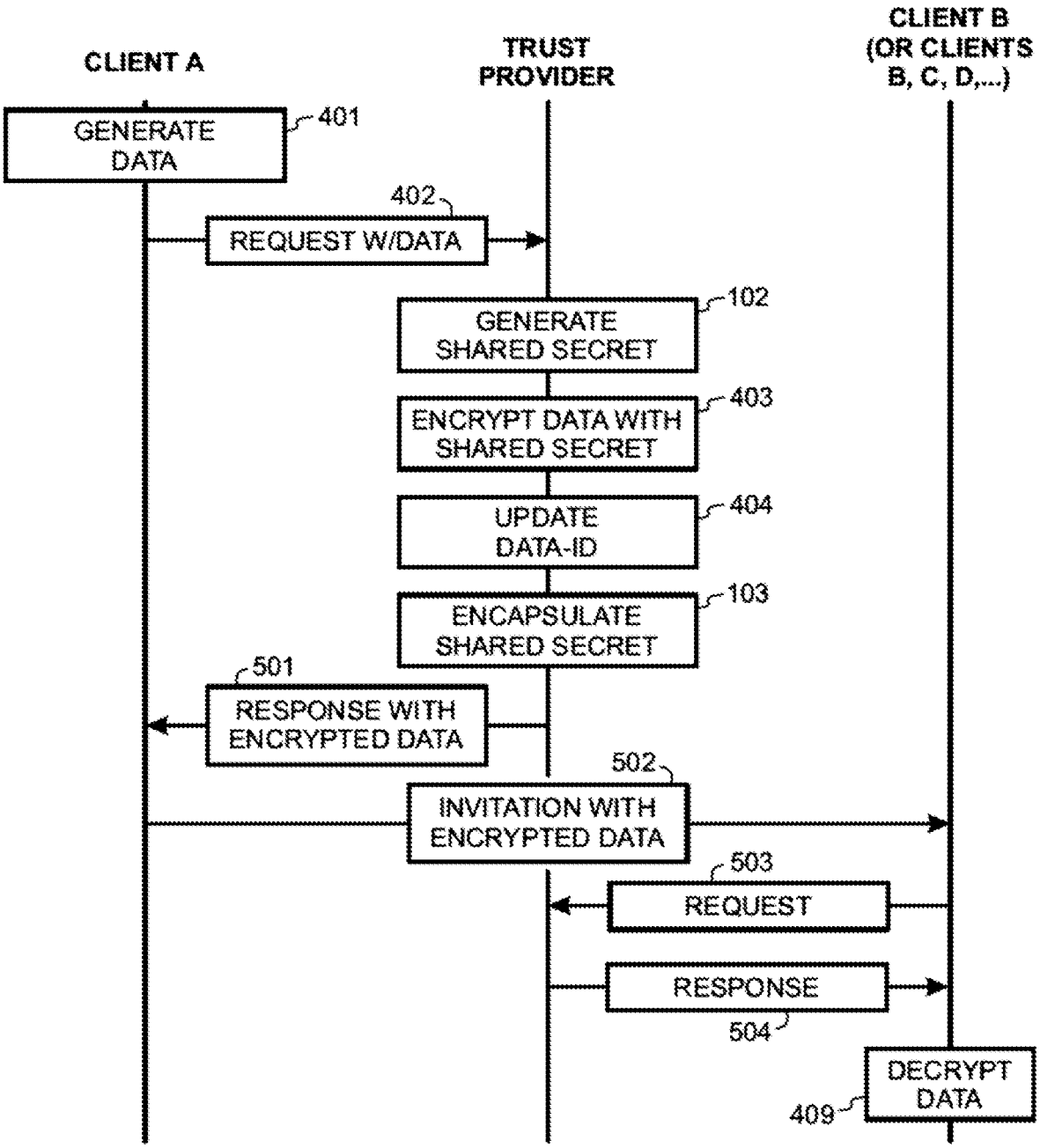
CLIENT A
TRUST PROVIDER
CLIENT B (OR CLIENTS B, C, D,...)
GENERATE DATA
401
402
REQUEST W/DATA
GENERATE SHARED SECRET
102
ENCRYPT DATA WITH SHARED SECRET
403
UPDATE DATA-ID
404
ENCAPSULATE SHARED SECRET
103
501
RESPONSE WITH ENCRYPTED DATA
502
INVITATION WITH ENCRYPTED DATA
503
REQUEST
RESPONSE
504
DECRYPT DATA
409

