

RESTORATION OF A DISTRIBUTED KEY FROM A BACKUP STORAGE

FIELD OF THE INVENTION

The present invention relates to a method for restoring a distributed secret key from a backup storage. The method according to the invention allows the secret key to be restored
5 in a secure manner, even if the distributed secret key has been refreshed several times since the backup was created.

BACKGROUND OF THE INVENTION

In threshold encryption schemes or threshold signature schemes, a secret key may be distributed among two or more servers in such a manner that each server holds at least one
10 share of the secret key, and none of the servers has access to all shares, and thereby to the entire secret key. When performing encryption, decryption or signatures, using the key, a subset of at least a predefined number of the servers needs to participate, in order for the transaction to be valid. An adversary would therefore need to compromise several of the
15 servers in order to gain access to a sufficient number of key shares to create a fraudulent signature, encryption or decryption which fulfils the threshold criteria.

Each of the servers will often want to create a backup of its share of the secret key at suitable time intervals. The backups are independent in the sense that the servers do not necessarily create backups at the same time. If one of the servers fails in a manner which requires that its content, including its share of the secret key, is restored from the backup,
20 this server will be restored, but the other servers would not be affected.

For long-term secret keys, for instance secret keys which are secret shared over an extensive time interval, an adversary may gradually gain access to a sufficient number of key shares to be able to use the secret key in a fraudulent manner. In order to avoid this, the secret key, and thereby the key shares, may be refreshed from time to time. Thereby an adversary will
25 not only need to gain access to a sufficient number of key shares. He would also need to gain access to the key shares within the same time window between two refreshes of the secret key. This is sometimes referred to as 'proactive secret sharing'.

However, in the case that the secret key has been refreshed one or more times from the point in time where a backup was created and until a point in time where it is necessary to
30 restore a key share from the backup, then the key share available from the backup differs from the currently applicable key share, and it is therefore no longer valid.

DESCRIPTION OF THE INVENTION

It is an object of embodiments of the invention to provide a method for restoring a distributed secret key from a backup storage, which allows a refreshed secret key to be restored without compromising the secrecy of the secret key.

- 5 It is a further object of embodiments of the invention to provide a method for restoring a distributed secret key from a backup storage, which allows a refreshed secret key to be restored, while maintaining proactive secrecy.

The invention provides a method for restoring a distributed secret key from a backup storage, the method comprising the steps of:

- 10 – generating an original secret key and distributing the original secret key among two or more servers, each server thereby holding a share of the original secret key and none of the servers holding all shares of the original secret key,
- a first server creating a backup, the backup containing at least the share of the original secret key which is held by the first server,
- 15 – the servers refreshing the original secret key at least once, where each refresh comprises the steps of:
- the servers generating a refreshed distributed secret key and a distributed difference between the previous secret key and the refreshed secret key, each server holding a share of the difference, and the difference constituting a
- 20 sharing of zero, and the servers discarding their shares of the previous secret key,
- each server generating a secret version of its share of the difference and sharing the secret version with at least some of the other servers, and
- each server generating an accumulated secret version of the shares of the difference received from the other servers based on the received secret
- 25 versions of the shares of the difference and previous accumulated secret versions,

- the first server restoring its share of the original secret key from the backup and requesting the accumulated secret version of its share of the difference from the other servers,
- 5 – at least some of the other servers providing the accumulated secret version of the first server's share of the difference to the first server, and
- the first server restoring its share of the latest refreshed secret key from the received accumulated secret version and the restored share of the original secret key.

Thus, the method according to the invention is a method for restoring a distributed secret key from a backup storage. The secret key could, e.g., be a secret encryption/decryption key
10 or a secret signature key, i.e. a secret key applied for generating digital signatures.

According to the method, an original secret key is generated and distributed among two or more servers, in such a manner that each server holds a share of the original secret key and none of the servers holds all shares of the secret key. Accordingly, none of the servers will be able to reconstruct the entire original secret key, and thereby at least a subset of a
15 predefined number of the servers need to participate in order to use the original secret key. The secret key may be generated first and then distributed among the servers, or the servers may cooperate in generating the secret key directly in a distributed form, e.g. using secret sharing.

The servers may be separate hardware servers which are physically separated from each
20 other. As an alternative, the servers may be in the form of virtual servers which may not necessarily be arranged on separate hardware. However, in any event, the servers are separated from each other in the sense that none of the servers can gain access to contents stored on any of the other servers.

Following the generation of the original secret key, a first server creates a backup. Since the
25 first server holds a first share of the original secret key, the created backup contains this first share of the original secret key. The backup of the first server is performed independently of corresponding backups of the other servers.

