Password systems: design and implementation

Gianluca Dini, Lanfranco Lopriore
Dipartimento di Ingegneria dell'Informazione, Università di Pisa, via G. Caruso 16, 56126 Pisa, Italy
E-mail: {g.dini, l.lopriore}@iet.unipi.it

Abstract — Critical infrastructures require protection systems that are both flexible and efficient. Flexibility is essential to capture the multi-organizational and state-based nature of these systems, efficiency is necessary to cope with limitations of hardware resources. To meet these requirements, we consider a classical protection environment featuring subjects that attempt to access the protected objects. We approach the problem of specifying the access privileges held by each subject. Our protection model associates a password system with each object; the password system features a password for each access privilege defined for this object. A subject can access the object if it holds a key matching one of the passwords in the password system, and the access privilege corresponding to this password permits to accomplish the access. Password systems are implemented as hierarchical bidimensional one-way chains. Trade-offs are possible between the memory requirements for storage of a password system and the processing time necessary to validate a key.

Keywords: access right; distribution; key; password; revocation.

1. INTRODUCTION

Critical infrastructures are essentially physical processes controlled by networked computers. They are usually as vulnerable as any other interconnected computer system, but their failure may have a high socio-economic impact [1]; furthermore, they present distinguishing features that make their protection a problematic challenge [2]. In critical infrastructures, the use of wireless sensor and actuator networks (WSANs) is becoming pervasive, and consequently the network boundaries of the system become blurred. The integration of multi-hop WSANs with supervisory control and data acquisition (SCADA) systems to monitor critical infrastructures is considered a promising approach [3], which extends the extent of SCADA systems and represents a cost-effective solution to the problem of limited deployment flexibility.

However, the integration of WSAN with SCADA poses new cybersecurity challenges, e.g. an adversary equipped with a radio transceiver can access the wireless medium and attack sensors and actuators at little effort, to eavesdrop their state, alter their set-up, and issue fraudulent commands. Sensors and actuators are usually resource-scarce devices, and this precludes utilization of off-the-shelf protection solutions, e.g. digital signatures and trusted hardware. Furthermore, a protection system for critical infrastructures is required to comply with a multi-organizational nature and different operational states. In fact, subjects from distinct organizations are often involved in a single critical infrastructure. Each subject executes operations in

the infrastructure, and almost all operations are based on a state model. It follows that the actions a subject can undertake in a given state may be forbidden in a different state.

In the design and implementation of a protection system, these complex aspects of a critical infrastructure should be taken into careful account. In particular, the multi-organizational and state-based characteristics imply that the protection system should be able to grant and revoke access privileges with flexibility and efficiency. Fine-grained forms of protection should be supported at level of a single object (a data item, as well as a device). The requirements of the protection system in terms of computing power and storage overhead should be kept to a minimum to comply with the limited resources available in the physical objects. Access control has been listed among the technical and management activities aimed at limiting or containing cybersecurity events that are common across critical infrastructure sectors [4]. In particular, accesses to assets and associated facilities must be limited to authorized users, processes, or devices, and to authorized activities and transactions. A suitable management of access permissions is required to incorporate the principles of least privilege and separation of duties.

In the following, we shall refer to a classic protection model featuring active entities, the *subjects*, which perform access attempts to passive entities, called *objects* [5], [6]. Objects are typed; the type of a given object states the values that can be assumed by this object and the operations that can be applied to these values. The type also states the set of *access rights* and the associations between the operations and the access rights. In order to access an object *G* of a given type *T* to perform one of the operations of this type, a subject must possess the access rights permitting successful execution of this operation.

A basic problem in every protection model is to specify the access rights that each subject holds on the protected objects. In a classical approach, a set of *passwords* is associated with each object, and each password corresponds to an access privilege defined in terms of one or more access rights [7], [8], [9]. Subject *S* is entitled to access object *G* if it possesses a key *k* matching one of the passwords associated with *G*. The key certifies that the subject holds the access rights corresponding to the matching password, and allows the subject to accomplish the operations made possible by these access rights. Keys are protected from forging by the password length [10]. If passwords are large and chosen at random, the probability to guess a valid key is vanishingly low.

1.1. Key distribution and derivation

Subject *S* that possesses key *k* for object *G* can transfer the key to another subject *S'*. Consequently, *S'* acquires the access rights for *G* that are granted by *k*. A salient problem of key

distribution consists in allowing subject *S* to grant *S'* only a part of the access rights included in the original key *k*. *Key derivation* is the process of transforming key *k* into another key *k'* that grants a weaker privilege. This means that the new key *k'* matches a password corresponding to the weaker privilege.