Fig. 5

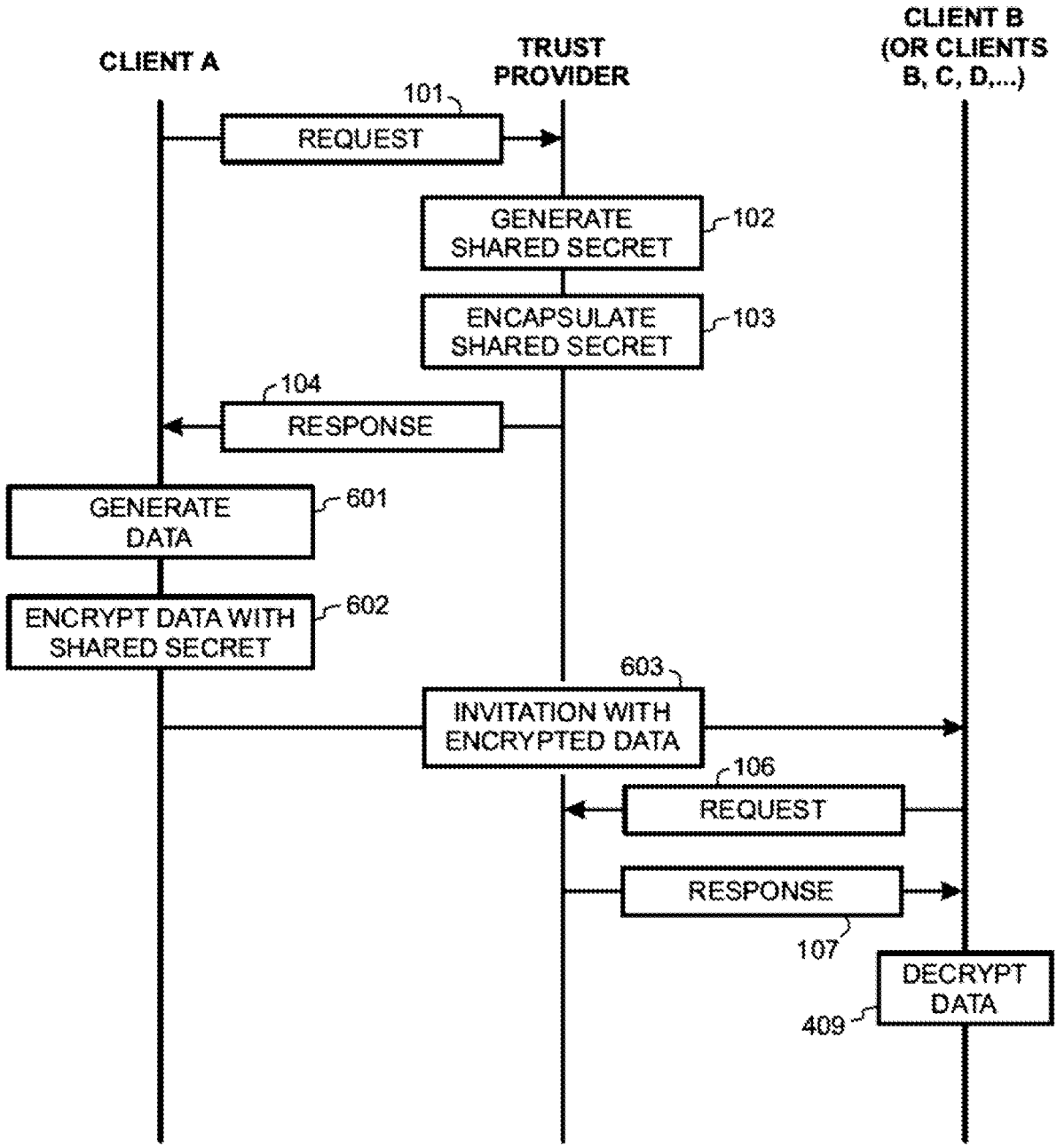
CLIENT A
TRUST PROVIDER
CLIENT B (OR CLIENTS B, C, D,...)
101
REQUEST
GENERATE SHARED SECRET
102
ENCAPSULATE SHARED SECRET
103
104
RESPONSE
GENERATE DATA
601
ENCRYPT DATA WITH SHARED SECRET
602
603
INVITATION WITH ENCRYPTED DATA
106
REQUEST
RESPONSE
107
DECRYPT DATA
409

