Digital Object Identifier

One Time User Key: a user-based secret sharing XOR-ed model for multiple user cryptography in distributed systems

STEFANO GALANTUCCI[1], DONATO IMPEDOVO[2] (Senior Member, IEEE) and GIUSEPPE PIRLO[3] (Senior Member, IEEE)
[1]University of Bari "Aldo Moro" - Department of Computer Science (e-mail: stefano.galantucci@uniba.it)
[2]University of Bari "Aldo Moro" - Department of Computer Science (e-mail: donato.impedovo@uniba.it)
[3]University of Bari "Aldo Moro" - Department of Computer Science (e-mail: giuseppe.pirlo@uniba.it)

Research activity co-financed by "Fondo Europeo di Sviluppo Regionale Puglia POR Puglia 2014 - 2020 - Axis I - Specific Objective 1a - Action 1.1 (RD) - Project Title: CyberSecurity and SOC product Suite" by BV TECH S.p.A. (CUP/CIG B93G18000040007)

ABSTRACT The generation of encrypted channels between more than two users is complex, as it is necessary to share information about the key of each user. This problem has been partially solved through the secret sharing mechanism that makes it possible to divide a secret among several participants, so that the secret can be reconstructed by a well-defined part of them. The proposed system represents an extension of this mechanism, since it is designed to be applied systematically: each user has his/her key, through which temporary keys (One Time User Keys) are generated and are used to divide the secret, corresponding to the real encryption key. The system also overcomes the concept of numerical threshold (i.e., at least n participants are required to reconstruct the secret), allowing the definition, for each encryption, of which users can access and which specific groups of users can access. The proposed model can be applied both in distributed user-based contexts and as an extension of cryptographic functions, without impacting the overall security of the system. It addresses some requirements of the European Union Council resolution on encryption and also provides a wide possibility of applications in user-based distributed systems.

INDEX TERMS Secret Sharing, One Time Key, Cryptography, Secret Splitting, Authentication, Multiple Cryptography.

I. INTRODUCTION

THE secret sharing mechanism is a cryptographic tool designed to divide information among several participants. Thus it is possible to reconstruct the secret by combining the information held by a predetermined number of participants. This mechanism has undergone several formulations over the years, which have established different secret sharing schemes and models for reconstruction [1][2][3][4][5].
Generating encrypted channels between more than two users is complicated in both symmetric and asymmetric encryption schemes. In asymmetric cryptography, each user has its private and public key, but generating a cryptographic channel between more than two users is not explicitly provided. In symmetric encryption models, data encryption is done through a single key, but the key is unrelated to a specific user (as there is no concept of a private/public key), and generating a multi-user channel requires agreement among them in choosing or creating the key. If it is intended to generate a multi-user encrypted channel with symmetric encryption, a different key must be used for each channel; if this key is stored, the information is exposed to the inherent risk due to the storage itself. The model that this work proposes provides an operational middle ground between asymmetric and symmetric encryption, aiming to overcome the limitations mentioned above and allowing easily to have encrypted channels between multiple users, and accessing the channel always and only through a single key, which will always be the same for the user, regardless of the number of channels on which it is applied. This paper proposes a new formal model and a possible implementation of this, which evolves from the classic `XOR` secret sharing mechanism and acts as a meta-cryptographic model. This work is not a new encryption/decryption algorithm, but a model that relies on existing algorithms, thus ensuring the security of the algorithms already evaluated by the scientific community. This model, designed to be applied in a systematic manner, allows

the creation of encrypted channels between multiple users. The main idea is to assign to each user a key, which is the only one in his/her possession and will always remain the same. Encrypted channels are then generated between multiple users, encrypted with a single key, which will be traced to the keys of the individual users involved, so that it is possible to derive the real encryption key from the key of a single user and some additional information. Applying this model, i.e., adding additional decryption keys, does not impact the overall security of the system, as will be demonstrated in section V of this article.

Accordingly, the main contributions of this work are:

- Proposal for a formal meta-cryptographic model that allows the generation of encrypted channels between more than two users in a simple manner;
- Implementation of this model through an approach `XOR`-ed secret sharing and One Time User Key;
- Extension of this model to groups of keys, overcoming the concept of the (t, n) thresholds of secret sharing;
- Evaluate model security and formally demonstrate that the addition of the model does not impact system security.

The presented model is designed to be applied in distributed file systems, in which it is possible to manage encrypted files and grant them access to a predefined list of users. Recently it has emerged that the European Union [6] wishes to provide cryptographic tools, that can be bypassed by government authorities, to fight terrorism and crime. Such an application must not impact the security of the cryptographic algorithm: adding flaws or backdoors in the algorithms can be dangerous, as those who manage to exploit such flaws can break a cryptographic system that, it seems, should become a new standard.

The proposed system can be applied to this end, but without affecting the overall safety of the system.

This paper is structured as follows: section 2 proposes some similar approaches; section 3 describes the proposed model; section 4 provides the Python implementation code of the model, a complete example of application and a performance evaluation; section 5 gives a detailed analysis of the system security; section 6 provides some suggestions for application possibilities, as well is the conclusions.

II. RELATED WORK

Meng et al. [7] propose a secret sharing scheme based on the Chinese remainder theorem, developing their previous work [8], which proposes a general scheme of secret sharing of type (t,n) that solves some problems of the Shamir's scheme [1]. This work analyzes why the Proactive Secret Sharing (PSS) scheme, i.e. a scheme in which users can refresh actions without changing the secret, used over an integer ring cannot work effectively when shares are refreshed too many times. It also utilizes the Asmuth-Bloom (t,n) threshold SS to propose a PSS scheme over the integer ring, and the Ning et al. [8] ideal (t,n) threshold SS for a PSS scheme over a polynomial ring. Deepika et al. [9] provide two approaches of a secret sharing scheme, that use an algorithm composed of `XOR` operations applied to the Gray code: a 7-out-of-7 scheme and a 3-out-of-7 scheme. Singh et al. [10] use the Chinese remainder theorem to distribute a number of secrets, providing a number of parts of the secret equal to the number of secrets divided. The participants are thus divided into several levels, and it is possible to reconstruct the secrets at any of these levels, provided that the threshold (t,n) is respected. Phiri et al. [11] propose, again with a view to a secret sharing with threshold (t,n), the application of Lagrange's polynomial interpolation with the aim of obtaining the fixed values of output for the function of reconstruction of the secret. Such an interpolation model can be a valid implementation for the function ω in this paper, provided that the possibility of a direct attack on the polynomial using the One Time User Key scheme is analyzed. Works [7][8][9][10][11], which make use of thresholds of type (t,n), are designed to be applied in specific cases: it is not possible to define different thresholds within the same secret, equating the users. The proposed work allows the evolution of this scheme, using groups of keys, which define specifically which groups of users are sufficient to decrypt, and these groups do not have to be composed of the same number of users. The previously mentioned papers are therefore aimed at the application in specific instances of the problem, whereas the present work is designed for general and repeated application.

D'Arco et al. [12] propose a probabilistic method for secret sharing. This model uses a scheme of type (t,n), but with n=∞, shifting the problem from dividing the secret into valid subsets to generating t sufficient parts to reconstruct it. The approach uses access structures, allowing the thresholds of secret sharing to be extended to potentially infinity, but still equating users. Unlike our work, paper [12] does not allow for the possibility of varying parameter t for specific subsets of users. On the other hand parameter n is free.

Boyle et al. [13] introduce the notion of Function Secret Sharing (FSS), which applies the secret sharing mechanism to a binary input function, which generalizes Distributed Point Functions (DPF), i.e., a primitive created by Gilboa et al. [14], whose field of application is private information retrieval. The FSS scheme provides a way to split the function into different keys. Each key allows the owner to generate a standard secret share for the evaluation of the function. An FSS can be used as a method for performing a secret sharing, obtained by the sum of these functions. This scheme is improved and extended by Boyle et al. in [15], adding new FSS constructions from one-way functions via a new operation, DPF Tensor, that combines FSS schemes. The method proposed in [13] and [15] shapes secret sharing using functions as basic elements, unlike the present work that uses One Time generated keys as the main element. The secret sharing proposed in the cited works is of the additive type, whereas the reconstruction in this work is done through the operation of `XOR`.

Ding et al. [16] use a class of two-weight and three-weight codes as an application for secret sharing schemes, since any

linear code on a Galois Field (of odd prime elements) can be used for such an application. That work has several points in common with the proposal in this paper, in that it discusses a subset of participants collaborating for decryption. However, it does not provide an explicit possibility to define subsets in a completely arbitrary way.
None of the works examined previously permit the linking of the secret to user information. The received part of the secret must be memorized for what it is, and there is no way to reconstruct this information starting from other information already known to the user. These works are therefore to be considered secret-based and not user-based.
Hsu et al. [17] have developed UMKESS, a user-oriented secret sharing protocol, which uses a dealer, called Key Generation Center (KGC). This protocol uses secret sharing, instead of RSA asymmetric encryption, as a tool for broadcast encryption. Our work uses an approach similar to that of UMKESS. In fact, in both there is a dealer (called KGC in UMKESS) which is responsible for arbitrating the system. The main difference between this work and [17] lies in the way the secret is divided: UMKESS uses Shamir's polynomial (t, n) secret sharing scheme, whereas in the present work a theoretical model is first proposed, whose implementation of the theoretical function ω can vary. In the proposed implementation a multiple XOR-ed secret sharing on the same secret is used. UMKESS has a similar conceptual approach to the present work in that the system is user-based and designed to be applied systemically. However, Mitchell et al [18] have proved that UMKESS does not work in some cases and is generally insecure.