At a point in time after the backup was created, the servers refresh the original secret key at least once. Each refresh is performed in the manner described below.

30 Initially, the servers generate a refreshed distributed secret key and a distributed difference between the previous secret key and the refreshed secret key. The refreshed distributed

secret key and/or the distributed difference may be generated in a manner which is similar to the manner in which the original secret key was generated. This will be described in further detail below. In any event, each of the servers now holds a share of the refreshed secret key as well as a share of the distributed difference. Furthermore, none of the servers holds all shares of the refreshed secret key and none of the servers holds all shares of the distributed difference. Thus, none of the servers is able to reconstruct neither the entire refreshed secret key, nor the entire distributed difference on its own.

In the present context the term 'previous secret key' should be interpreted to mean the distributed secret key which is valid at the point in time where the refresh is performed. Thus, when the first refresh is performed, the previous secret key is the original secret. When the second refresh is performed, the previous secret key is the refreshed secret key which was generated during the first refresh of the secret key, and so forth.

The distributed difference constitutes a sharing of zero. Thereby the refreshed secret key is identical to the previous secret key, and thereby to the original secret key, in the sense that the difference between the distributed secret keys amounts to zero. However, the shares of the secret key which are held by the servers after the key refreshment are not identical to the shares of the secret key which are held by the servers prior to the key refreshment.

Once the refreshed secret key has been generated and distributed among the servers, the servers discard their respective shares of the previous secret key. Thereby the previous secret key is discarded, and the refreshed distributed secret key is the version which is valid until the next refresh is performed. Furthermore, it is no longer possible to gain access to any of the shares of the previous secret key, and accordingly, it is not possible to construct the entire previous secret key.

Furthermore, each server generates a secret version of its share of the difference and shares the secret version with at least some of the other servers. The secret version could, e.g., be an encrypted version of the share of the difference. Alternatively or additionally, the server may distribute shares of its share of the difference among the other servers, in which case the secret version may be in the form of a secret sharing of the share. In any event, the servers only reveal a secret version of their shares of the difference to the other servers, and only a given server is able to reconstruct its share of the difference from the secret version. Accordingly, none of the servers will be able to gain any knowledge of the actual shares of the difference which are held by the other servers, based on the received secret versions. Thereby, an adversary who has gained access to a previous share of the distributed secret key and a secret version of the share of the difference will not be able to reconstruct the corresponding share of the refreshed distributed key.

Finally, each server generates an accumulated secret version of the shares of the difference received from the other servers based on the received secret versions of the shares of the difference and previous accumulated secret versions. Thus, the accumulated secret versions of the shares of the difference represent the respective differences between the shares of the original distributed secret key and the currently valid shares of the refreshed secret key. The accumulated secret versions may be generated in various manners. This will be described in further detail below.

When the distributed secret key has been refreshed a number of times, in the manner described above, an incident occurs at the first server which requires that the first server, and thereby the share of the distributed secret key which is held by the first server, is restored. However, the share of the distributed key which is stored in the backup storage is a share of the original distributed secret key, and since the distributed secret key was refreshed at least once since the backup was created, the original secret key is no longer valid, and the share of the currently valid secret key, which is held by the first server, can not be immediately and directly restored from the backup.

Instead, the first server restores its share of the original secret key from the backup, i.e. the share which is actually available, and requests the accumulated secret version of its share of the difference from the other servers.

In response to this request, at least some of the other servers provide the accumulated secret version of the first server's share of the difference to the first server. In some embodiments, only one of the other servers may provide the accumulated secret version, in other embodiments two or more of the other servers may provide the accumulated secret version. Accordingly, the first server is now in the possession of its share of the original secret key and a secret representation of the difference between the first server's share of the original secret key and the first server's share of the currently valid secret key. However, since only the accumulated secret version was provided to the first server, none of the servers possess any knowledge regarding the first server's share of any intermediate versions of the refreshed secret key.

Finally, the first server restores its share of the latest refreshed secret key, i.e. the currently valid secret key, from the received accumulated secret version and the restored share of the original secret key. As described above, the accumulated secret version represents the difference between the first server's share of the original secret key and the first server's share of the currently valid secret key, or the latest refreshed secret key. Accordingly, the first server is able to derive this difference from the received accumulated secret version, and

thereby the first server can easily reconstruct its share of the latest refreshed secret key from the original share and the derived difference.

Thus, the method according to the invention allows independent backup of secret key shares of a threshold key system, and allows frequent refreshing of the distributed secret key, while
5 maintaining a high security in the sense of a low risk of leaking of the secret key, and allowing the key shares to be reliably recovered from the backup.