Fig. 6

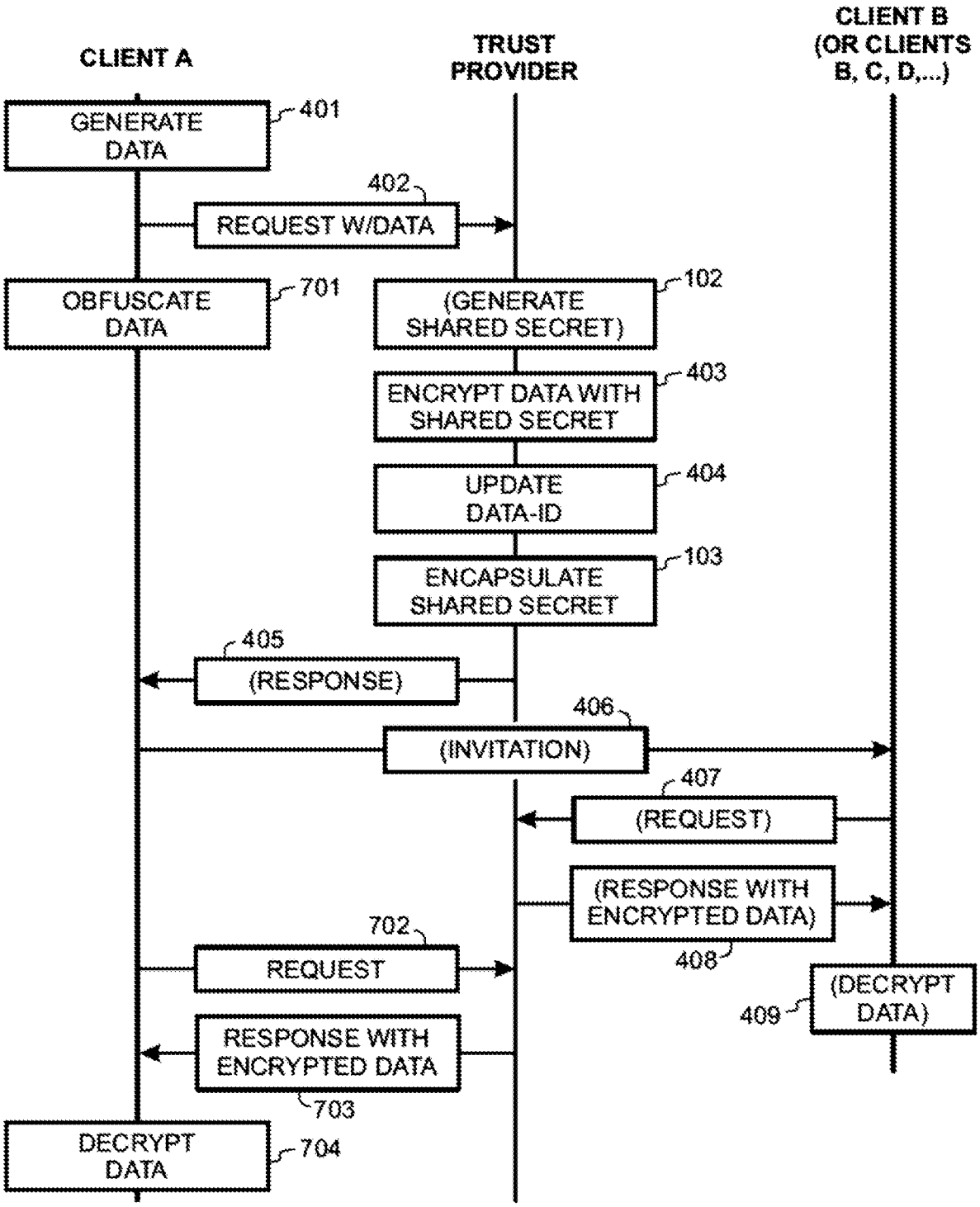
CLIENT A
TRUST PROVIDER
CLIENT B (OR CLIENTS B, C, D,...)
GENERATE DATA
401
402
REQUEST W/DATA
OBFUSCATE DATA
701
(GENERATE SHARED SECRET)
102
ENCRYPT DATA WITH SHARED SECRET
403
UPDATE DATA-ID
404
ENCAPSULATE SHARED SECRET
103
405
(RESPONSE)
406
(INVITATION)
407
(REQUEST)
(RESPONSE WITH ENCRYPTED DATA)
408
(DECRYPT DATA)
409
702
REQUEST
RESPONSE WITH ENCRYPTED DATA
703
DECRYPT DATA
704