III. PROPOSED METHOD

A. FORMAL MODEL

The goal is to achieve a system that allows encryption between multiple users, each with their own key. The system can be defined through a function. Let K be the set of possible keys for a certain encryption/decryption algorithm $\mathfrak{C}$. K can be divided into two sets: K^+, the set of valid keys for decrypting a specific channel, and K^-, the set of invalid keys.

$$K^+ \cap K^- = \emptyset \tag{1}$$

$$K^+ \cup K^- = K \tag{2}$$

Set I is the set of the users in the system; it is similarly divided into I^+ (authorized users) and I^- (unauthorized users). Encryption will be performed in a standard way by a single key, called **real key** $k_* \in K$. The system must generate a function ω in K, that, if it receives as input the key of an authorized user, then returns the real key:

$$\omega : K \times I \rightarrow K \tag{3}$$

$$\forall k \in K^+, i \in I^+ \quad \omega(k, i) = k_* \tag{4}$$

$$\forall k \in K^-, i \in I^- \quad \omega(k, i) \neq k_* \tag{5}$$

TABLE 1. Table of notations used in the formal model, in One Time User Key and in the model extension to groups of valid keys

Notation definition	
K	Set of all possible keys for a certain cryptographic algorithm
K^+	Set of valid keys in a specific multiple user cryptography channel
K^-	Set of invalid keys in a specific multiple user cryptography channel
k_*	Real key used as the encryption/decryption key in the standard cryptographic algorithm
I	Set of all users in the system
I^+	Set of authorized users in a specific multiple user cryptography channel
I^-	Set of unauthorized users in a specific multiple user cryptography channel
$\omega(k,i)$	Transformation function of the key k belonging to the user i into the real key k_* if user i is authorized
$\oplus$	XOR operation
$k_i^{\oplus}$	Part of key obtained by XOR-ed secret sharing of user key k_i and real key k_*
$K^{\oplus}$	Set of all key parts $k_i^{\oplus}$
$h(m)$	Standard cryptographic hash function on message m
j	Salts number for generating a One Time User Key for a user
s_{i_n}	n-th salt to be applied in $\mathfrak{H}_i$ for user i
S_i	Set of salts to be applied in $\mathfrak{H}_i$ for user i, composed of the various s_{i_n}
S	Set of all salts, composed of the various S_i
$*$	Concatenation function
$\mathfrak{H}_i(k)$	Function that generates the One Time User Key from a key k through multiple hashing by applying S_i salts
HK^+	Set K^+ after applying $\mathfrak{H}_i$ to each key
$\mathfrak{C}(k,m)$	Decryption function of message m with key k using a standard cryptographic algorithm
$\Omega(k,i,m)$	Function that decrypts message m using user key k belonging to user i in the multiple user cryptography channel
$\tilde{K}^+$	Set of valid groups of keys in a specific multiple user cryptography channel (in groups of keys extension)
$\tilde{K}^-$	Set of invalid groups of keys in a specific multiple user cryptography channel (in groups of keys extension)
$\tilde{I}^+$	Set of authorized user groups in a specific multiple user cryptography channel (in groups of keys extension)
$\tilde{I}^-$	Set of unauthorized user groups in a specific multiple user cryptography channel (in groups of keys extension)
$k_{[X]}$	Group key corresponding to the group of keys X, obtained by XOR between the individual One Time User Keys of the group
$[K]^+$	Set of valid keys K^+ in the context of the groups of keys extension. Set of keys $k_{[X]}$ obtained from the individual key groups
$k_{[X]}^{\oplus}$	Part of key obtained by XOR-ed secret sharing of group key $k_{[X]}$ and real key k_*
$\tilde{\omega}(X,Y)$	Transformation function of the keys in X belonging to the users Y into the real key k_* if the group Y of users is authorized
$\tilde{\Omega}(X,Y,m)$	Function that decrypts message m using user keys X belonging to users in group Y in the multiple user cryptography channel (with groups of keys extension)

Any implementation of ω that ensures conditions (4) and (5) and does not impact system security is fine for the purposes of this paper.

B. MULTIPLE XOR-ED SECRET SHARING APPROACH

A valid implementation is possible by defining an XOR-ed secret sharing for each of the keys in K^+:

$$K^+ = \{k_1, k_2, ..., k_n\} \tag{6}$$

$$\forall i \in [1, n] \quad k_i^{\oplus} := k_* \oplus k_i \tag{7}$$

where $\oplus$ is the XOR operation between two keys.
Index i represents the user, to whom the key is assigned: key k_1 will be assigned to user 1 and so on. Let all the key parts obtained by $k_i^{\oplus}$ be grouped into a set ($K^{\oplus}$):

$$K^{\oplus} = \{k_1^{\oplus}, k_2^{\oplus}, ..., k_n^{\oplus}\} \tag{8}$$

So, function ω can then be implemented as follows:

$$\omega(k, i) := k \oplus k_i^{\oplus} \tag{9}$$

This implementation ensures the validity of (4) for the properties of XOR operation. (5) is respected since when a malicious user tries to submit his/her key, he/she will be forced to identify himself/herself (i.e., in the proposed formal model he provides his value i), and will therefore have to pretend to be another legitimate user, submitting his/her key $k_{i'}^{\oplus}$. However, the only case in which function ω will return value k_* for the user with key $k_{i'}$ is if he/she submits the correct key k_i, due to the properties of the XOR:

$$\omega(k_{i'}, i) = k_{i'} \oplus k_i^{\oplus} = k_* \quad \Longleftrightarrow \quad k_{i'} = k_i \in K^+ \tag{10}$$

To provide a comparison with asymmetric encryption mechanisms, it can be considered that, for each user, k_i and $k_i^{\oplus}$ represent, respectively, private key and public key analogs. k_i represents the private key since it is known only to the user, whereas $k_i^{\oplus}$, the public key, must be known to all to proceed with decryption. As in asymmetric encryption schemes, the combination of the two keys enables decryption, in this case, achieved in the generation of the real key k_* through the XOR operation between the two keys.

C. ONE TIME USER KEY

Since each key k_i must be the only key in the user's possession, using the previous scheme creates the problem that each user $u \in I^+$ can derive the other keys in the same set by the following algorithm:

1) Calculate $\omega(k, u) = k_u \oplus k_u^{\oplus} = k_*$
2) Calculate, for all the others i in I^+ : $\omega(k_*, i) = k_* \oplus k_i^{\oplus} = k_i$

This problem can be solved by generating, for each encrypted channel, a One Time User Key (OTUK) for each authorized user. These keys must be generated from the user keys so that an OTUK key can be derived from the user key, but not the contrary.
The solution is provided by cryptographic hash functions.

The chosen hash function h should have an output size compatible with the used cryptographic function $\mathfrak{C}$, since the output of h will be the key of $\mathfrak{C}$:

$$\text{Ran}(h) = K \tag{11}$$

For each user $i \in I^+$ a set of salts s_{i_j} will be generated (randomly), with $j \geq 1$:

$$S_i = \{s_{i_1}, s_{i_2}, ..., s_{i_j}\} \tag{12}$$

The j value (i.e., the number of salts to apply) can be different for each user and each encryption. Having different values of j involves only a change in the number of times the hash function must be applied. These salts will need to change for each cryptographic channel where user i intervenes.
Set S will be the set of salt groups:

$$S = \{S_1, S_2, ..., S_n\} \tag{13}$$

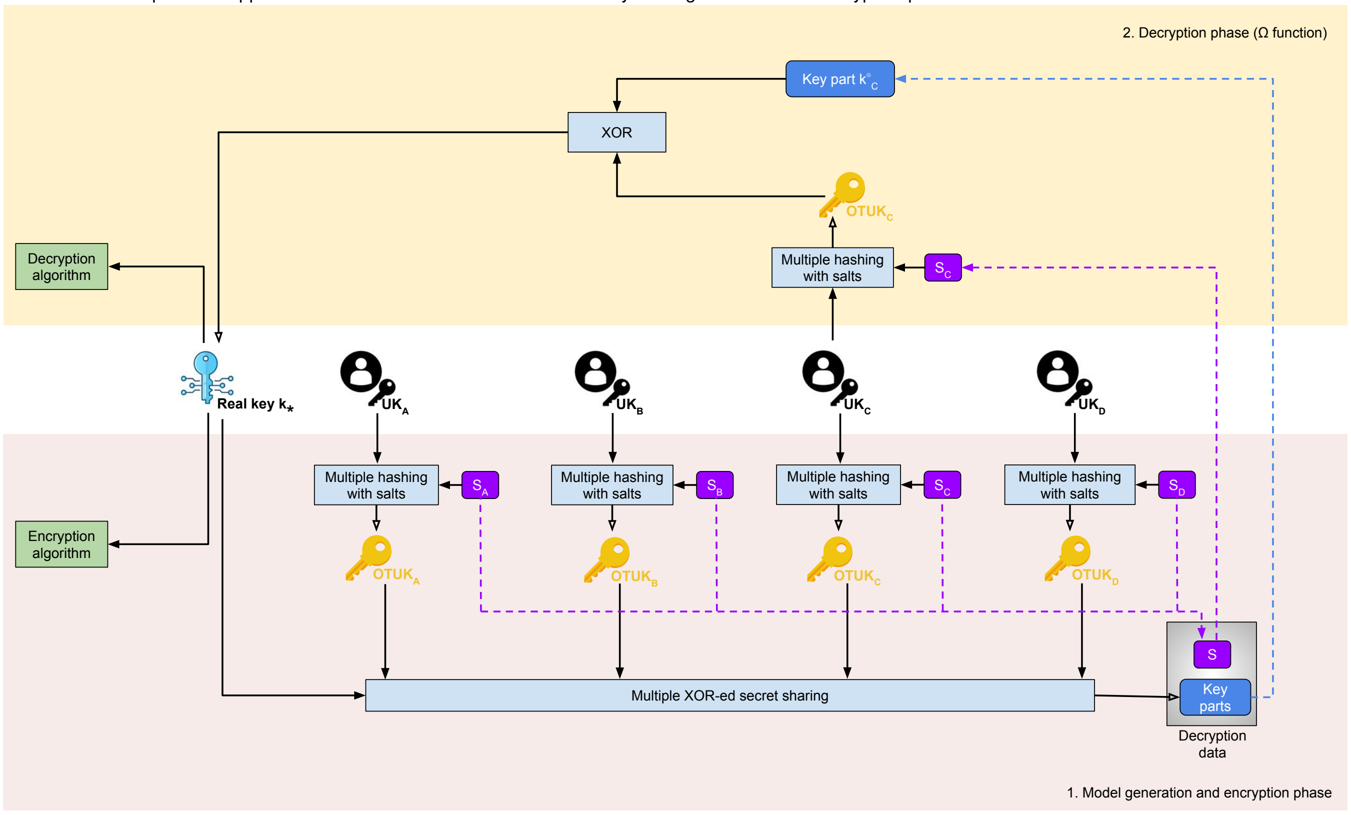
Let $*$ be the concatenation function. Function $\mathfrak{H}$ will be defined as follows:

$$\mathfrak{H} : K \times I \to K \tag{14}$$

$$\mathfrak{H}_i(k) := h(s_{i_j} * h(s_{i_{j-1}} * ...h(s_{i_2} * h(s_{i_1} * k))...)...) \tag{15}$$

Each salt can be obtained through a safe random generator. In particular, the set of salts must always be different for each new generation. It is crucial that the set of salts be consistently different for each new generation. In terms of security, it is sufficient for at least one of them to differ, but it is strongly recommended that the entire set of salts change.
A current timestamp can also be used as a salt: this allows it to vary with each new channel generation. Considerations for balancing time and safety (salt length) proposed in [19] can be evaluated. A value $j = 1$ is enough to provide system security, however, the use of $j \geq 3$ is suggested to avoid possible single hash inversions. The $\mathfrak{H}$ function will generate the keys that will be used for encryption, but these will only be correctly computable by the user in possession of the key on which the function was executed. The output of the $\mathfrak{H}$ function is a new generated key, and it will be a One Time User Key because it will be used only for the specific channel.
If one of the users in I^+ obtains the other K^+ keys, and K^+ is composed only of One Time User Keys, he/she will have available keys that can be used only for that specific channel (where he/she already has his own, so the others are useless) and he will not be able to get the user keys because of the non-invertibility of the hash function.
One Time User Key has some points in common with the One Time Pad [20] mechanism, as both use a different key for each encryption. The main difference between them is that One Time Pad uses keys that do not correlate with each other. However, in the One Time User Key model, each of these keys is obtained from the user key, so there is a one-way function $\mathfrak{H}$ that produces a One Time key from k_i. Moreover, in One Time Pad the key changes constantly to avoid being attacked by Kasiski's method [21]. In One Time User Key the key changes for each channel because, as it can ben derived

FIGURE 1. Example of an application of the model with One Time User Key. The figure shows the decryption phase for user C.


2. Decryption phase (Ω function)
Key part k⊕C
XOR
OTUKC
Multiple hashing with salts
SC
Decryption algorithm
Real key k*
UKA
UKB
UKC
UKD
Encryption algorithm
Multiple hashing with salts
SA
SB
SC
SD
OTUKA
OTUKB
OTUKC
OTUKD
S
Key parts
Multiple XOR-ed secret sharing
Decryption data
1. Model generation and encryption phase

from other valid keys, it could be reused maliciously.
The One Time User Key mechanism is equivalent to the derived key mechanism (in fact it is not strictly necessary that the user key size is compatible with the cryptographic function keys) but differs from the derived key because the salts must be changed for each different encryption.
Then, to have an effective system, set HK^+ is defined from set K^+:

$$K^+ = \{k_1, k_2, ..., k_n\} \tag{16}$$

$$HK^+ = \{\mathfrak{H}_1(k_1), \mathfrak{H}_2(k_2), ..., \mathfrak{H}_n(k_n)\} \tag{17}$$

Let $\mathfrak{C}(k, m)$ be the decryption function of the message m with key k.
The Ω function that will decrypt a message m starting from user key k will be:

$$\Omega(k, i, m) = \mathfrak{C}(\omega(\mathfrak{H}_i(k), i), m) \tag{18}$$

In this case, the ω function is interpreted to operate with $k_i^{\oplus}$ computed from HK^+.
Therefore, for each encrypted channel, it will be necessary to store information for decryption: the enabled users, the set of salts, and the parts of the key (computed on HK^+) $\{I^+, S, K^{\oplus}\}$.
Fig. 1 shows the complete working of the system: One Time User Keys are generated ($\mathfrak{H}$ function) from the keys of authorized users A, B, C and D using randomly generated salts (S_A, S_B, S_C, S_D); these OTUKs are combined with the real key k_* in a multiple XOR-ed secret sharing, generating the key parts ($K^{\oplus}$ set); the decryption data (S and the key parts) are stored; in the decryption phase, the user C takes his/her salts S_C from the decryption data, generates his/her One Time User Key and combines it by XOR with the key part $k_C^{\oplus}$ (that is stored in decryption data), obtaining the real key k_*.
The addition of the One Time User Key mechanism changes the analogy between the keys and the idea of private and public key. The public key $k_i^{\oplus}$ (computed on HK^+) is unchanged, since it needs to be public; k_i continues to represent the private key available to the user. However, the real private key of the single encryption is $\mathfrak{H}_i(k_i)$, which is derived directly from k_i and the salts.

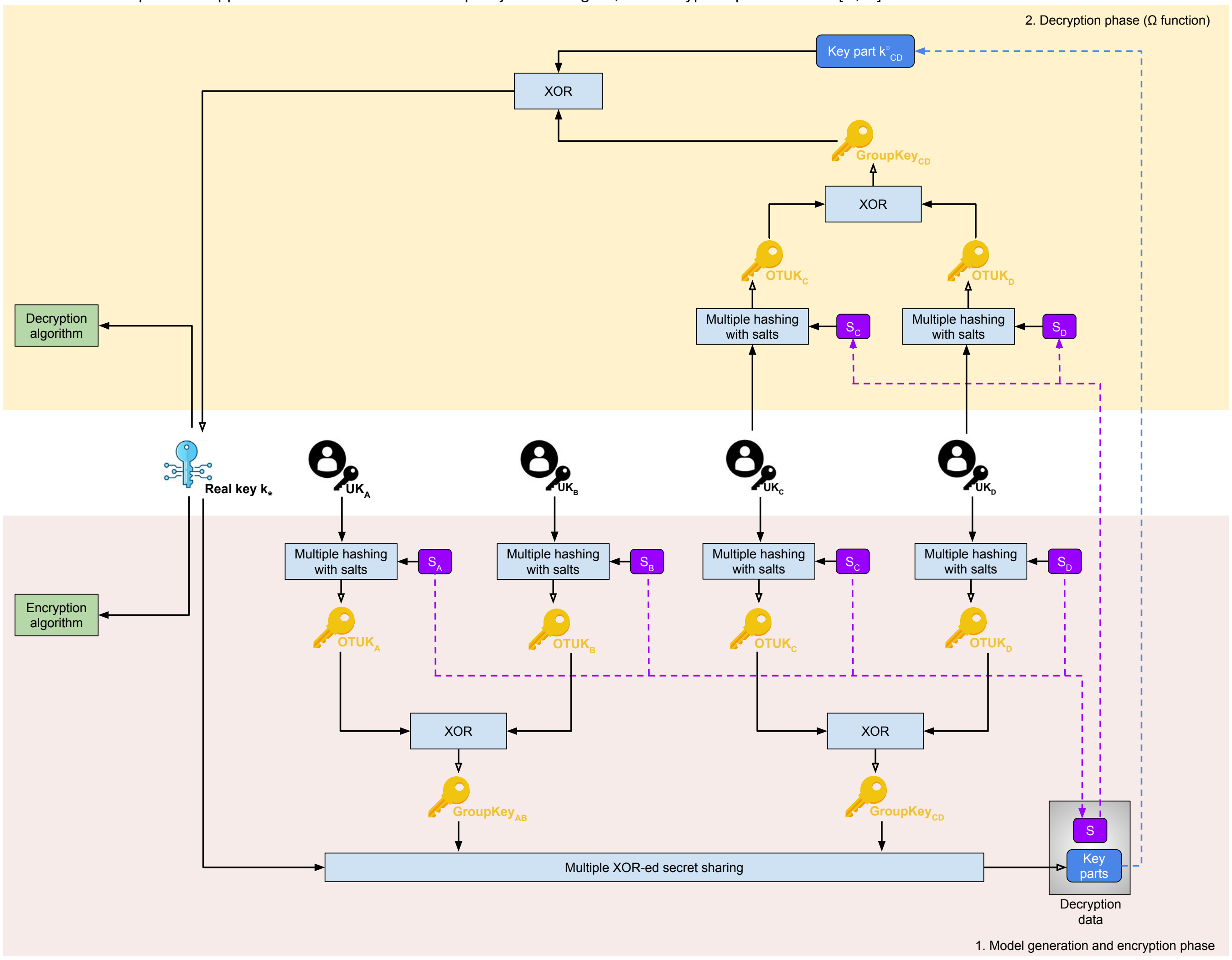
D. GROUPS OF VALID KEYS

The model can be extended from single keys to groups of valid keys by transforming set $K^+ \subset K$ into $\tilde{K^+} \subset P(K)$.
The algorithm will return the real key when all the keys of a certain valid set are submitted. A single $k \in K$ can be part of zero, one or more $\tilde{K^+}$ elements.
This extension involves rephrasing (4) and (5) in the following manner, considering $\tilde{K^+}$ as the set of single sets of valid keys, and $\tilde{K^-} = P(K) - \tilde{K^+}$

$$\tilde{\omega} : P(K) \times P(I) \rightarrow K \tag{19}$$

$$\forall X \in \tilde{K^+}, Y \in \tilde{I^+} \quad \omega(X, Y) = k_* \tag{20}$$