In particular, since the restoring of the key share only restores the latest refreshed version of the key share, and not any of the potential intermediate refreshed versions, it is not possible for an adversary to gain access to such intermediate refreshed versions. Thereby an
10 adversary is prevented from collecting a sufficient number of key shares belonging to the same refresh, after the period of the refresh has expired.

The method according to the invention, thus, allows a server to restore its current secret key share from a backup that includes an outdated version of the server's key share and a secret accumulative difference that the server receives from one or more of the other servers. In
15 particular, the method allows recovery of the first server's share without granting the other servers the capability to restore the first server's secret key share.

The step of each server generating a secret version of its share of the difference may be performed by means of an additively homomorphic scheme, and the step of each server generating an accumulated secret version may comprise each server adding the received
20 secret versions of the shares of the difference to the respective previous accumulated secret versions, and discarding the previous accumulated secret versions.

An additive homomorphic scheme is a scheme in which addition can be performed in the encrypted domain, i.e. $E(d_1 + d_2) = E(d_1) + E(d_2)$, where E designates encryption. Thus, when an additive homomorphic scheme is applied for generating the secret versions of the shares of
25 the difference, the respective servers may encrypt their shares, d_i , and forward the encrypted version, $E(d_i)$, to the other servers. The other servers may then obtain an encrypted version of the sum of shares of differences, simply by adding the encrypted versions.

Thus, according to this embodiment, the method may be performed in the following manner. For the sake of simplicity, the method will be described from the point of view of the first
30 server, i.e. only shares which are held by the first server are described. It should, however, be noted that similar steps may also be performed by the other servers.

When the first refresh is performed, the first server generates its share, d_1 , of the difference between the original secret key and the first refreshed secret key. This share, d_1 , of the difference corresponds to the difference between the first server's share of the original secret key and the first server's share of the first refreshed secret key. The first server then
 5 encrypts its share, d_1 , using the additive homomorphic scheme, thereby obtaining a secret version, $E(d_1)$, of the share, d_1 . The secret version, $E(d_1)$, is then forwarded to at least one of the other servers. This secret version of the share, d_1 , constitutes a first accumulated secret version, $A_1=E(d_1)$.

When the second refresh is performed, the first server similarly generates its share, d_2 , of the
 10 difference between the first refreshed secret key and the second refreshed secret key. Thereby the difference between the first server's share of the original secret key and the first server's share of the second refreshed secret key is d_1+d_2 .

The first server then decrypts its share, d_2 , using the additive homomorphic scheme, thereby obtaining a secret version, $E(d_2)$, of the share, d_2 . The secret version, $E(d_2)$, is then
 15 forwarded to at least one of the other servers. In response to the receipt of the secret version, $E(d_2)$, of the second share, the other servers generate a second accumulated secret version, A_2 , of the first server's share, d_1+d_2 , of the difference between the original secret key and the second refreshed secret key, by adding the newly received secret version, $E(d_2)$, to the previously received secret version, $E(d_1)$, thereby obtaining $A_2=E(d_1)+E(d_2)=A_1+E(d_2)$.
 20 Since E represents an additive homomorphic encryption scheme, the accumulated secret version is in fact a secret version of the accumulated difference, i.e.
 $A_2=E(d_1)+E(d_2)=E(d_1+d_2)$. Once the second accumulated secret version, A_2 , has been generated, the servers discard the first accumulated secret version, A_1 .

When the third refresh is performed, the steps described above are repeated, and a third
 25 accumulated secret version, A_3 , is generated, where $A_3=A_2+E(d_3)=E(d_1+d_2+d_3)$, and the previous accumulated secret version, A_2 , is discarded.

For each refresh, this is repeated, and thereby the servers, at all times, hold the latest version of the accumulated secret version, $A_n=E(d_1+d_2+\dots+d_n)$, but no previous versions of the accumulated secret version.

30 When a backup of the first server is required, the first server retrieves its share of the original secret key from the backup storage, and requests the accumulated secret version of the difference from the other servers. At least one of the other servers then provides the currently valid version of the accumulated secret version, A_n , to the first server. The first server is then able to decrypt the received accumulated secret version, A_n , thereby obtaining

the actual difference, $d_1 + d_2 + \dots + d_n$, between the first server's share of the original secret key and the first server's share of the currently valid refreshed secret key. Accordingly, the first server can restore its share of the currently valid refreshed secret key by adding the accumulated difference to the restored share of the original secret key. Accordingly, the

5 share of the currently valid refreshed key is restored without revealing any of the intermediate shares.

One example of an additive homomorphic scheme is Paillier's public key encryption scheme.