Fig. 7

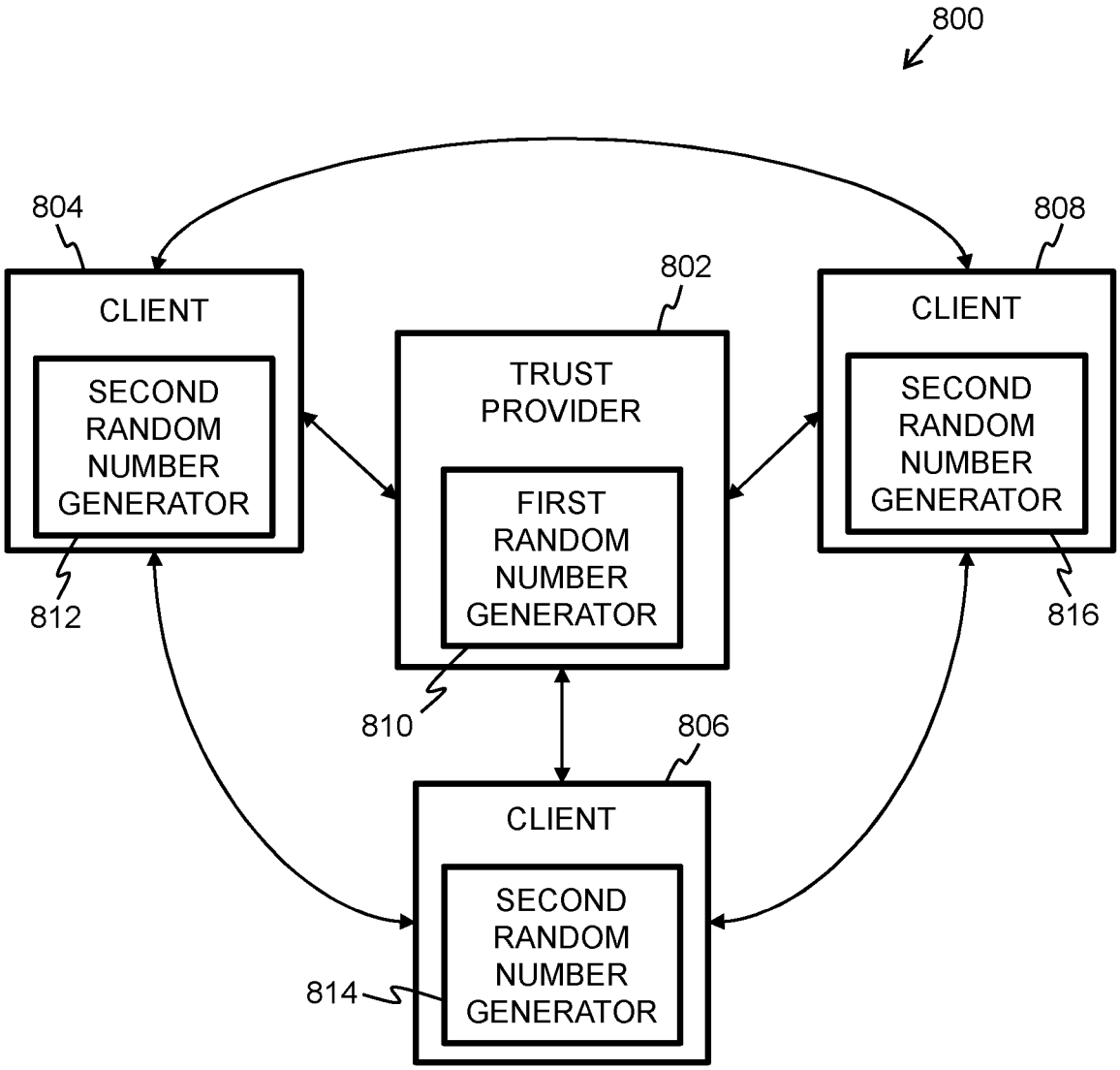
800
804
808
802
CLIENT
SECOND RANDOM NUMBER GENERATOR
TRUST PROVIDER
FIRST RANDOM NUMBER GENERATOR
CLIENT
SECOND RANDOM NUMBER GENERATOR
812
816
810
806
CLIENT
SECOND RANDOM NUMBER GENERATOR
814