FIGURE 2. Example of an application of the model with Group Keys. In the figure, the decryption phase for the [C, D] users is shown.



$$\forall X \in \tilde{K^-}, Y \in \tilde{I^-} \quad \omega(X, Y) \neq k_* \tag{21}$$

The result is obtained by returning the problem from the domain of valid key sets to single valid keys. It is then necessary to obtain a single valid key from each set in $\tilde{K^+}$. This can be achieved, again, by using secret splitting:

$$X = \{\tilde{k_1}, \tilde{k_2}, ..., \tilde{k_m}\} \in \tilde{K^+} \tag{22}$$

$$k_{[X]} = \mathfrak{H}_1(\tilde{k_1}) \oplus \mathfrak{H}_2(\tilde{k_2}) \oplus ... \oplus \mathfrak{H}_m(\tilde{k_m}) \tag{23}$$

Then set $\tilde{K^+}$ will be represented and transformed into set K^+ (hereafter referred to as $[K]^+$ to differentiate the notations) by the following conversion rule:

$$[K]^+ := \{k_{[Y]} | Y \in \tilde{K^+}\} \tag{24}$$

The key parts in $K^\oplus$ will then be derived from the keys $k_{[1]}, k_{[2]}, ..., k_{[m]}$. Function ω is rewritten as follows:

$$X' = \{k_a, k_b, ..., k_z\} \in \tilde{K^+} \tag{25}$$

$$Y' = \{a, b, ..., z\} \tag{26}$$

$$k^\oplus_{[X']} = (k_* \oplus \mathfrak{H}_a(k_a) \oplus \mathfrak{H}_b(k_b) \oplus ... \oplus \mathfrak{H}_z(k_z)) \in K^+ \tag{27}$$

$$\tilde{\omega}(X', Y') := k^\oplus_{[X']} \oplus (\mathfrak{H}_a(k_a) \oplus \mathfrak{H}_b(k_b) \oplus ... \oplus \mathfrak{H}_z(k_z)) \tag{28}$$

and function Ω becomes:

$$\tilde{\Omega}(X', Y', m) = \mathfrak{C}(\tilde{\omega}(X', Y'), m) \tag{29}$$

Fig. 2 shows the complete working of the system using Groups Keys: the authorized users are the pairs $[A, B]$ and $[C, D]$; a One Time User Key is generated for each of the four users involved using the randomly generated salts (S_A, S_B, S_C, S_D); the OTUKs of users A and B are combined through XOR generating the Group Key AB and the same happens for users C and D; these Group Keys are used for multiple XOR-ed secret sharing with the real key k_*, generating the key parts that will be stored together with the salts; in the decryption phase, users C and D generate their own OTUKs using the salts present in the stored data (S_C and S_D) and combine these by XOR, obtaining the Group Key CD; this Group Key is combined by XOR with the key part $k^\oplus_C D$, obtaining the real key k_*.

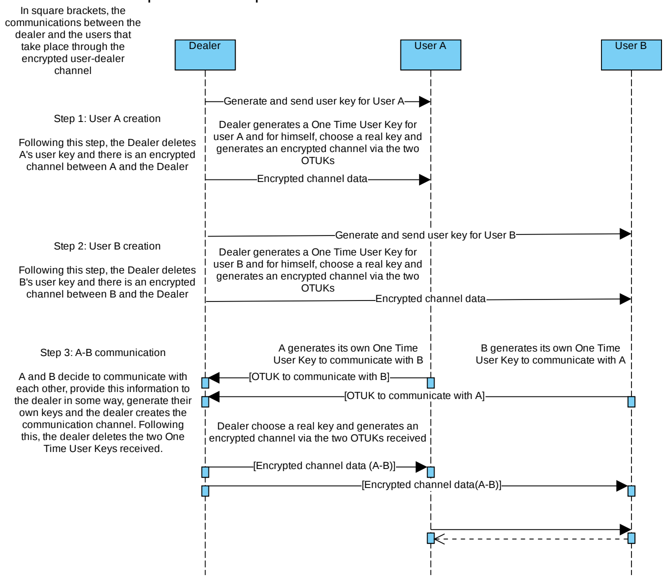
E. DEALER AND KEY MANAGEMENT

The implementation of the model needs an arbitrator (dealer). The dealer must provide at least the following functions:

- Accrediting users in the system;
- Generating for each cipher the real key k_*;
- Securely retrieving a single One Time User Key from each user;
- Generating sets $HK^+, K^\oplus, \tilde{K}^+$ from the One Time User Keys;
- Providing individual parties with decryption information for each encrypted channel;
- Regenerating data in the case of change in access permissions of individual users (i.e. in the case of I^+ and I^- changes on a single encryption).

Each user must be equipped with an application that allows him to generate his/her own One Time User Keys and transfer them to the dealer. Each user must also be able to read the decryption information of each channel and must be able to apply the ω and Ω functions. Users must be able to communicate confidently with the dealer.
The dealer doesn't necessarily have to be a third-party trustee, but could also be impersonated from time to time by a different user, who acts as the coordinator. The detailed implementation of the dealer is not the object of the present theoretical study, as these features would have quite different applications, depending on the purposes for using this model. For example, where the model is to be used in a distributed file system, it is easily handled by a trusted third-party dealer acting as a super-user within the system. If the model is to be used for the implementation of the requirements of the European Union Council resolution on cryptography, the situation has to be treated in a completely different way, since the main goal would be to generate trusted One Time User Keys starting from the user key of government institutions. Therefore, it would be necessary to develop a library or a hardware implementation that hooks up to external generators through the network, or to pre-packaged generators (such as bank OTPs).
A possible protocol for the dealer (Fig. 3) that acts like a Public Key Infrastructure: the dealer is a user of the system, he/she generates his/her key and acts as a third party. The dealer issues user keys to individual users and, immediately after, generates through the One Time User Key model an encrypted channel between the user just created and the dealer. This is possible because the dealer knows the user key, since he/she issued it, therefore, he/she can generate a One Time User Key for that user and couple it to a One Time User Key linked to his key. Once everything is done, he/she no longer needs to store the user key of the new user. By having secure channels between the dealer and each user, the dealer can exercise its arbitrage functions by requesting and receiving One Time User Keys from individual users. In the case of key groups, it is still the dealer who builds the group keys. There is no need to authenticate the dealer to the users, because the users never transmit the user key to the dealer, but at most the One Time User Keys, which, even in clear text, have no value if not used in encryption. A fake malicious dealer can then ask the user for an infinite number of One Time User Keys, however, will not be able to derive the user key or keys of other encryptions because of the non-invertibility of the cryptographic hash function. In any case, the dealer communicates with the users only through the generated encrypted channels, so a malicious dealer cannot intrude into a channel managed by a trusted dealer, since the intruder, to communicate with the user, would have to know the dealer-user channel key. Similarly, the dealer never transmits critical information to the user, except when issuing the key, so the user does not need to be authenticated to the dealer. The only problem is if the dealer who emits the user key is malicious: the dealer could store the information of the user key, One Time User Keys, or real keys and use it to decrypt the channels generated later between users.

FIGURE 3. Example of dealer protocol


In square brackets, the communications between the dealer and the users that take place through the encrypted user-dealer channel
Dealer
User A
User B
Generate and send user key for User A
Step 1: User A creation
Following this step, the Dealer deletes A's user key and there is an encrypted channel between A and the Dealer
Dealer generates a One Time User Key for user A and for himself, choose a real key and generates an encrypted channel via the two OTUKs
Encrypted channel data
Generate and send user key for User B
Step 2: User B creation
Following this step, the Dealer deletes B's user key and there is an encrypted channel between B and the Dealer
Dealer generates a One Time User Key for user B and for himself, choose a real key and generates an encrypted channel via the two OTUKs
Encrypted channel data
Step 3: A-B communication
A generates its own One Time User Key to communicate with B
B generates its own One Time User Key to communicate with A
A and B decide to communicate with each other, provide this information to the dealer in some way, generate their own keys and the dealer creates the communication channel. Following this, the dealer deletes the two One Time User Keys received.
[OTUK to communicate with B]
[OTUK to communicate with A]
Dealer choose a real key and generates an encrypted channel via the two OTUKs received
[Encrypted channel data (A-B)]
[Encrypted channel data(A-B)]