The additive homomorphic scheme may be a public-key encryption scheme, and the step of the first server creating a backup may comprise storing a private decryption key as part of

10 the backup. According to this embodiment, the private decryption key is required when the first server needs to decrypt the accumulated secret version received from the other server(s), but not necessarily for anything else. By storing the private decryption key as part of the backup, it is ensured that it is difficult for an adversary to gain access to the private encryption key, since the backup storage is only accessed when it is necessary to restore the

15 first server from the backup. On the other hand, when this situation occurs, the first server retrieves the contents of the backup storage, including the private decryption key, and thereby the private decryption key is available to the first server for decrypting the received accumulated secret version as part of restoring the share of the currently valid secret key. When creating a backup, the first server may further delete the private decryption key from

20 the first server once it has been stored in the backup storage. Similarly, when restoring a backup the first server may delete the private decryption key from the first server once it has used it to decrypt the received secret accumulative versions of the difference. This will increase security since it will limit the time frame in which an adversary who compromises the first server will be able to gain access to the private decryption key.

25 So far described the invention has been described in terms of additive sharing, where the shares of the secret key s satisfy $s = s_1 + s_2 + \dots + s_n$, and where the difference generated in the refresh process is in the form of an additive sharing of zero which is added to the sharing of the secret, by each server locally adding its share of the secret with its sharing of zero. In an alternative embodiment, the sharing of the secret may be multiplicative, i.e., the shares of

30 the secret s satisfy $s = s_1 * s_2 * \dots * s_n$, and the step of refreshing the sharing of the secret consists of generating a multiplicative random sharing of one and multiplying this to the existing sharing of the secret to produce the refreshed sharing, by each server locally multiplying its share of one with its share of the secret. In this case, generating a secret version of the share of the difference may be performed by means of a multiplicatively

35 homomorphic scheme. This could be a multiplicatively homomorphic public key encryption scheme such as the ElGamal encryption scheme.

In the following we continue to use additive notation, i.e., we will say that the sharing of the secret is refreshed by adding a sharing of zero to produce the sum $d_1+d_2+\dots+d_n$. However, the method works similarly in the case of a multiplicative sharing where the sharing of the secret is refreshed by multiplying a sharing of one to produce a secret version of the accumulated $d_1*d_2*\dots*d_n$.

As an alternative, the private decryption key may be stored separately, e.g. in an external storage, such as a cold storage, a hardware secure model (HSM), and/or it may be decrypted by mean of an encryption key or protected by a password, which is not readily available at the first server. For instance, the encryption key or the password may be managed by an administrator. In this case, the private encryption key must be obtained separately during restoration from the backup, e.g. by requesting the private decryption key from the cold storage or requesting the administrator to decrypt the private decryption key or to perform the decryption of the accumulated secret version.

The step of each server generating a secret version of its share of the difference may comprise each server creating a sharing of its share of the difference and distributing the shares among at least some of the other servers. The sharing may, e.g., be in the form of a secret sharing. According to this embodiment, the secret version of the share of the difference is in the form of the created sharing. Thereby none of the other servers will be in the possession of the entire share of the difference. According to this embodiment, when the servers receive a share of the first server's share of the next distributed difference, it adds this to its previous share of the first server's share of the distributed difference to obtain the new accumulated share of the first server's share of the distributed difference, and it discards its old share. Creating a sharing in this manner may be applied as an alternative to the public-key encryption scheme described above. As an alternative, these two approaches may be combined in the sense that the shares being distributed to the other servers may additionally be encrypted.

Each refresh may comprise the steps of:

- the servers creating a sharing of zero, and each server adding its share of the sharing of zero to its share of previous secret key, thereby creating the refreshed secret key being distributed among the servers, and discarding their shares of the previous secret key, the sharing of zero constituting the distributed difference, and
- each server generating a secret version of its share of the sharing of zero and sharing the secret version with at least some of the other servers.

According to this embodiment, the refreshed secret key is generated by the servers generating, amongst them, a sharing of zero, such as a secret sharing of zero, and subsequently adding this sharing of zero to the previous secret key. This is done by each server adding its share of the sharing of zero to its share of the previous secret key.

- 5 Furthermore, according to this embodiment, the shares of the distributed difference held by the servers are the shares of the sharing of zero which are held by the respective servers.

As an alternative, the servers may generate the refreshed secret key by creating a new sharing of the previous secret key, and derive the distributed difference from their respective shares of the new refreshed secret key and the previous refreshed secret key.

- 10 The method may further comprise the step of storing the backup in an offline storage. According to this embodiment, the backup is not accessible online, and an adversary would therefore need to physically break into the offline storage in order to gain access to the backup. The offline storage could, e.g., be in the form of a portable storage medium, such as a USB storage or a disk, which may be locked into a safe or another restricted area. The
- 15 location of the offline storage may differ from the locations of the servers, in which case it is necessary to physically transfer the backup between the locations when the backup is created, as well as when restoration of the backup is required. This makes it cumbersome to create backups, and therefore backups may not be created very frequently. This makes the method of the invention very relevant, because it can therefore be expected that a high
- 20 number of refreshes of the secret key will be performed between two backups.