In a centralized approach, key derivation can be obtained by associating a password manager with each object. The password manager receives a key corresponding to a given password and returns a key corresponding to a password with reduced access rights. This approach implies interaction between a subject and the password manager. In a different approach, the protection system includes a key derivation mechanism that allows subjects to transform keys into weaker keys autonomously.

1.2. Key revocation

Ease of access privilege distribution certainly was one of the main reasons for the introduction of password-based protection. A subject that receives a key matching a given password is free to propagate the key further to other subjects; a result of this type will be simply obtained by a key copy. It follows that keys tend to propagate throughout the system.

A related problem is that of *key revocation* [11], [12], [13], [14]. A key is revoked when it can no longer be used for successful object access. The protection system should support a key revocation mechanism, and a *revoke* access right, so that a subject that holds this access right for given keys can revoke these keys. Revocation should be selective, so that only a subset of the keys distributed for a given object are revoked. Revocation should transitively extend throughout the system to all the copies of the revoked keys, as well as to all the keys derived from the revoked keys, and their copies.

Of course, in a password-oriented system, a simple solution to the key revocation problem is password replacement. If we change the password for a given access privilege, all the keys matching this password are revoked. This solution does not meet the requirement to extend revocation automatically to all the keys derived from the key being revoked. It follows that a subject can circumvent revocation by taking advantage of the derived keys.

In this paper, we present a model of a protection system based on typed objects, passwords and keys. Our model was designed to meet the following requirements:

- A subject that holds a given key should be given the ability to derive new keys with reduced access privileges. The entire key derivation process should be local to the subject, with no need for intervention of a centralized password manager.
- Forms of selective key revocation should be supported. Revocation should be transitive

with respect to key derivation, so that revocation of a given key k implies revocation of all copies of k, as well as of all the keys derived from k, and their copies.

- At the implementation level, trade-offs should be possible between the memory space necessary for password storage and the processing time for key validation, so that if low memory cost is not a stringent requirement, the time necessary to validate a key can be kept to a minimum, for instance.

The rest of this paper is organized as follows. Section 2 introduces our protection model with special reference to access privileges and their division into privilege levels. The concept of access privilege distribution is introduced whereby a subject that holds an access privilege in a given privilege level can grant another subject this privilege as well as any weaker privilege in the same level. Our protection model associates a password system with each given object; the password system features a password for each access privilege defined for this object. Possession of an access privilege is certified by possession of a key matching one of the passwords in the password system. Section 3 presents an implementation paradigm of password systems taking the form of a hierarchical bidimensional one-way chain. Special attention is paid to key distribution and revocation. Section 4 discusses our protection system from a number of salient viewpoints, which include the memory costs for password storage and the relation to previous work. Section 5 gives concluding remarks.

2. ACCESS PRIVILEGES

Let us refer to object type T, let op_0, $op_{1, \ldots}$ be the operations, and ar_0, ar_1, … be the access rights specified by T. For each operation, the definition of type T states the access rights that are necessary to execute this operation successfully. An *access privilege* is a collection of one or more access rights.

In our protection model, each access privilege is always part of a *privilege level*. The privilege levels of type T are ordered from the highest lv_0 to the lowest lv_{m-1}, where quantity m is specific to T. Privilege level lv_i consists an ordered sequence of access privileges $ap_{i,0}$, $ap_{i,1}$, … and different privilege levels may feature a different number of access privileges. A salient property is that, in level lv_i, the j-th access privilege $ap_{i,j}$ *inherits* the access rights of all the subsequent, weaker access privileges $ap_{i,j+1}$, $ap_{i,j+2}$,… etc.

When a subject creates an object, it is assigned access privilege $ap_{0,0}$, i.e. the first (strongest) access privilege of the highest privilege level lv_0. This privilege includes the *own* access right that incorporates the prerogatives of all the other access rights and also makes it possible

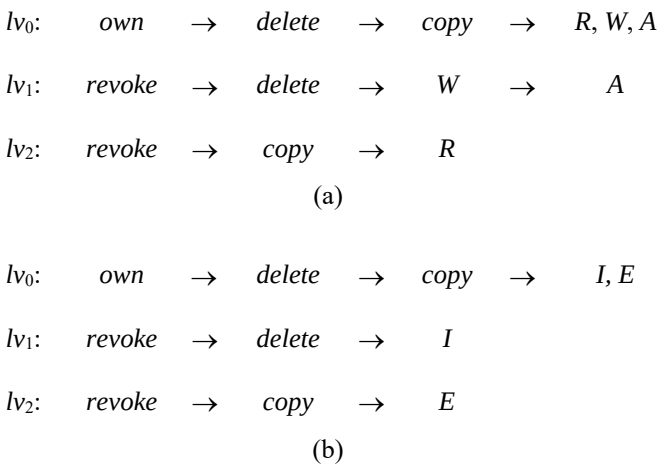

Figure 1. The privilege levels of: (a) the *Document* type; and (b) the *Buffer* type. In a compact notation, each access privilege is specified by only those access rights that are not inherited from the subsequent privileges.

to modify the composition of all access privileges. In all object types, the *revoke* access right for a given object makes it possible to revoke the access privileges for this object (access privilege revocation will be considered in subsequent Section 3.2); the *delete* access right makes it possible to delete the object; and, finally, the *copy* access right makes it possible to create copies of the object.