IV. PYTHON ALGORITHM IMPLEMENTATION

Here below we present the implementation in Python of the single functions constituting the model.

```
class Key:
  def __init__(self, keyData, users, keyType
     , salts):
    self.keyData = keyData
    self.users = users
    self.keyType = keyType
    self.salts = salts

def generateOTUK(userKey, salts):
  x = userKey.keyData
  for salt in salts:
    x = hashFunction(concatenation(salt, x))
  return Key(x, userKey.users, "OTUK", salts
     )
```

```
def generateGroupKey(OTUKs):
  key = zeros(key_size) #create a bytes
      0000...
  saltdb = {}
  usersset = set()
  for OTUK in OTUKs:
    key = XOR(key, OTUK.keyData)
    saltdb[OTUK.users] = OTUK.salts
    usersset.add(OTUK.users)
  return Key(key, sorted(usersset), "
      GroupOTUK", saltdb)

def generateCryptoChannel(realKey, groupKeys
    ):
  #IMPORTANT: Each user must be assigned the
       same salts, even if he participates in
       more than one set of keys
  decipherData = {"keyparts": {}, "salts":
      {}}
  for groupKey in groupKeys:
    key = XOR(realKey.keyData, groupKey.
        keyData)
    users = repr(groupKey.users)
    decipherData["keyparts"][users] = key
    for user in groupKey.users:
      decipherData["salts"][user] = groupKey
          .salts[user]
  return decipherData

def decryptData(data, userKeys, decipherData
    ):
  OTUKs = []
  for userKey in userKeys:
    OTUKs.append(generateOTUK(userKey,
        decipherData["salts"][userKey.users])
        )
  grKey = generateGroupKey(OTUKs)
  realKeyData = XOR(grKey.keyData,
      decipherData["keyparts"][repr(grKey.
      users)])
  decrypt(data, realKeyData)
```

The example code for generating an encrypted channel between a user A and a user B ($I^{+} = \{A, B\}$) is

```
# >>> 1. User keys generation
ukey_A = randomKey("UserA")
ukey_B = randomKey("UserB")

# >>> 2. Real key generation and data
    encryption
realK = randomKey("THEREALKEY")
encryptedData = encrypt(data, realK.keyData)

# >>> 3. Generation of an OTUK for each user
# j is number of salts
OTUK_A = generateOTUK(ukey_A, generatesalts(
    j, salt_size))
OTUK_B = generateOTUK(ukey_B, generatesalts(
    j, salt_size))

# >>> 4. Generation of a Group Key for each
    OTUK
# Group Keys in this case are equivalent to
    the OTUKs because allowed groups of users
     are [A] and [B]
group_A = generateGroupKey([OTUK_A])
group_B = generateGroupKey([OTUK_B])

# >>> 4. Generation of encrypted channel
    data
decipherAB = generateCryptoChannel(realK, [
    group_A, group_B])

# >>> 5. Deciphering by users

#USER A
decryptData(encryptedData, [ukey_A],
    decipherAB)

#USER B
decryptData(encryptedData, [ukey_B],
    decipherAB)
```

The example code for generating an encrypted channel authorizing $\tilde{I^{+}} = \{\{A, B\}, \{C, D\}, \{E\}\}$ is

```
# >>> 1. User keys generation
ukey_A = randomKey("UserA")
ukey_B = randomKey("UserB")
ukey_C = randomKey("UserC")
ukey_D = randomKey("UserD")
ukey_E = randomKey("UserE")

# >>> 2. Real key generation and data
    encryption
realK = randomKey("THEREALKEY")
encryptedData = encrypt(data, realK.keyData)

# >>> 3. Generation of an OTUK for each user
# j is number of salts
OTUK_A = generateOTUK(ukey_A, randomsalts(j,
     salt_size))
OTUK_B = generateOTUK(ukey_B, randomsalts(j,
     salt_size))
OTUK_C = generateOTUK(ukey_C, randomsalts(j,
     salt_size))
OTUK_D = generateOTUK(ukey_D, randomsalts(j,
     salt_size))
OTUK_E = generateOTUK(ukey_E, randomsalts(j,
     salt_size))

# >>> 4. Generation of a Group Key for each
    group
group_AB = generateGroupKey([OTUK_A, OTUK_B
    ])
group_CD = generateGroupKey([OTUK_C, OTUK_D
    ])
group_E = generateGroupKey([OTUK_E])

# >>> 4. Generation of encrypted channel
    data
decipherData = generateCryptoChannel(realK,
    [group_AB, group_CD, group_E])

# >>> 5. Deciphering by users groups
#This is just an inshore example, actually
    to perform decryption, individual users
    generate their own OTUKs and combine them
     via XOR

#USER A-B
decryptData(encryptedData, [ukey_A, ukey_B],
     decipherData)
```

```
#USER C-D
decryptData(encryptedData, [ukey_C, ukey_D],
     decipherData)

#USER E
decryptData(encryptedData, [ukey_E],
    decipherData)
```

A. A COMPLETE APPLICATION EXAMPLE

Below is a complete application example. Keys and salts are expressed in the hexadecimal notation of their component bytes. The keys have a size of 256 bits (32 bytes); the salts are generated in groups of 3, each with a size of 10 bytes. The hashing algorithm used is `SHA256`.

Real Key

$k_* = \texttt{1b950c8f7669370b7c875c116bf7ed9833a4db32f56b007515bef42a3a1fab19}$

User Keys

$uk_A = \texttt{8d36c127bb7d012303460e3acaec175adf193428f0304564b19bd696c6c9a66d}$
$uk_B = \texttt{14e3e502710a39e84ecaaaafb18341d80dd51fb20efe2f5f84f9151d61219f0d}$
$uk_C = \texttt{da3037a8cddd0f3ace3b741534cb14c811d1c3d0895999d617afdf219af62151}$
$uk_D = \texttt{2c394bc4cc865c42a44e7b5ed71e186eefb90f9f0d17bb297be6885e07cecf04}$
$uk_E = \texttt{847ea5dc50d46ebdfa3aac5e08ec95bd1278dcdb88764be532b2a4cff604d785}$

Generation of One Time User Keys

User A
$S_A = \{\texttt{6f1d9a5993ebdea3261f}, \texttt{fc8ff4103551d72aaddd}, \texttt{4569c24eeebe54d4a631}\}$
$OTUK_A = \mathfrak{H}_A(uk_A) = \texttt{a11e6b457758ebad3895d758001356d73c5de06fd2f3771892c5bf610acb0307}$

User B
$S_B = \{\texttt{b8989ca2bda49aaec6d8}, \texttt{ce02819fea290ff5132e}, \texttt{4530b6988cbd55d1effe}\}$
$OTUK_B = \mathfrak{H}_B(uk_B) = \texttt{8af560fce41bd07aaa95ab548e3ec3e96c66c4fdbf7718318064b6c2e20b8698}$

User C
$S_C = \{\texttt{ea5c51826f804136a49b}, \texttt{6f19ce91885062466e1e}, \texttt{ee01fae38806f5a0f69c}\}$
$OTUK_C = \mathfrak{H}_C(uk_C) = \texttt{c6c956f3196e354a2e82030525813581eb23bc18da9cf79d4f694b3d3060bb14}$

User D
$S_D = \{\texttt{e86f0d1fe10ed91c0bb7}, \texttt{a336e71b5a254cdd2c36}, \texttt{313512cae5052bf021d5}\}$
$OTUK_D = \mathfrak{H}_D(uk_D) = \texttt{dd810e714959e1e58ddc78a23ecc36b8613d4aa079c87ceadfb60d2e2ee911cc}$

User E
$S_E = \{\texttt{57e7f6a2b6f2124beafe}, \texttt{3fe34d9e292485fc8abb}, \texttt{a652bb45a873906d5016}\}$
$OTUK_E = \mathfrak{H}_E(uk_E) = \texttt{25deee26ef08e42cb185461fe3ddeeda6d4ab82281ae27eb21a5e8dc8c056f3a}$

Generation of Group Keys

$\tilde{I^+} = \{\{A, B\}, \{C, D\}, \{E\}\}$

Group A-B
$k_{[A,B]} = OTUK_A \oplus OTUK_B = \texttt{2beb0bb993433bd792007c0c8e2d953e503b24926d846f2912a109a3e8c0859f}$

Group C-D
$k_{[C,D]} = OTUK_C \oplus OTUK_D = \texttt{1b4858825037d4afa35e7ba71b4d03398a1ef6b8a3548b7790df46131e89aad8}$

Group E
$k_{[E]} = OTUK_E = \texttt{25deee26ef08e42cb185461fe3ddeeda6d4ab82281ae27eb21a5e8dc8c056f3a}$

Generation of set $K^{\oplus}$

Group A-B
$k^{\oplus}_{[A,B]} = k_{[A,B]} \oplus k_* = \texttt{307e0736e52a0cdcee87201de5da78a6639fffa098ef6f5c071ffd89d2df2e86}$

Group C-D
$k^{\oplus}_{[C,D]} = k_{[C,D]} \oplus k_* = \texttt{00dd540d265ee3a4dfd927b670baeea1b9ba2d8a563f8b028561b239249601c1}$

Group E
$k^{\oplus}_{[E]} = k_{[E]} \oplus k_* = \texttt{3e4be2a99961d327cd021a0e882a03425eee631074c5279e341b1cf6b61ac423}$

Decryption attempts

Using only key A - A tries to force deciphering by itself
$OTUK_A \oplus k^{\oplus}_{[A,B]} = \texttt{91606c739272e771d612f745e5c92e715fc21fcf4a1c184495da42e8d8142d81}$ ✗
$OTUK_A \oplus k^{\oplus}_{[C,D]} = \texttt{a1c33f4851060809e74cf0ee70a9b87685e7cde584ccfc1a17a40d582e5d02c6}$ ✗
$OTUK_A \oplus k^{\oplus}_{[E]} = \texttt{9f5589ecee39388af597cd568839559562b3837fa6365086a6dea397bcd1c724}$ ✗