Each step of refreshing the secret key may further comprise the step of each server verifying correctness of the received secret versions of the shares of the difference by verifying a zero-knowledge proof.

- 25 Apart from attempting to gain unauthorised access to the secret key, an adversary may also seek to actively corrupt the system by providing false information to honest parties, i.e. honest servers. For instance, an adversary may corrupt one of the servers, e.g. the first server. In this case the shares of differences received by the other servers and originating from the first server may be corrupted. Accordingly, there may be a need to be able to detect if one of the parties deviates from the prescribed protocol. According to this embodiment,
- 30 this is done by verifying a zero-knowledge proof.

In the present context the term 'zero-knowledge proof' should be interpreted to mean a method in which one party proves to another party that he knows a secret without revealing any details about the secret. Thus, according to this embodiment, the first server proves that the secret version of the share of the difference forwarded to the other servers is indeed a

secret version of the actual share. The first server does this in zero knowledge, i.e. without revealing any details about the actual share. The other servers then verify the proof, and if it turns out that the proof is invalid, the other servers will know that the first server is corrupted, and they may cause the refresh protocol to abort.

- 5 Alternatively or additionally, the method may further comprise the step of the first server verifying correctness of the received accumulated secret version by verifying a zero-knowledge proof.

- 10 This is similar to the embodiment described above. However, in this case it is the first server which ensures that the accumulated secret version, which it receives during restoration from the backup, is in fact correct. Thereby it is possible for the first server to detect if one or more of the other servers is corrupt and/or deviates from the prescribed protocol. If this is the case, the first server may cause the restoring process to abort.

- 15 Alternatively or additionally, each step of refreshing the secret may further comprise the step of each server verifying that it holds the same secret difference as the other servers, e.g., by broadcasting and comparing the difference, or a hash digest of the difference. In the case that the servers do not agree that they hold the same secret difference, the refreshing process may be aborted.

The distributed secret key may be a secret signature key. According to this embodiment, the system is applied for providing digital signatures, using the distributed secret key.

- 20 Furthermore, according to this embodiment, correctness of the secret shares of the differences and/or of the accumulated secret version may be ensured by the servers generating a test signature, e.g. a threshold signature, and the parties may verify that the test signature is valid and correct.

- 25 As an alternative, the distributed secret key may be a secret encryption key. According to this embodiment, the system is applied for providing encryption and decryption of data, using the distributed secret key.

- 30 In the special case where there is only two servers, the second server already knows the value of the first server's share of zero, because it knows the value of its own share and it knows that the sum of the two shares is zero. In this case, and if the secret version of the secret and the zero shares are generated using encryption, using a zero-knowledge proof is not necessary. The second server may instead verify correctness of the received secret share of zero by performing itself the encryption of the first server's share of zero and comparing

this to the secret version received from the first server. This can be done by using a deterministic encryption scheme, or alternatively, by letting the first server reveal to the second server the randomness that it used to encrypt its share of zero.

5 The method may further comprise the step of the first server creating a new backup, and repeating the steps of the servers refreshing the original secret key at least once, and the step of the first server restoring its share of the original secret key may comprise restoring a share of the secret key stored with the new backup.

10 According to this embodiment, the process described above is repeated each time a new backup is created, and when restoration from a backup is required, the latest available backup is used. Thus, the distributed secret key which was valid at the time where the new backup was created, may, e.g., be regarded as the original secret key in the sense of the claimed method.

15 Alternatively or additionally, each refresh may further comprise the step of the first server generating an accumulated secret version of its share of the difference and storing the accumulated secret version at the first server, wherein the secret version of the share of the difference forms part of the new backup, and the step of the first server restoring its share of the latest refreshed secret key may be performed based on the restored share of the secret key forming part of the new backup, the secret version of the accumulated difference forming part of the new backup, and the accumulated secret version received from at least some of
20 the other servers.

In some cases, server backup may be performed as an independent and autonomous process, and with no correlation to generation of distributed secret keys. Therefore, the share of a secret key, which is contained in a given backup, may not necessarily be consistent with the secret versions of the share of the accumulated difference held by the other servers in
25 the sense that the first server's share of the latest refreshed key cannot readily be restored based on the share which is contained in the backup and the accumulated secret version received from the other servers. According to this embodiment, this may be handled in the following manner.