Figure 1 shows the privilege levels of two object types, the *Document* type and the *Buffer* type. Besides access rights *own*, *revoke*, *delete* and *copy*, the *Document* type defines three access rights, namely *read* (*R*), *write* (*W*) and *append* (*A*; Figure 1a). The *read* access right for a given document makes it possible to access the document for read; the *write* access right makes it possible to overwrite the document with new data; and the *append* access right makes it possible to add new data at the end of the document.

As stated previously, each access privilege of a given level inherits the access rights in all the subsequent access privileges of that level. Inheritance suggests a compact representation of access privileges whereby each privilege is denoted by the new access rights only, i.e. in Figure 1 we omit to specify the inherited access rights. The *Document* type defines three privilege levels. In level lv_0, the composition of the last (weakest) access privilege is (*R*, *W*, *A*). The previous access privilege includes access right *copy* and inherits access rights *R*, *W* and *A* from the subsequent privilege. Then we have a stronger access privilege that adds access right *delete*, and the first (strongest) privilege that includes access right *own*.

Besides *own*, *revoke*, *delete* and *copy*, the *Buffer* type defines access rights *insert* (*I*) and *extract* (*E*; Figure 1b). *Insert* makes it possible to put data items into the buffer, *extract* makes it possible to get data items from the buffer. The highest privilege level lv_0 is intended for the buffer owner, privilege level lv_1 is intended for producer subjects, and privilege level lv_2 is

intended for consumer subjects.

A subject that holds an access privilege of a given privilege level can distribute this privilege as well as any weaker privilege of the same level to other subjects. In the example of the *Document* type, access privilege $ap_{1,2}$ includes the *W* access right and inherits the *A* access right from the subsequent privilege $ap_{1,3}$; thus, the composition of $ap_{1,2}$ is (*W*, *A*). A subject that holds $ap_{1,2}$ can distribute this access privilege as well as subsequent privilege $ap_{1,3}$.

3. PASSWORDS AND KEYS

Function F is *one-way* if it is easy to evaluate but computationally hard to invert [15]. This means that given a value x it is easy to compute $F(x)$, but given a value y it is computationally unfeasible to find an x so that $y = F(x)$. A *one-way chain* is a sequence of values $x_0, x_1, \ldots, x_{n-1}$ such that $x_i = F(x_{i-1})$ for $0 < i \leq n - 1$, i.e. each value is the result of the application of one-way function F to the previous value. The first value x_0 is called the *seed* of the chain. A *hierarchical bidimensional one-way chain* is a collection of chains organized in two levels. At the first level, we have a single chain, which is called the *primary chain*. Each value of the primary chain is the seed of a chain at the second level (a *secondary chain*) [16].

In our protection model, a password is associated with each access privilege. The resulting password system takes the form of a bidimensional one-way chain. The primary chain includes the passwords for the strongest access privilege of all privilege levels. The *i*-th password of the primary chain is the seed of the *i*-th secondary chain whose values correspond to the passwords for the access privileges in privilege level lv_i. We shall denote the password for access privilege $ap_{i,j}$ by $w_{i,j}$.

In more detail, to set up the password system, password $w_{0,0}$ corresponding to the *own* privilege is chosen at random. This password is the seed of the primary chain, which features *m* values, one value for each privilege level. A one-way function, called the *primary function* PF, is used to generate the subsequent passwords in the primary chain, i.e. $w_{1,0} = PF(w_{0,0})$, $w_{2,0} = PF(w_{1,0})$, …, $w_{m-1,0} = PF(w_{m-2,0})$. These passwords are associated with the first privilege of each privilege level, from the highest level lv_0 to the lowest level lv_{m-1}. Password $w_{i,0}$ is the seed of the secondary chain for privilege level lv_i. A further one-way function, the *secondary function* SF_i, is used to set up the secondary chain for level lv_i, so we have $w_{i,1} = SF_i(w_{i,0})$, $w_{i,2} = SF_i(w_{i,1})$,… etc.[1]