Using only key B - B tries to force deciphering by itself
$OTUK_B \oplus k^{\oplus}_{[A,B]}$ = ba8b67ca0131dca644128b496
be4bb4f0ff93b5d2798776d877b4b4b30d4a81e ✗
$OTUK_B \oplus k^{\oplus}_{[C,D]}$ = 8a2834f1c24533de754c8ce2f
e842d48d5dce977e9489333050504fbc69d8759 ✗
$OTUK_B \oplus k^{\oplus}_{[E]}$ = b4be82557d7a035d6797b15a061
4c0ab3288a7edcbb23fafb47faa34541142bb ✗

Using only key C - C tries to force deciphering by itself
$OTUK_C \oplus k^{\oplus}_{[A,B]}$ = f6b751c5fc443996c0052318c
05b4d2788bc43b8427398c14876b6b4e2bf9592 ✗
$OTUK_C \oplus k^{\oplus}_{[C,D]}$ = c61402fe3f30d6eef15b24b35
53bdb20529991928ca37c9fca08f90414f6bad5 ✗
$OTUK_C \oplus k^{\oplus}_{[E]}$ = f882b45a800fe66de380190bada
b36c3b5cddf08ae59d0037b7257cb867a7f37 ✗

Using only key D - D tries to force deciphering by itself
$OTUK_D \oplus k^{\oplus}_{[A,B]}$ = edff0947ac73ed39635b58bfd
b164e1e02a2b500e12713b6d8a9f0a7fc363f4a ✗
$OTUK_D \oplus k^{\oplus}_{[C,D]}$ = dd5c5a7c6f07024152055f144
e76d819d887672a2ff7f7e85ad7bf170a7f100d ✗
$OTUK_D \oplus k^{\oplus}_{[E]}$ = e3caecd8d03832c240de62acb6e
635fa3fd329b00d0d5b74ebad11d898f3d5ef ✗

Using only key E - E lawfully accesses
$OTUK_E \oplus k^{\oplus}_{[E]}$ = 1b950c8f7669370b7c875c116bf
7ed9833a4db32f56b007515bef42a3a1fab19 ✓
for completeness:
$OTUK_E \oplus k^{\oplus}_{[A,B]}$ = 15a0e9100a22e8f05f0266020
607967c0ed54782194148b726ba15555eda41bc ✗
$OTUK_E \oplus k^{\oplus}_{[C,D]}$ = 2503ba2bc95607886e5c61a99
367007bd4f095a8d791ace9a4c45ae5a8936efb ✗

Using another OTUK for E - Someone has obtained this key, used in another encrypted channel, and tries to use it
S_{E_2} = {6cadf033d25b6b1aa38e, 4b2a0ae00c066
7c53cb7, acf5aa0fa48eaa506b04}
$OTUK_{E_2} = \mathfrak{H}_{E_2}(uk_E)$ = 3a3afab8d007ba0ddb17e
0fe708fe349627cab2d14d48645334d33ee63213
ac1
$OTUK_{E_2} \oplus k^{\oplus}_{[A,B]}$ = 0a44fd8e352db6d13590c0e39
5559bef01e3548d8c3be9193452ce67b1fe1447 ✗
$OTUK_{E_2} \oplus k^{\oplus}_{[C,D]}$ = 3ae7aeb5f65959a904cec7480
0350de8dbc686a742eb0d47b62c81d747b73b00 ✗
$OTUK_{E_2} \oplus k^{\oplus}_{[E]}$ = 047118114966692a1615faf0f8
a5e00b3c92c83d6011a1db07562f18d53bfee2 ✗

Using group key A-B - A-B lawfully accesses
$k_{[A,B]} \oplus k^{\oplus}_{[A,B]}$ = 1b950c8f7669370b7c875c116bf7
ed9833a4db32f56b007515bef42a3a1fab19 ✓
for completeness:
$k_{[A,B]} \oplus k^{\oplus}_{[C,D]}$ = 2b365fb4b51dd8734dd95bbafe97
7b9fe98109183bbbe42b97c0bb9acc56845e ✗
$k_{[A,B]} \oplus k^{\oplus}_{[E]}$ = 15a0e9100a22e8f05f02660206079
67c0ed54782194148b726ba15555eda41bc ✗

Using a new group key B-C - B and C try to force decipher by combining their keys
$k_{[B,C]} = OTUK_B \oplus OTUK_C$ = 4c3c360ffd75e5308
417a851abbff668874578e565ebefaccf0dfdffd
26b3d8c
$k_{[B,C]} \oplus k^{\oplus}_{[A,B]}$ = 7c423139185fe9ec6a90884c4e65
8ecee4da8745fd0480f0c812007600b4130a ✗
$k_{[B,C]} \oplus k^{\oplus}_{[C,D]}$ = 4ce16202db2b06945bce8fe7db05
18c93eff556f33d464ae4a6c4fc6f6fd3c4d ✗
$k_{[B,C]} \oplus k^{\oplus}_{[E]}$ = 7277d4a6641436174915b25f2395f
52ad9ab1bf5112ec832fb16e1096471f9af ✗

Using group key C-D - C-D lawfully accesses
$k_{[C,D]} \oplus k^{\oplus}_{[C,D]}$ = 1b950c8f7669370b7c875c116bf7
ed9833a4db32f56b007515bef42a3a1fab19 ✓
for completeness:
$k_{[C,D]} \oplus k^{\oplus}_{[A,B]}$ = 2b365fb4b51dd8734dd95bbafe97
7b9fe98109183bbbe42b97c0bb9acc56845e ✗
$k_{[C,D]} \oplus k^{\oplus}_{[E]}$ = 2503ba2bc95607886e5c61a993670
07bd4f095a8d791ace9a4c45ae5a8936efb ✗

B. PERFORMANCE

Table 2 shows the time performance in generating One Time User Keys. The hashing algorithm used is `SHA-256`. This performance is calculated on Google Colaboratory (without using hardware acceleration) and is inclusive of the random salt generation time. Each reported time is calculated as the arithmetic mean of 20,000 One Time User Key generations using the reference salt number-dimension configuration. It can be observed that as the number of salts increases (parameter j), there is a considerable increase in the generation time of the One Time User Key. The size of the salts does not prove to be a particularly relevant parameter, in comparison with the number of hashing operations to be performed. Therefore,

TABLE 2. Time performance in OTUK generation. Each value is in e-05 seconds.

		Salt size (bytes)					
		5	**10**	**20**	**30**	**40**	**50**
Salt number	**3**	1.148698	1.077464	1.172358	1.147355	1.292105	1.329993
	4	1.437464	1.413291	1.443406	1.505799	1.62866	1.709932
	5	1.75503	1.676196	1.689763	1.791142	2.053413	2.009451
	6	1.990707	1.969763	2.038242	2.156322	2.379203	2.358516
	7	2.288272	2.275149	2.334738	2.414774	2.700992	2.656931
	8	2.581429	2.583169	2.579329	2.77001	3.104515	3.0252
	9	2.854338	2.822433	2.881157	3.112465	3.406454	3.38677
	10	3.188862	3.136392	3.223689	3.373617	3.663958	3.878438
	11	3.436275	3.508794	3.449614	3.739653	4.079279	4.113926
	12	3.678104	3.763142	3.733964	3.981841	4.435476	4.41532
	13	4.057726	3.967748	4.057006	4.296496	4.76375	4.797256
	14	4.343076	4.336928	4.298309	4.61588	5.153102	5.051832
	15	4.604455	4.515618	4.623206	5.007675	5.451142	5.458481

it is possible to choose a value j that balances the need for performance and security in terms of non-invertibility.

V. SYSTEM SECURITY

A. ONE TIME USER KEY STRENGTH

The security of the OTUK mechanism depends on the security of the hash algorithm used and the size of the salts. A fundamental condition for the operation is to change the salts for each encryption, otherwise, the user's key could be exposed. Thus, the security of such a mechanism depends substantially on the possibility of having a collision between any two user keys, which produce, with a certain combination of salts, the same OTUK. The choice of the parameter of the number of salts has an impact on security: as the number of salting and encryption processes increases, the possibility of collisions occurring increases. However, this consideration should be deferred to the security of the hashing algorithm. In theory, where there should be a repetition of some salts for the same user, it is better for security that only the first salts used are repeated, because the more salts are repeated, the more it is (theoretically) possible to reverse the encryption or generate a collision on OTUKs. This consideration is, however, overzealous, because if it were possible to reverse even a single hash in a situation like this, in which the keys are completely unrelated to any semantic interpretation, the hashing algorithm used would be considered unsafe. If only the last salts were to be repeated, so the first salts are different, thus would not be a problem because the process of generating One Time User Keys would apply those salts on completely different values. Whether any OTUK can be traced back to the user key depends only on the non-invertibility of the chosen hash function. Dictionary or rainbow table attacks aimed at inverting the hash have extremely limited effectiveness, since none of the One Time User Keys comes from values with semantic meaning, but only from random bit sequences.
What is expressed depends only on the cryptographic hash function chosen, so it is independent of the proposed model.