30 During each refresh, the first server will not only generate accumulated secret versions of its share of the difference and send these to the other servers, it will also itself keep an accumulated secret version of its own share of the accumulated difference, storing this along with its share of the refreshed secret key which is valid at the time, and deleting previous versions. The accumulated secret version of the difference should preferably be of a kind which does not readily allow the individual differences of the accumulated difference to be

derived. For instance, this may require that the accumulated secret version of the difference is based on homomorphic public-key encryption for which decryption is not accessible from the first server, for instance the decryption key may be stored in cold storage or in a hardware security module (HSM).

- 5 When the new backup is created, the share of the refreshed secret key which is valid at the time, as well as the secret version of the accumulated difference which is valid at the time, form part of the new backup, i.e. they are stored as part of the new backup.

- At a later point in time, where it is necessary to restore the share of the secret key from the backup, the first server restores the no longer valid refreshed share and secret version of the
10 accumulated difference from the backup. The first server also receives the secret version of the accumulated difference from at least some of the other servers. Using the homomorphic property of the encryption, the first server then adds the secret version of the accumulated difference from the backup and the secret version of the accumulated difference received from at least some of the other servers. This produces a secret version of a combined
15 accumulated difference that is the accumulated difference between the no longer valid refreshed share from the backup and the latest refreshed share which the first server wants to restore.

- The first server then decrypts this combined secret accumulated difference, e.g. by requesting a private decryption key from a cold storage, by sending the combined secret
20 accumulated difference to a hardware security module (HSM), or the like. The first server then adds the decrypted accumulated difference to its no longer valid share from the backup in order to restore its latest refreshed version of its share (adding the accumulated difference from the backup brings the share back to the time of its creation, and adding the accumulated difference from the other servers brings it from its time of creation and up to
25 the share of the current time).

- Thus, the step of restoring the share of the original secret key and the step of restoring the share of the latest refreshed secret key may be performed in a single step, as described above. Alternatively, they may be performed as separate steps, in which case the first server's share of the original secret key may be restored by means of the accumulated
30 difference forming part of the backup, and the share of the latest refreshed secret key may subsequently be restored based on the restored original share and the received accumulated secret version, in the manner described above.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will now be described in further detail with reference to the accompanying drawings in which

Fig. 1 is a diagram illustrating a prior art method, and

- 5 Figs. 2-8 are diagrams illustrating methods according to seven different embodiments of the invention.

DETAILED DESCRIPTION OF THE DRAWINGS

- Fig. 1 is a diagram illustrating a prior art method for restoring a distributed secret key from a backup storage. In the example illustrated in Fig. 1, an original secret key, s , is distributed
10 among three servers, i.e. server 1, server 2 and server 3. Accordingly, each server holds a share of the secret key, and none of the servers is in the possession of all shares, and thereby the entire key. Thus, server 1 holds share s_1^1 , server 2 holds share s_2^1 , and server 3 holds share s_3^1 . It should be noted that the secret key may, alternatively, be distributed among two servers or among four or more servers, and that the presence of three
15 servers in Fig. 1 is merely illustrative.

At time T^1 , which may be immediately after the original secret key was distributed among the servers, server 1 stores a backup in a cold store. The backup comprises server 1's share of the original secret key, s_1^1 .

- At time T^2 , the servers refresh the distributed secret key, and each server now holds a
20 share of the refreshed secret key. Accordingly, server 1 holds share s_1^2 , server 2 holds share s_2^2 , and server 3 holds share s_3^2 , and the previous shares are no longer valid, and have therefore been deleted.

- Another refresh of the distributed secret key is performed at time T^3 in a similar manner. Accordingly, server 1 now holds share s_1^3 , server 2 holds share s_2^3 , and server 3 holds
25 share s_3^3 . It should be noted that the refresh process may be repeated more times.

While the key shares s_1^3 , s_2^3 and s_3^3 are valid, server 1 experiences a data loss which requires a restoration from the backup. However, the key share which was stored with the backup is s_1^1 , which is no longer valid, and not the currently valid key share, s_1^3 . Server 1 is therefore not able to restore its share of the currently valid secret key from the backup.

Fig. 2 is a diagram illustrating a method according to a first embodiment of the invention. In Fig. 2 only two servers, server 1 and server 2, are illustrated for the purpose of simplicity. However, it should be noted that three or more servers could, alternatively, be applied.

Similarly to the prior art method described above with reference to Fig. 1, a secret key is distributed among the two servers, and server 1 stores a backup of its share, $s1^1$, in a cold store. However, server 1 also creates a key pair with a private key, sk , and a public key, pk , and the private key, sk , is stored in the cold store with the backup. The key pair constitutes an additive homomorphic encryption scheme.