[1] Primary function PF and all secondary functions SFs can be effectively implemented by taking advantage of a parametric one-way function and different parameters. A *parametric one-way function* H is a function that given a value z and a parameter p it is computationally unfeasible to find an x so that $z = H(x, p)$ [17]. Parametric one-

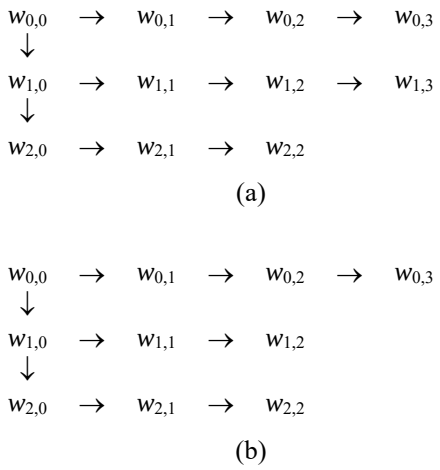

Figure 2. The password system for: (a) the *Document* type; and (b) the *Buffer* type. In both cases, the primary chain includes the passwords $w_{0,0}$, $w_{1,0}$ and $w_{2,0}$ for the strongest privilege of the three privilege levels. Each of these passwords is the seed of the corresponding secondary chain.

Figure 2 shows the password system for the two previous examples of object types, the *Document* type and the *Buffer* type. Let us refer to the *Document* type, for instance (Figure 2a). To set up the password system, password $w_{0,0}$ for the *own* privilege is chosen at random. A primary one-way function PF is used to generate the subsequent passwords $w_{1,0}$ and $w_{2,0}$ for the strongest privilege of the lower privilege levels. Each of these passwords is the seed of a secondary chain. For instance, for privileges level lv_1, password $w_{1,0}$ is the seed, and secondary one-way function SF_1 is used to generate passwords $w_{1,1}$ to $w_{1,3}$ in the corresponding secondary chain. A key matching password $w_{1,1}$ grants the corresponding access permission, which includes access right *delete*, and inherits access rights *write* and *append* from the subsequent access privileges in the first privilege level (see Figure 1a).

It is worth noting that, when a copy of a given object is generated by exercising the *copy* access right, the value of the new object is the same of the original, but the password system should be reconstructed so that the values of the passwords are different. This is necessary to avoid that a subject that holds a key for the original object can use this key to access the object copy.

3.1. Key distribution and derivation

A subject *S* that holds a key *k* matching a given password $w_{i,j}$ can transmit the access privilege corresponding to this password to other subjects by simply distributing a copy of the key to these subjects. Subject *S* may even distribute a key *k'* derived from *k* and relevant to a weaker

way functions are families of one-way functions that are parameterized by parameter *p*. In a practical implementation of a parametric one-way function, we can take advantage of a symmetric encryption cipher, e.g. if G is a hash function and $E_x(p)$ denotes the encryption of *p* using symmetric cipher E with key *x*, then $H(x, p) = G(E_x(p))$ is a parametric one-way function.

access privilege. A result of this type can be obtained by taking advantage of the structure of the password system and the password chains, as follows:

- Subject S that holds key k matching password $w_{i,0}$ for the strongest access privilege of privilege level lv_i can derive key k' matching password $w_{i',0}$ for the strongest access privilege of a subsequent privilege level $lv_{i'}$, $i' > i$. To this aim, S will apply primary function PF iteratively $i' - i$ times.
- Subject S that holds key k matching password $w_{i,j}$ for the j-th access privilege of privilege level lv_i can derive key k' matching password $w_{i,j'}$, $j' > j$ for a weaker access privilege of the same privilege level. To this aim, S will apply secondary function SF_i iteratively $j' - j$ times.
- Subject S that holds key k matching password $w_{i,0}$ for the strongest access privilege of privilege level i can derive key k' matching password $w_{i',j}$ for an access privilege of a lower privilege level $lv_{i'}$, $i' > i$. To this aim, S will apply primary function PF iteratively $i' - i$ times to find the key corresponding to the password $w_{i',0}$ of the strongest access privilege of privilege level i'. Then, this key will be used to evaluate key k' by applying the i-th secondary function SF_i iteratively j times.

Of course, a subject that holds a given key is not able to evaluate the key for a stronger access privilege in the same privilege level. This is a consequence of the fact that the passwords of a given privilege level are evaluated by iterated applications of a one-way function, which is computationally not invertible. Similarly, a subject that holds a key for the strongest access privilege of a given privilege level cannot evaluate the key for the strongest access privilege of a higher level.