B. STRENGTH OF MULTIPLE SECRET SHARING

In the proposed method, the same secret (k_*) is shared in different ways with each participant. Set $K^\oplus$ is public and known to everyone:

$$K^\oplus = \{k_1^\oplus, k_2^\oplus, ..., k_n^\oplus\} \tag{30}$$

$$k_1^\oplus = k_1 \oplus k_* \tag{31}$$

$$k_2^\oplus = k_2 \oplus k_* \tag{32}$$

$$... \tag{33}$$

$$k_n^\oplus = k_n \oplus k_* \tag{34}$$

It is necessary to understand what information about k_* can be exposed by this procedure.
Obtaining k_* from $K^\oplus$ is not possible. The only way to obtain the value of k_*, starting with $K^\oplus$, is to have an expression in the form:

$$k_* = \underbrace{x \oplus k_*}_{\in K^\oplus} \oplus x \tag{35}$$

It is obvious that $x \in K^+$, so the only way to derive k_* is to have one of the valid keys.
Combining the elements of $K^\oplus$ does not produce information about k_*. Taking any subset $T \subseteq K^\oplus$ ($|T| > 1$), performing the XOR operation among the elements of T, viz.:

$$T = \{k_a^\oplus, k_b^\oplus, ..., k_z^\oplus\} \subseteq K^\oplus \tag{36}$$

$$t^\oplus = k_a^\oplus \oplus k_b^\oplus \oplus ... \oplus k_z^\oplus \tag{37}$$

can lead to only two different situations:

1) $\underline{\text{If } |T| \text{ is even:}}$ $t^\oplus = k_a \oplus k_b \oplus ... \oplus k_z$, which does not contain information about k_*;
2) $\underline{\text{If } |T| \text{ is odd:}}$ $t^\oplus = k_* \oplus k_a \oplus k_b \oplus ... \oplus k_z$, which leads to a more complex case than the one currently being solved.

It is still possible to combine situation (1) and situation (2), i.e., to simultaneously find values $t^\oplus$ and $t^\oplus \oplus k_*$:

$$t^\oplus \oplus (t^\oplus \oplus k_*) = k_* \tag{38}$$

However, this is not possible since for both to be computable it must be simultaneously true that $|T|$ is even and that $|T|$ is odd.
The most information that can be obtained is by taking two by two elements of $K^\oplus$ and performing the XOR between them. The result show which bits each OTUK key has in common. As demonstrated in the next section, this information is not useful. If the keys had a semantic value (e.g. they were images), this would be a problem since the XOR between them would allow us to display the two overlapping semantics. However, in the proposed system, these are keys produced by hashing and used only once, so the problem does not arise.

C. BRUTEFORCE ATTACK

It could be assumed that the presence of multiple valid keys in this system increases the possibility of a bruteforce attack, but this is demonstrably not the case. This demonstration certainly applies to what is expressed in III-C, and, since the case in III-D is traced back to the case in III-C, it applies also for the III-D extension.
To perform a bruteforce attack, it is necessary to find a key that has already been hashed, as hashing operations would only lead to a slowdown of the attack.
Having $K^+ = \{k_1, k_2, ..., k_n\}$, $K^\oplus = \{k_1^\oplus, k_2^\oplus, ..., k_n^\oplus\}$ and k_*, a bruteforce attack means one of the following:

0. Attempting all possible keys directly on the decryption algorithm $\mathfrak{C}$ until k_* is found;
1. Attempting all possible keys, performing XOR with $k_1^\oplus$ and checking if the result is k_* by decryption $\mathfrak{C}$;
2. Attempting all possible keys, performing XOR with $k_2^\oplus$ and checking if the result is k_* by decryption $\mathfrak{C}$;

...

$n.$ Attempting all possible keys, performing XOR with $k_n^{\oplus}$ and checking if the result is k_* by decryption $\mathfrak{C}$.

Problems (0.), (1.), (2.), ..., $(n.)$ are equivalent to each other. As explained in V-B, only key similarity information can be obtained. This information does not allow us to improve the bruteforce attack: each of the problems (0.), (1.), (2.), ..., $(n.)$ can be traced back to one of these problems, hereafter referred to as $(p.)$, of our choice. This is done by choosing as a reference the key $k_p^{\oplus}$ and rewriting the other keys according to it.

Below evidence of this is provided through a practical example:

<u>Secret Information</u> - not known

$$k_* = 11110110$$

$$\begin{aligned} K^+ &= \{k_1, k_2, k_3, k_4\} \\ k_1 &= 10001010 \\ k_2 &= 01100111 \\ k_3 &= 10101011 \\ k_4 &= 00011010 \end{aligned}$$

$$\begin{aligned} k_1^{\oplus} &= k_1 \oplus k_* = 10001010 \oplus 11110110 = 01111101 \\ k_2^{\oplus} &= k_2 \oplus k_* = 01100111 \oplus 11110110 = 10010001 \\ k_3^{\oplus} &= k_3 \oplus k_* = 10101011 \oplus 11110110 = 01011101 \\ k_4^{\oplus} &= k_4 \oplus k_* = 00011010 \oplus 11110110 = 11101101 \end{aligned}$$

<u>Public Information</u> - usable

$$\begin{aligned} K^{\oplus} &= \{k_1^{\oplus}, k_2^{\oplus}, k_3^{\oplus}, k_4^{\oplus}\} \\ k_1^{\oplus} &= 01111101 \\ k_2^{\oplus} &= 10010001 \\ k_3^{\oplus} &= 01011101 \\ k_4^{\oplus} &= 11101101 \end{aligned}$$

In the example, assign $p = 3$, i.e., it is required to trace problems (1.), (2.) and (4.) back to (3.).

Let key k_p (in this example key k_3) be defined as a sequence of bits b:

$$k_p = k_3 = [b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8]$$

Now check, bit by bit, taking the bits of $k_p^{\oplus}$ as a reference, when the bits of the other keys $k_i^{\oplus}$ are the same (not underlined) and when they are different (underlined):

$$\begin{aligned} k_1^{\oplus} &= 01\underline{1}11101 \\ k_2^{\oplus} &= \underline{10}01\underline{00}01 \\ k_3^{\oplus} &= 01011101 \\ k_4^{\oplus} &= \underline{1}1\underline{10}1101 \end{aligned}$$

Consider with notation $k_i[j]$ the b_j relative to key k_i.

Now the keys k_1, k_2, k_4 can be rewritten as a function of k_3:

$$\forall k_i, i \neq p \quad \forall j \quad k_i'[j] = \begin{cases} b_j, & \text{if } b_j = k_i^{\oplus}[j] \\ \bar{b}_j, & \text{if } b_j \neq k_i^{\oplus}[j] \end{cases}$$

The keys are then rewritten as:

$$\begin{aligned} k_1 &= [b_1|b_2|\bar{b}_3|b_4|b_5|b_6|b_7|b_8] \\ k_2 &= [\bar{b}_1|\bar{b}_2|b_3|b_4|\bar{b}_5|\bar{b}_6|b_7|b_8] \\ k_3 &= [b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8] \\ k_4 &= [\bar{b}_1|b_2|\bar{b}_3|\bar{b}_4|b_5|b_6|b_7|b_8] \end{aligned}$$

This does not add any kind of information about the key k_*, since

$$\forall j \quad b_j = k_p[j] \oplus k_*[j]$$

which depends only on the keys k_p and k_*, that are unknown to us. Thus, it is possible to know the value of b_j as a function of other values, but never the actual boolean value, providing no advantage whatsoever in a bruteforce attack, since it is impossible to determine the value of any bit a priori. There is also no correlation between any two b_x and b_y, since the operation XOR is bitwise.

Therefore, it is possible, by employing b_j, to trace the problem $(i.)$ of finding any valid k_i back to the problem $(p.)$ of finding k_p, since there is a biunivocal correspondence between each key searched in the domain of the problem $(i.)$ with a key searched in the domain of the problem $(p.)$. Similarly, by transitivity, there is an obvious correspondence between all problems in this class. This correspondence occurs by placing each b_j equivalent to the bit of the attempted key in the domain $(p.)$. In the example above:

Trying the key 01010101 in the problem domain $(p.)$/(3.)

$$\begin{aligned} k_p = k_3' &= [b_1|b_2|b_3|b_4|b_5|b_6|b_7|b_8] \\ &= [\,0\,|\,1\,|\,0\,|\,1\,|\,0\,|\,1\,|\,0\,|\,1\,] \end{aligned}$$

is equivalent in domain (1.) to trying the key

$$\begin{aligned} k_1' &= [b_1|b_2|\bar{b}_3|b_4|b_5|b_6|b_7|b_8] \\ &= [\,0\,|\,1\,|\,1\,|\,1\,|\,0\,|\,1\,|\,0\,|\,1\,] \end{aligned}$$

and is equivalent in domain (2.) to trying the key

$$\begin{aligned} k_2' &= [\bar{b}_1|\bar{b}_2|b_3|b_4|\bar{b}_5|\bar{b}_6|b_7|b_8] \\ &= [\,1\,|\,0\,|\,0\,|\,1\,|\,1\,|\,0\,|\,0\,|\,1\,] \end{aligned}$$

and is equivalent in domain (4.) to trying the key

$$\begin{aligned} k_4' &= [\bar{b}_1|b_2|\bar{b}_3|\bar{b}_4|b_5|b_6|b_7|b_8] \\ &= [\,1\,|\,1\,|\,1\,|\,0\,|\,0\,|\,1\,|\,0\,|\,1\,] \end{aligned}$$

and they are all equivalent to trying, in domain (0.), the key 00001000 directly

$$\begin{aligned} &(1.) \quad k_1' \oplus k_1^{\oplus} = 01110101 \oplus 01111101 = 00001000 \\ &(2.) \quad k_2' \oplus k_2^{\oplus} = 10011001 \oplus 10010001 = 00001000 \\ &(3.) \quad k_3' \oplus k_3^{\oplus} = 01010101 \oplus 01011101 = 00001000 \\ &(4.) \quad k_4' \oplus k_4^{\oplus} = 11100101 \oplus 11101101 = 00001000 \end{aligned}$$

Attempting a certain key also does not result in excluding $|K^+|$ keys from it, since the rejection of that key applies only to the attempt of the user of the domain in which the key is being tested. To exclude equivalent keys in other domains, it is still necessary to test them.