Fig. 8

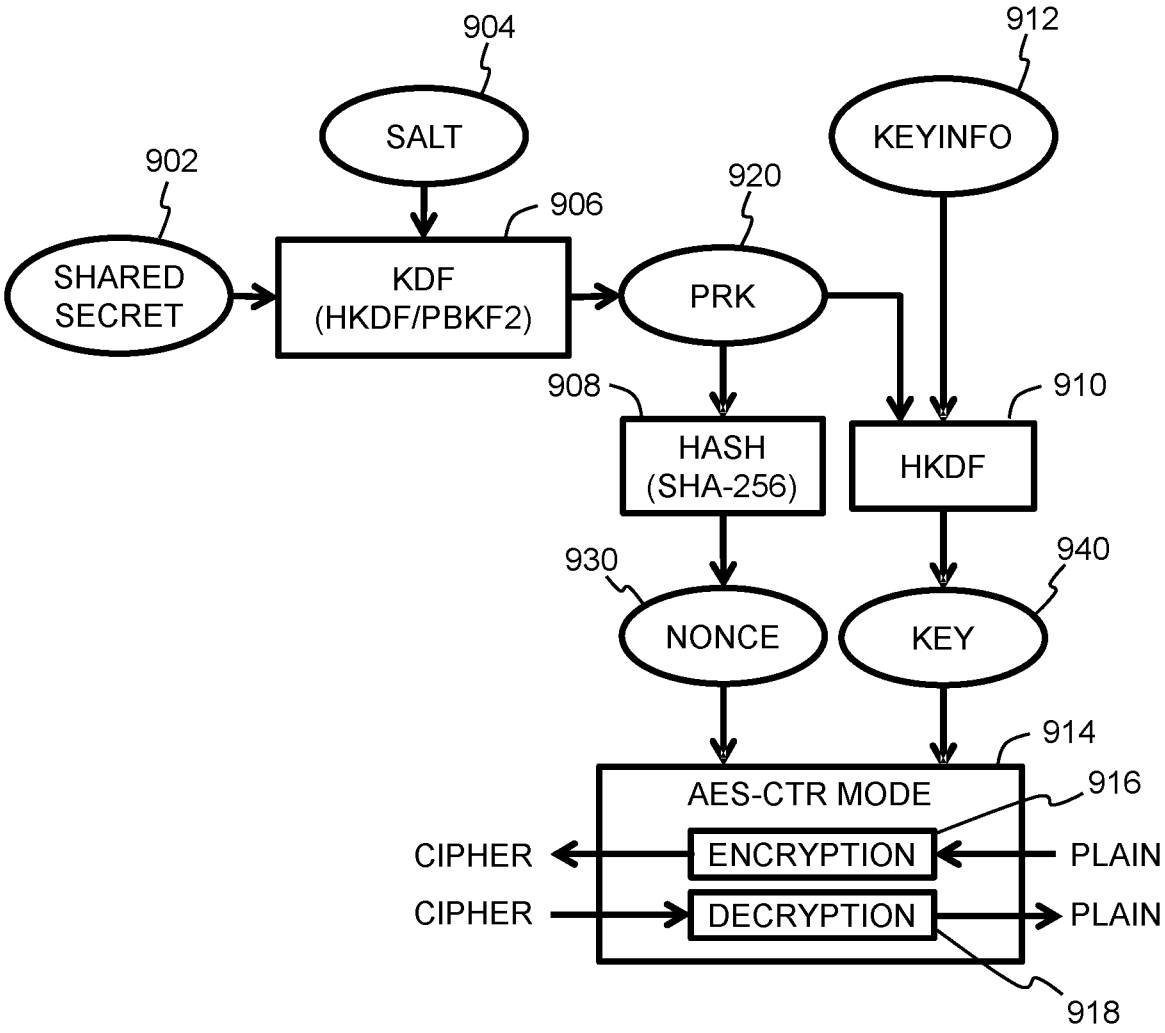
904
SALT
912
KEYINFO
902
SHARED SECRET
906
KDF (HKDF/PBKF2)
920
PRK
908
HASH (SHA-256)
910
HKDF
930
NONCE
940
KEY
914
AES-CTR MODE
916
CIPHER
ENCRYPTION
PLAIN
CIPHER
DECRYPTION
PLAIN
918