3.2. Key revocation

Key revocation can be effectively obtained by changing the one-way functions.[2] In detail:

- If we replace secondary function SF_i with a new one-way function, we revoke all the keys for the access privileges of privilege level lv_i except the key for the strongest privilege of this level, $ap_{i,0}$. The new secondary function will be used to re-evaluate the passwords. A subject that holds a key for the strongest privilege of privilege level lv_i as well as of any higher privilege level can take advantage of the key derivation process illustrated in Section 3.1 to derive the new keys for the access privileges in lv_i.
- If we replace primary function PF, we revoke all the keys except those matching password

[2] In an implementation taking advantage of a parametric one-way function (see footnote 1), the change of the primary function or of a secondary function will be simply obtained by changing the corresponding parameter.

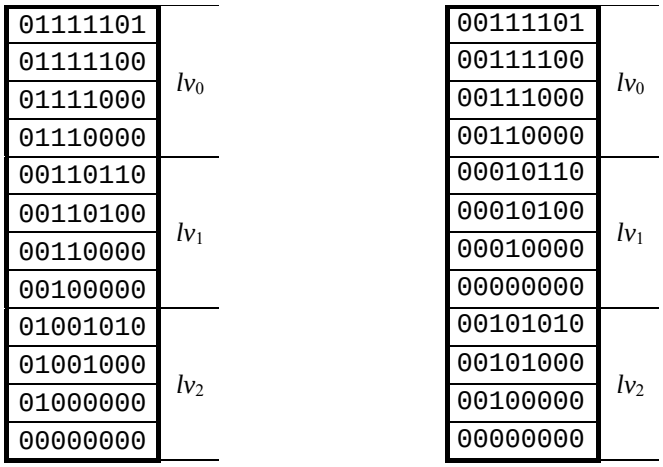

Figure 3. The privilege array for: (a) the *Document* type; and (b) the *Buffer* type. Starting from the least significant bit, in the *Document* type the access rights are in the order *own*, *revoke*, *delete*, *copy*, *write*, *append*, *read*; in the *Buffer* type, the order is *own*, *revoke*, *delete*, *copy*, *insert*, *extract*.

$w_{0,0}$. Consequently, the whole password system will be reconstructed. The new primary function will be used to re-evaluate the password for the strongest privilege of each privilege level. Then, each secondary function will be used to re-evaluate the passwords for the remaining access privileges of the corresponding privilege level. A subject that holds a key matching password $w_{0,0}$ will be in the position to derive the new keys.

4. DISCUSSION

4.1. Implementation issues

The composition of the access privileges of a given object in terms of access rights can be effectively represented in the form of a privilege array stored as part of the internal representation of the object. The privilege array features $m \cdot n$ elements, where m is the number of privilege levels and n is the maximum number of access privileges in a privilege level. Array element $i \cdot n + j$ corresponds to access privilege $ap_{i,j}$ (here, we hypothesize that access privileges are stored by privilege levels). In each array element, the r-th bit, if asserted, specifies that the r-th access right is part of the corresponding access privilege. If a privilege level includes less than n access privileges, the array elements corresponding to the lacking privileges will feature a bit configuration of all 0's.

In our previous examples of the *Document* and the *Buffer* types we have $m = 3$ and $n = 4$. In both cases, the privilege array will feature 12 elements (Figure 3). Let us consider the privilege array for the *Document* type, for instance (Figure 3a). We reserve a byte for each access privilege. The least significant bit of each byte corresponds to access right *own*, the subsequent

bit corresponds to access right *revoke*, then we have access rights *delete*, *copy*, *write*, *append* and *read*. Privilege level lv_2 contains only three access privileges, and consequently, the last element reserved for this privilege level contains all 0's.

As seen in Section 2, access right *own* allows us to change the definition of the access privileges in terms of access rights. This means that *own* makes it possible to modify the contents of the elements of the privilege array. Modifications will have to reflect access right inheritance between the access privileges of the same privilege level, i.e. the array element corresponding to a given privilege should include the access rights in the elements corresponding to all weaker privileges of the same privilege level.

4.2. Considerations concerning performance

A subject that executes operation *op* on object *G* of type *T* must present a key for this object. Execution of *op* should include the actions necessary to verify that the key matches a password in the password system of *G*, and that the access privilege corresponding to this password includes the access rights that are necessary for successful execution of *op*. Trade-offs are possible between the memory space necessary for storage of the password system and the time necessary to validate a key.

In a possible approach, the password system takes the form of a password array stored as part of the internal representation of the object. The password array features $m \cdot n$ elements, where m is the number of privilege levels and n is the maximum number of access privileges in a privilege level. Element $i \cdot n + j$ is reserved to contain password $w_{i,j}$ corresponding to access privilege $ap_{i,j}$. If a privilege level includes less than n access privileges, the array elements corresponding to the lacking privileges will feature a bit configuration denoting an invalid password, e.g. all 0's. In this approach, the expected number of key-password comparisons necessary to find the password matching a given key is $(m \cdot n + 1) / 2$.