In conclusion, problems (1.), ... $(n.)$ are mutually equivalent and correspond to a key bruteforce attempt, with the addition of an XOR operation for each key attempted. Attempting this way is, because of the XOR operation, more onerous than directly attempting bruteforce on k_*, so problem (0.) is always preferable. Since direct bruteforce on the algorithm is the best attack attempt, the problem is deferred to the security of the chosen cryptographic algorithm, so the problem is independent of the use of the proposed system.

VI. CONCLUSIONS

In this paper, a new user-based cryptographic model for generating encryption among multiple users is proposed. The proposed model allows obtaining mutually equivalent keys. In the proposed implementation, these keys are associated with user keys through the One Time User Key mechanism, to participate in a certain encrypted communication. The transmission of keys between the parties is done and the choice of the real key is make through a synchronization protocol, for which a possible mode of operation is proposed. The mechanism proves to be a decorator concerning current symmetric encryption mechanisms and is similarly applicable in asymmetric encryption mechanisms.

It has been proved that the security of the model depends solely on the cryptographic functions chosen and the size of the keys. So the system, as it is proposed, proves to be a non-impactful addition to security, as it depends only on factors not directly related to the system itself.

The proposed mechanism can be applied in distributed user-based systems, where ad-hoc encryptions can be defined based on the users who will be able to access this information. However, a dealer is needed to manage the keys, and, in particular, to generate and choose key k_*, and consequently set $K^{\oplus}$. It is necessary to use a synchronization protocol between the parties to make the system work, and if the proposed example protocol is chosen, the dealer must be benevolent, otherwise the dealer could illegally decrypt the communications between users. The system also needs to store information in addition to the encrypted data, as this information is essential for decryption.

The model can be used by government agencies by providing new cryptographic algorithms that already contain such a mechanism. The model becomes even more effective if, of course, such a mechanism is implemented in a hardware manner, effectively forcing users to use it. In this specific case, an OTUK, derived from a user key corresponding to the user *Government*, must also be added for each cipher. The chips should therefore be supplied with a government OTUK generator, without ever exposing the user key, or with a number of these keys. In the latter case, it is best that each chip has different lists. It is also possible to avoid concentrating the decryption power in a single key by distributing it among different keys assigned to different agencies, as specified in the III-D section. Thus, it is necessary for government agencies to work together to enable the specific decryption.

REFERENCES

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[2] G. R. Blakley, "Safeguarding cryptographic keys," in Managing Requirements Knowledge, International Workshop on, 1979, pp. 313–313.
[3] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in 26th Annual Symposium on Foundations of Computer Science (sfcs 1985), 1985, pp. 383–395.
[4] D. R. Stinson, "An explication of secret sharing schemes," *Designs, Codes and Cryptography*, vol. 2, no. 4, pp. 357–390, 1992.
[5] E. Dawson and D. Donovan, "The breadth of shamir's secret-sharing scheme," *Computers & Security*, vol. 13, no. 1, pp. 69–78, 1994.
[6] Council of the European Union, *Council Resolution on Encryption Security through encryption and security despite encryption*, 2020.
[7] K. Meng, F. Miao, Y. Ning, W. Huang, Y. Xiong, and C.-C. Chang, "A proactive secret sharing scheme based on chinese remainder theorem," *Frontiers of Computer Science*, vol. 15, no. 2, pp. 1–10, 2021.
[8] Y. Ning, F. Miao, W. Huang, K. Meng, Y. Xiong, and X. Wang, "Constructing ideal secret sharing schemes based on chinese remainder theorem," in International Conference on the Theory and Application of Cryptology and Information Security, 2018, pp. 310–331.
[9] M. Deepika and A. Sreekumar, "Secret sharing scheme using gray code and xor operation," in 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), 2017, pp. 1–5.
[10] N. Singh, A. N. Tentu, A. Basit, and V. C. Venkaiah, "Sequential secret sharing scheme based on chinese remainder theorem," in 2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2016, pp. 1–6.
[11] K. K. Phiri and H. Kim, "Linear secret sharing scheme with reduced number of polynomials," *Security and Communication Networks*, vol. 2019, 2019.
[12] P. D'Arco, R. De Prisco, A. De Santis, A. Pérez del Pozo, and U. Vaccaro, "Probabilistic secret sharing," in 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018), 2018.
[13] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in Annual international conference on the theory and applications of cryptographic techniques, 2015, pp. 337–367.
[14] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2014, pp. 640–658.
[15] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1292–1303.
[16] K. Ding and C. Ding, "A class of two-weight and three-weight codes and their applications in secret sharing,"

IEEE Transactions on Information Theory, vol. 61, no. 11, pp. 5835–5842, 2015.
[17] C.-F. Hsu, L. Harn, and B. Zeng, “Umkess: user-oriented multi-group key establishments using secret sharing,” *Wireless Networks*, vol. 26, no. 1, pp. 421–430, 2020.
[18] C. J. Mitchell, “Yet another insecure group key distribution scheme using secret sharing,” *Journal of Information Security and Applications*, vol. 57, p. 102713, 2021.
[19] S. Boonkrong and C. Somboonpattanakit, “Dynamic salt generation and placement for secure password storing,” *IAENG International Journal of Computer Science*, vol. 43, no. 1, pp. 27–36, 2016.
[20] F. L. Bauer, “Vernam cipher,” in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg, Ed. Springer, 2005, pp. 1359–1360.
[21] A. L. Hananto, A. Solehudin, A. S. Y. Irawan, and B. Priyatna, “Analyzing the kasiski method against vigenere cipher,” *arXiv preprint arXiv:1912.04519*, 2019.

STEFANO GALANTUCCI is a PhD student in Computer Science at the University of Bari. He holds a MSc degree in Cybersecurity with full marks and honors, defending a thesis on the generation of multiple cryptographic keys equivalent to each other. He is involved in research in the areas of cybersecurity, cryptography and biometrics. He is a reviewer for IEEE Access.

DONATO IMPEDOVO (M’08-SM’17) received the MEng degree cum laude in computer engineering and the PhD degree in computer engineering. He is associate professor with the Department of Computer Science of the University of Bari (IT). His research interests include field of signal processing, pattern recognition, machine learning and biometrics. He is co-author of more than 80 articles on these fields in both international journals and conference proceedings. He received the “distinction” award in May 2009 at the International Conference on Computer Recognition Systems (CORES – endorsed by IAPR), and the first prize of the first Nereus-Euroavia Academic competition on GMES in October 2012. He is also very involved in research transfer activities as well as in industrial research, he has managed more than 25 projects funded by public institutions as well as by private SMEs. He is IEEE Access associate editor and he serves as reviewer for many international journals including IEEE Access, IEEE Transactions on Systems, Man, and Cybernetics: Systems, Pattern Recognition and many others. He was the general cochair of the International Workshop On Artificial Intelligence With Application In Health (WAIAH2017), of the International Workshop on Emergent Aspects in Handwritten Signature Processing (EAHSP 2013) and of the International Workshop on Image-Based Smart City Application (ISCA 2015). He was a reviewer in the scientific committee and program committee of many international conferences in the field of computer science, pattern recognition and signal processing, such as the ICPR and ICASSP. He is IAPR and IEEE senior member.

GIUSEPPE PIRLO (M’92–SM’13) received the degree in computer science (cum laude) from the Department of Computer Science, University of Bari, Italy, in 1986. Since 1986, he has been carrying out research in the field of computer science and neuroscience, signal processing, handwriting processing, automatic signature verification, biometrics, pattern recognition and statistical data processing. Since 1991, he has been an assistant professor with the Department of Computer Science, University of Bari, where he is currently a full professor. He developed several scientific projects and authored more than 250 papers on international journals, scientific books and proceedings. He is currently an associate editor of the IEEE Transactions on Human–Machine Systems. He also serves as a Reviewer for many international journals including the IEEE Transactions on Pattern Analysis and Machine Intelligence, the IEEE Transactions on Fuzzy Systems, IEEE transactions on Systems, Man, and Cybernetics: Systems, the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Image Processing, the IEEE Transactions on Information Forensics and Security, the Pattern Recognition, the International Journal on Document Analysis and Recognition, and the Information Processing Letters. He was the general chair of the International Workshop on Emerging Aspects in Handwriting Signature Processing, Naples, in 2013, the International Workshop on Image-based Smart City Applications, Genoa, in 2015, and the general co-chair of the International Conference on Frontiers in Handwriting Recognition, Bari, in 2012. He was a reviewer in the scientific committee and program committee of many international conferences in the field of computer science, pattern recognition and signal processing, such as the ICPR, ICDAR, ICFHR, IWFHR, ICIAP, VECIMS, and CISMA. He is also the editor of several books. He was an editor of the Special Issue Handwriting Recognition and Other PR Applications of the Pattern Recognition Journal in 2014 and the Special Issue Handwriting Biometrics of the IET Biometrics Journal in 2014. He was the guest editor of the Special Issue of the Je-LKS Journal of e-Learning and Knowledge Society Steps toward the Digital Agenda: Open Data to Open Knowledge in 2014. He is currently the guest co-Editor of the Special Issue of the IEEE Transactions on Human–Machine Systems on Drawing and Handwriting Processing for User-Centered Systems. He is a member of the Governing Board of Consorzio Interuniversitario Nazionale per l’Informatica (CINI), a member of the Governing Board of the Societa Italiana di e-Learning and the e-learning Committee of the University of Bari. He is currently the deputy representative of the University of Bari in the Governing Board of CINI. He is also the managing advisor of the University of Bari for the Digital Agenda and Smart Cities. He is the chair of the Associazione Italiana Calcolo AutomaticoPuglia. He is also a member of the Gruppo Italiano Ricercatori Pattern Recognition, the International Association Pattern Recognition, the Stati Generali dell’Innovazione, and the Gruppo Ingegneria Informatica. He is a senior member of the IEEE.