Fig. 9

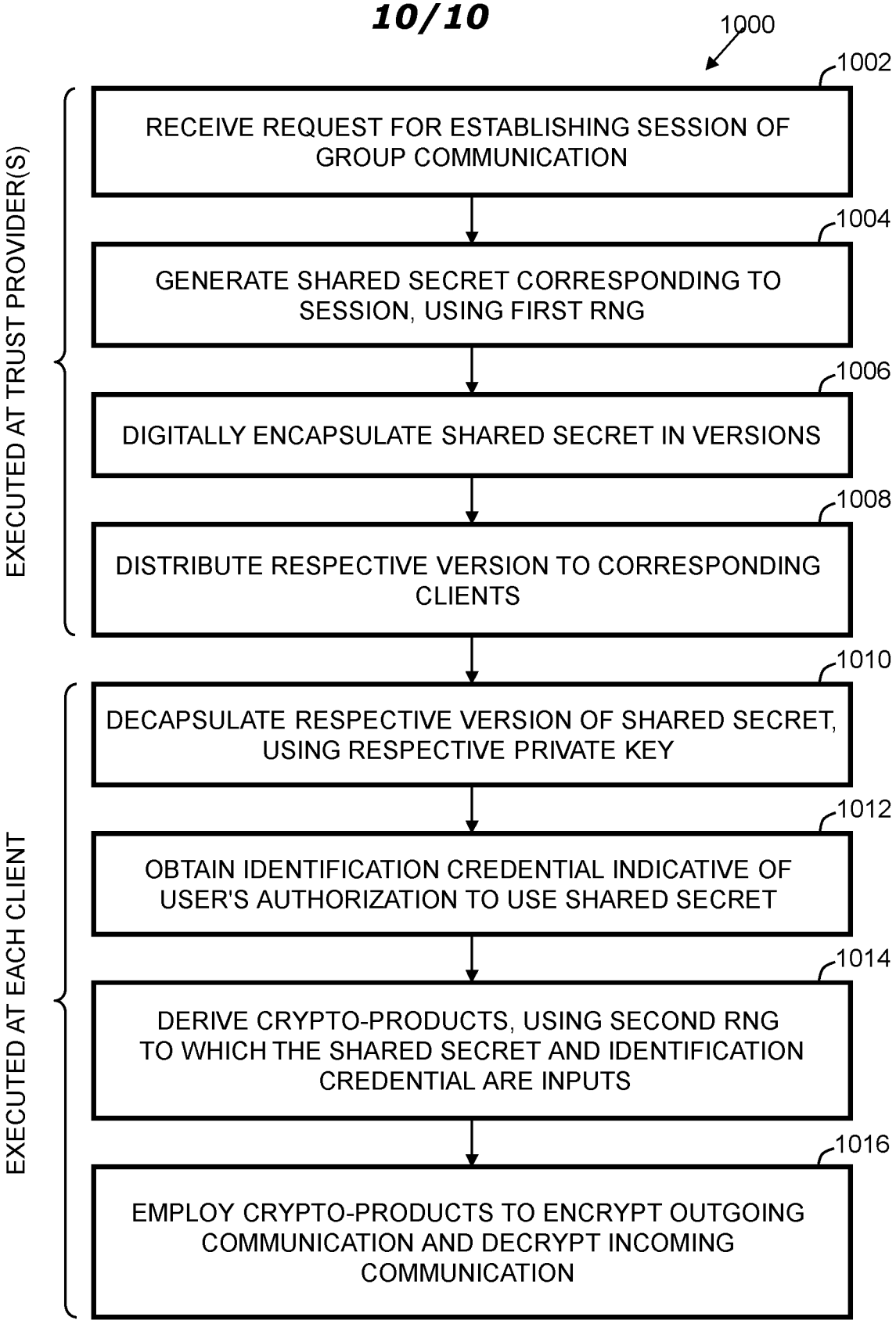
1000
1002
RECEIVE REQUEST FOR ESTABLISHING SESSION OF GROUP COMMUNICATION
1004
GENERATE SHARED SECRET CORRESPONDING TO SESSION, USING FIRST RNG
1006
DIGITALLY ENCAPSULATE SHARED SECRET IN VERSIONS
1008
DISTRIBUTE RESPECTIVE VERSION TO CORRESPONDING CLIENTS
EXECUTED AT TRUST PROVIDER(S)
1010
DECAPSULATE RESPECTIVE VERSION OF SHARED SECRET, USING RESPECTIVE PRIVATE KEY
1012
OBTAIN IDENTIFICATION CREDENTIAL INDICATIVE OF USER'S AUTHORIZATION TO USE SHARED SECRET
1014
DERIVE CRYPTO-PRODUCTS, USING SECOND RNG TO WHICH THE SHARED SECRET AND IDENTIFICATION CREDENTIAL ARE INPUTS
1016
EMPLOY CRYPTO-PRODUCTS TO ENCRYPT OUTGOING COMMUNICATION AND DECRYPT INCOMING COMMUNICATION
EXECUTED AT EACH CLIENT

Fig. 10

INTERNATIONAL SEARCH REPORT

International application No
PCT/FI2025/060124

A. CLASSIFICATION OF SUBJECT MATTER

INV. H04L9/08
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Alwen Joël ET AL: "How Multi-Recipient KEMs can help the Deployment of Post-Quantum Cryptography", , 12 April 2024 (2024-04-12), pages 1-15, XP093349724, Retrieved from the Internet: URL:https://csrc.nist.gov/presentations/2024/how-multi-recipient-kems-help-deploy-pqc	16
Y	section 1	1-15, 17-28
Y	WO 2023/049372 A1 (APPLE INC [US]) 30 March 2023 (2023-03-30) column 8; figures 4a, 4b	1-15, 17-28

-/--

[x] Further documents are listed in the continuation of Box C.

[x] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance;; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance;; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 December 2025	19/01/2026

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Billet, Olivier

1

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No
PCT/FI2025/060124

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 7 234 063 B1 (BAUGHER MARK [US] ET AL) 19 June 2007 (2007-06-19) paragraphs [0031] - [0034]; figures 4,5 -----	1-15, 17-28

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/FI2025/060124

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 2023049372 A1	30-03-2023	CN 118020270 A	10-05-2024
		DE 112022004542 T5	01-08-2024
		US 2023093992 A1	30-03-2023
		WO 2023049372 A1	30-03-2023
US 7234063 B1	19-06-2007	US 7234058 B1	19-06-2007
		US 7234063 B1	19-06-2007