Time efficiency will be significantly improved, and a single key-password comparison will be sufficient, if each key is stored in the form (k, i, j), where quantity k denotes the key value and quantities i and j indicate the corresponding password $w_{i,j}$. Of course, if value k does not match password $w_{i,j}$, key validation fails. The cost of a key in terms of memory space increases for the necessity to store quantities i and j. For up to 16 privilege levels and up to 16 access privileges in a privilege level, a four-bit nibble will be sufficient for each of these quantities, so the additional cost is a single byte, and is negligible.

Alternatively, we can take advantage of a small password array including only the m pass-

words in the primary chain. In this approach, verification of the validity of key (k, i, j) supposedly matching password $w_{i,j}$ implies evaluation of j passwords in the i-th secondary chain, starting from password $w_{i,0}$ in the direction of weaker access privileges. The expected number of password evaluations is n / 2.

If memory is an extremely scarce resource, we shall maintain a single password in the internal representation of the object, i.e. password $w_{0,0}$ corresponding to the *own* access privilege. In this case, verification of the validity of key (k, i, j) implies evaluation of i passwords in the primary chain to obtain password $w_{i,0}$, and j passwords in the i-th secondary chain to finally obtain password $w_{i,j}$.

4.3. Revocation of access privileges

As seen in Section 3.2, in our protection system, access privilege revocation is simply obtained by replacing a one-way function with a new function. By replacing secondary function FS_i we revoke the keys for all the access privileges of privilege level lv_i except the first privilege $ap_{i,0}$; by replacing primary function PF we revoke all keys except those for the *own* privilege $ap_{0,0}$. If a password array is used for password storage, after replacement of a one-way function all the passwords affected by the change should be recalculated and the new passwords should be inserted into the array.

In spite of its simplicity, our mechanism for the revocation of access privileges possesses a number of interesting properties [12]:

- Revocation is *transitive*. If a given key is revoked, all the copies of this key are revoked, too, independently of the subjects that hold these keys and the paths followed by these keys to reach these subjects. In fact, the copies of a given key are indistinguishable from the original. Remarkably, revocation propagates to all keys derived from a revoked key, so that it is not possible to take advantage of key derivation to circumvent revocation.
- Revocation is *selective*, that is, it can be limited to a subset of all the subjects that hold the access privilege being revoked. An effect of this type will be obtained by defining two or more access privileges in terms of the same set of access rights. These access privileges will be positioned in different privilege levels, so that they can be revoked independently of each other.
- Revocation is *temporal*, that is, it can be reversed through the same mechanism used for revocation. After a one-way function has been replaced, we can reverse the effects of revocation by restoring the original function.

4.4. Relation to previous work

Capability-based addressing

As seen in Section 1, in a protection system featuring subjects and protected objects, a basic problem is to specify the relation in terms of access rights existing between the subjects and the objects. A classical solution takes advantage of the concept of a *capability* [18]. This is a pair (*G*, *AR*), where quantity *G* identifies an existing object and quantity *AR* specifies a collection of access rights for this object. A subject that holds a capability for object *G* can access this object and perform the operations that are made possible by the access rights in *AR*. The object identifier must be unique system-wide and should never be reused, even after the corresponding object has been deleted from the system. This is necessary to avoid that a pre-existing capability is used to reference a new object. Furthermore, capabilities should be segregated in reserved memory areas. This is necessary to prevent a subject from altering the *AR* field of a capability, for instance, to purse an undue amplification of the access permissions, or even to forge a new capability from scratch.

Several solutions have been proposed to the capability segregation problem [19]. Capabilities can be stored in reserved memory segments, the *capability lists* [20], [21], [22]. In this approach, the representation of an object is complicated by the necessity to reserve a capability list for storage of the capabilities for the data segments containing the object value. Alternatively, in a system featuring hardware support for memory tagging, a specific tag configuration will indicate that the corresponding cell contains a capability [23], [24], [25]. Contrary to hardware standardization, this approach requires an *ad hoc* memory system. Hardware support is needed in the processor in terms of special instructions for capability processing.

Password capabilities

Password capabilities are an important improvement to the capability concept [8], [9], [10], [26], [27], [28]. In a password-capability environment, a set of passwords is associated with each object. Each password corresponds to a collection of access rights for that object. A password capability is a pair (*G*, *w*) where *G* is an object identifier and *w* is a password. Possession of a password capability makes it possible to access the object identified by *G* to execute the operations permitted by the access rights corresponding to password *w*. Password capabilities are protected from forging and tampering by the password length. If passwords are large and chosen at random, the probability of guessing a valid password to forge a password capability is vanishingly low. The association between object identifiers and passwords is logical rather than physical. A subject that holds two or more passwords for the same object needs to store

the object identifier only once. A password capability will be reconstructed when the object is actually accessed.