Also similar to the prior art method described above with reference to Fig. 1, the distributed secret key is refreshed a number of times. In the method illustrated in Fig. 2, each refresh of the distributed secret key is performed in the following manner.

The distributed secret key is refreshed in a standard manner, thereby generating a refreshed distributed secret key and a difference between the refreshed secret key and the previous secret key, the difference constituting a sharing of zero among the servers. This may be done in at least two ways. The servers may generate the sharing of zero, and each server may add its share of the sharing of zero to its share of the previous secret key, and thereby obtain its share of the refreshed key. As an alternative, the servers may generate the refreshed key, e.g. by creating a new sharing of the previous key, and each server may derive its share of the sharing of zero as the difference between its share of the previous secret key and its share of the refreshed secret key.

In any event, after the first refresh of the secret key, at time T^2 , the first server holds share $s1^2$ of the refreshed secret key, and $s1^2 = s1^1 + d1^1$, where $d1^1$ is server 1's share of the sharing of zero.

Server 1 then encrypts its share of the sharing of zero, $d1^1$, using the public key of the previously generated key pair, thereby obtaining $D^1 = \text{Enc}_{pk}(d1^1)$, and provides D^1 to the other servers, including server 2. Server 1 further deletes $s1^1$, $d1^1$ and sk , sk thereby being stored in the backup storage only.

Upon receipt of D^1 , server 2 generates an accumulated version of the received encrypted shares of the sharing of zero. Since D^1 relates to the first refresh, it is also the first encrypted share which server 2 receives, and therefore the accumulated version is, in this case, $A^1 = D^1$.

During the next refresh, server 1 provides encrypted share, $D^2 = \text{Enc}_{pk}(d1^2)$ to server 2, and server 2 generates a new accumulated version, $A^2 = A^1 + D^2 = D^1 + D^2$, and deletes A^1 .

Furthermore, for each subsequent refresh, server 2 generates a new accumulated version, $A^n = A^{(n-1)} + D^n = D^1 + D^2 + \dots + D^{(n-1)} + D^n$, and deletes the previous accumulated version, $A^{(n-1)}$. Since the encryption applied to the shares of the sharing of zero is performed using an additive homomorphic encryption scheme, the sum of the encrypted versions is equal to the encrypted versions of the sum, i.e.

$\text{Enc}_{pk}(d1^1 + d1^2) = \text{Enc}_{pk}(d1^1) + \text{Enc}_{pk}(d1^2) = D^1 + D^2$. Accordingly, the accumulated version held by server 2 corresponds to an encrypted version of the sum of server 1's shares of sharings of zero associated with each refresh performed since the backup was created.

In the method illustrated in Fig. 2, server 1 experiences a data loss which requires restoration from the backup, while secret key share $s1^4$ is valid. Accordingly, server 1 restores its share of the original distributed secret key, i.e. $s1^1$, and the private key, sk , from the backup. Furthermore, server 1 requests the currently valid accumulated version, A^4 , from the other servers, and at least server 2 returns A^4 to server 1.

Server 1 then decrypts A^4 , using the restored private key, sk , and thereby obtains the sum of all the differences which has been added to server 1's share of the original distributed secret key. In other words:

$$\begin{aligned} \text{Dec}_{sk}(A^4) &= \text{Dec}_{sk}(D^1 + D^2 + D^3) \\ &= \text{Dec}_{sk}(D^1) + \text{Dec}_{sk}(D^2) + \text{Dec}_{sk}(D^3) \\ &= d1^1 + d1^2 + d1^3 \end{aligned}$$

Thus, server 1 can restore its share of the valid distributed key, $s1^4$, by adding the decrypted accumulated version, $\text{Dec}_{sk}(A^4)$, to the restored original share, $s1^1$, because $s1^4 = s1^1 + d1^1 + d1^2 + d1^3 = s1^1 + \text{Dec}_{sk}(A^4)$.

Furthermore, the valid share is restored without revealing anything about any of the intermediate refreshed key shares $s1^2$ and $s1^3$ to server 1.

Fig. 3 is a diagram illustrating a method according to a second embodiment of the invention. The method is very similar to the method illustrated in Fig. 2, and it will therefore not be described in detail here. However, in the method illustrated in Fig. 3, the refresh of the distributed secret key is performed in the following manner.

Server 1 obtains a share of the refreshed secret key and a share of the sharing of zero constituting the difference between the refreshed key and the previous key, in the manner described above with reference to Fig. 2. A secret sharing of server 1's share of the difference, $d1^1$ in the first refresh, is then created, and each of the other servers receives a share. For instance, server 2 receives share $a2^1$.