In the original formulation of the password capability concept, a subject that holds a given password capability is not in a position to autonomously generate a weaker password capability, corresponding to a subset of the access rights in the original password capability. An action of this type will be necessary if the subject is aimed at distributing only part of the access rights, for instance. In our previous example of the *Document* type, let us consider a subject that holds a password capability that grants access rights *write* and *append* and is aimed at distributing access right *append* only. In a situation of this type, the subject will have to ask for intervention of a password manager that forges the desired, weaker password capability.

We are now in a position to compare the main characteristics of the forms of protection systems, considered so far, i.e. capability lists, password capabilities and the password system presented in this paper. The points of view have been outlined in Section 1: segregation (i.e. the method to prevent forging), derivation (i.e. the method to restrict the extent of an access privilege by eliminating one or more access rights), and revocation (i.e. the ability to prevent further utilization of a given access privilege) [29]. In the capability list approach, segregation needs to be supported by ad-hoc mechanisms, e.g. capability lists or tagged memory, whereas no form of segregation is necessary in both the password capability approach and in the password system approach, provided that passwords are large and chosen at random.

Capability derivation is obtained by modifying the access right field to suppress the unwanted access rights. As a consequence of capability segregation, an action of this type should be supported at the machine instruction level, or by a protection system routine. In password capability systems, derivation requires the intervention of a password manager to forge the weaker password capability. In contrast, in a password system, a subject is in a position to carry out password derivation autonomously, taking advantage of the structure of the password chains.

Finally, in capability-based protection environments, revocation needs to be supported by *ad-hoc* mechanisms whose complexity tends to subvert the simplicity of access right transmission between subjects, e.g. a propagation graph associated with each given capability, which keeps track of all successive transferals of this capability [12], or short-lived capabilities whose validity has to be renewed periodically [30]. In password capability environments, revocation can be obtained by a password replacement, whose effects, however, do not extends automati-

cally to all derived password capabilities. In our password systems, revocation can be effectively obtained by changing the one-way functions.

5. CONCLUDING REMARKS

In a protection environment featuring subjects and protected objects, we have considered the problem of specifying the access privileges held by the subjects on the objects. In our solution, the access privileges are organized in privilege levels. A password system is associated with each object, featuring a password for each access privilege. A subject can access a given object if it holds a key matching one of the passwords in the password system of this object, and the access privilege corresponding to this password includes the access rights necessary to accomplish the access.

The password system is a significant extension of the classical concept of a password capability. The following is a brief summary of the main results we have obtained:

- A subject that holds a key for a given access privilege is in a position to autonomously derive and distribute keys for weaker privileges in the same privilege level as the original key. Furthermore, a subject that holds a key for the strongest access privilege in a given privilege level can derive keys for the access privileges in this privilege level as well as in the lower levels. No intervention of a form of password manager is necessary for key derivation. This is made possible by our implementation of the password system as a hierarchical bidimensional one-way chain.
- A subject that holds a key for the *revoke* access privilege of a given privilege level can revoke the keys for the access privileges in that privilege level as well as in the lower levels. Exceptions are the keys for the strongest access privilege of each privilege level, which can be revoked only presenting a key for the *own* privilege. Revocation results to be transitive, selective and temporal.
- Trade-offs are possible between the memory requirements for storage of a password system and the time necessary for key validation. If all the passwords for a given object are stored in a password array, a single key-password comparison is sufficient to validate a key. Alternatively, we can store only the passwords for the strongest privilege of each privilege level, or even only the password for the *own* privilege, but in this case several passwords must be re-evaluated to validate a key.

The resulting protection system provides a highly flexible and efficient mechanism to grant and revoke access rights to and from subjects, which makes it particularly suitable for critical

infrastructure systems.

ACKNOWLEDGMENT

This work has been partially supported by the TENACE PRIN Project (Grant no. 20103P34XC_008) funded by the Italian Ministry of Education, University and Research.

REFERENCES

[1] A. A. Cárdenas, S. Amin, S. Sastry, "Research challenges for the security of control systems," *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security*, San Jose, CA, USA, July 2008.

[2] A. N. Bessani, P. Sousa, M. Correia, N. F. Neves, P. Verissimo, "The Crutial way of critical infrastructure protection," *IEEE Security & Privacy*, vol. 6, no. 6 (November-December 2008), pp. 44–51.