The other servers, including server 2, generate accumulated versions, $A2^n$, of the differences by, during each refresh, adding the newly received share to the previous accumulated version, $A2^{(n-1)}$, and subsequently deleting the previous accumulated version, $A2^{(n-1)}$.

- 10 While secret key share $s1^4$ is valid, server 1 experiences a data loss which requires restoration from the backup. Accordingly, server 1 restores its share of the original distributed secret key, $s1^1$, from the backup, and requests the accumulated versions from the other servers. The other servers provide their accumulated versions to server 1. For instance, server 2 provides $a2^4$, and server i provides ai^4 . Server 1 is then able to restore
- 15 the sum of the differences associated with each refresh by recombining the accumulated versions received from the other servers, and by adding the restored sum to the restored original share, $s1^1$, the valid share, $s1^4$, is obtained. Also in this embodiment, nothing is revealed about any of the intermediate refreshed key shares $s1^2$ and $s1^3$.

- Fig. 4 is a diagram illustrating a method according to a third embodiment of the invention.
- 20 The method illustrated in Fig. 4 is very similar to the methods illustrated in Figs. 2 and 3, and it will therefore not be described in detail here. However, in the embodiment of Fig. 4, server 1's share of the difference, $d1^1$, is secret shared, as described above with reference to Fig. 3, and the secret shares are encrypted using an additively homomorphic encryption scheme, as described above with reference to Fig. 2, before being provided to the respective other
- 25 servers. Accordingly, the embodiment of Fig. 4 may be regarded as a combination of the embodiment of Fig. 2 and the embodiment of Fig. 3. This adds additional security to the system.

- Fig. 5 is a diagram illustrating a method according to a fourth embodiment of the invention. The method of Fig. 5 is very similar to the method illustrated in Fig. 2, and it will therefore
- 30 not be described in detail here.

In the method illustrated in Fig. 5, server 1 generates a zero-knowledge proof, ZKP, during each refresh, and forwards this to the other servers along with the encrypted version of its share of the sharing of zero. The other servers then verify the zero-knowledge proof. If at

least one server finds that the zero-knowledge proof is not valid, then the refresh process is aborted.

Fig. 6 is a diagram illustrating a method according to a fifth embodiment of the invention. The method of Fig. 6 is very similar to the method illustrated in Fig. 2, and it will therefore not be described in detail here. However, in the method illustrated in Fig. 6, the servers compare their accumulated versions during each refresh by sending their version to all of the other servers, and each server comparing the received versions. In the case that at least one of the servers receives accumulated versions which are not identical, the refresh process is aborted.

- Fig. 7 is a diagram illustrating a method according to a sixth embodiment of the invention. The method of Fig. 7 is very similar to the method illustrated in Fig. 3, and it will therefore not be described in detail here. However, in the embodiment of Fig. 7, each refresh includes an additional verification step performed before the refreshed secret key and the sharing of zero are generated.
- The distributed secret key is, in this case, a signature key which is used for generating digital threshold signatures, and each server is in the possession of a public verification key, VK. In the additional verification step, the servers agree on a random message, M^1 , and they then sign the random message, using their respective shares of the distributed key, thereby generating a signature, SIG^1 . The generated signature is provided to each of the servers, and each server verifies the signature by means of the public verification key, VK. The refresh process is aborted if at least one server is unable to verify the signature.

It should be noted that, even though Figs. 4-7 only illustrate the refreshes of the distributed secret key, it should be understood that restoration from the backup can also be performed, essentially in the manner described above with reference to Fig. 2 or Fig. 3.

- Fig. 8 is a diagram illustrating a method according to a seventh embodiment of the invention. The method of Fig. 8 is very similar to the method illustrated in Fig. 2, and it will therefore not be described in detail here.

The method illustrated in Fig. 8 uses, similarly to the embodiment of Fig. 7, a threshold signature scheme, but in contrast to the embodiment of Fig. 7, in the method of Fig. 8 the distributed secret key does not have to be the same as the distributed key of the threshold signature scheme. In the embodiment of Fig. 8 the public verification key of the threshold signature scheme (VK) is stored in the backup

In the embodiment of Fig. 8, the servers sign the encrypted share of the difference, D^1 , using the distributed signature key, thereby generating a signature, SIG^1 . Each of the servers stores SIG^1 along with the accumulated version of the difference.

5 When restoration is required, and server 1 therefore requests the accumulated version from the other servers, the other servers also provide the currently valid signature, in the example of Fig. 8 SIG^2 . Server 1 then verifies the signature, SIG^2 , and if it is not able to do so, the restoration process is aborted.

Optionally, a time stamp may be included, in which case server 1 also verifies that T^2 in fact represents the time immediately before the data loss.