[3] A. M. Grilo, J. Chen, M. Diaz, D. Garrido, A. Casaca, "An integrated WSAN and SCADA system for monitoring a critical infrastructure," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 3 (August 2014), pp. 1755–1764.

[4] National Institute of Standars and Technology, *Framework for Improving Critical Infrastructure Cybersecurity*, version 1.0, February 2014. Available (February 2015) at: http://www.nist.gov/cyberframework/

[5] T. A. Linden, "Operating system structures to support security and reliable software," *ACM Computing Surveys*, vol. 8, no. 4 (1976), pp. 409–445.

[6] R. S. Sandhu, P. Samarati, "Access control: principle and practice," *IEEE Communications Magazine*, vol. 32, no. 9 (September 1994), pp. 40–48.

[7] M. Anderson, R. D. Pose, C. S. Wallace, "A password-capability system," *The Computer Journal*, vol. 29, no. 1 (February 1986), pp. 1–8.

[8] L. Lopriore, "Password capabilities revisited," *The Computer Journal*, first published online November 11, 2013; doi:10.1093/comjnl/bxt131

[9] J. Vochteloo, S. Russell, G. Heiser, "Capability-based protection in the Mungi operating system," *Proceedings of the Third International Workshop on Object Orientation in Operating Systems*, Asheville, North Carolina, USA, December 1993.

[10] M. D. Castro, R. D. Pose, C. Kopp, "Password-capabilities and the Walnut kernel," *The Computer Journal*, vol. 51, no. 5 (2008), pp. 595–607.

[11] J. Bacon, "Generic support for distributed applications," *Computer*, vol. 33, no. 3 (2000), pp. 68–76.

[12] V. D. Gligor, "Review and revocation of access privileges distributed through capabilities," *IEEE Transactions on Software Engineering*, vol. SE-5, no. 6 (November 1979), pp. 575–586.

[13] P. A. Karker, "New methods for immediate revocation," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, California, USA, May 1989.

[14] L. Lopriore, “Protection structures in multithreaded systems,” *The Computer Journal*, vol. 56, no. 4 (April 2013), pp. 478–496.

[15] L. Lamport, “Password authentication with insecure communication,” *Communications of the ACM*, vol. 24, no. 11 (November 1981), pp. 770–772.

[16] Y.-C. Hu, M. Jakobsson, A. Perrig, “Efficient constructions for one-way hash chains,” *Proceedings of the Third International Conference on Applied Cryptography and Network Security*, New York, NY, USA, June 2005; in: *Lecture Notes in Computer Sciences*, vol. 3531, Berlin, Heidelberg: Springer-Verlag, 2005.

[17] W. Trappe, J. Song, R. Poovendran, K. R. Liu, “Key management and distribution for secure multimedia multicast,” *IEEE Transactions on Multimedia*, vol. 5, no. 4 (2003), pp. 544–557.

[18] H. M. Levy, *Capability-Based Computer Systems*. Bedford, Mass.: Digital Press, 1984.

[19] L. Lopriore, “Encrypted pointers in protection system design,” *The Computer Journal*, vol. 55, no. 4 (April 2012), pp. 497–507.

[20] R. P. Colwell, E. F. Gehringer, E. D. Jensen, “Performance effects of architectural complexity in the Intel 432,” *ACM Transactions on Computer Systems*, vol. 6, no. 3 (1988), pp. 296–339.

[21] D. M. England, “Capability concept mechanisms and structure in System 250,” *Proceedings of the International Workshop on Protection in Operating Systems*, IRIA, Paris, 1974, pp. 63–82.

[22] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, S. Winwood, “seL4: formal verification of an OS kernel,” *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, Big Sky, Montana, USA, October 2009, pp. 207–220.

[23] K. Ghose, P. Vasek, “A fast capability extension to a RISC architecture,” *Proceedings of the 22nd EUROMICRO Conference*, Prague, Czech Republic, September 1996, pp. 606–613.

[24] G. J. Myers, B. R. S. Buckingham, “A hardware implementation of capability-based addressing,” *ACM SIGOPS Operating Systems Review*, vol. 14, no. 4 (1980), pp. 13–25.

[25] P. G. Neumann, R. J. Feiertag, “PSOS revisited,” *Proceedings of the 19th Annual Computer Security Applications Conference*, Las Vegas, NV, USA, December 2003, pp. 208–216.

[26] J. S. Chase, H. M. Levy, M. J. Feeley, E. D. Lazowska, “Sharing and protection in a single-address-space operating system,” *ACM Transactions on Computer Systems*, vol. 12, no. 4 (1994), pp. 271–307.

[27] G. Heiser, K. Elphinstone, J. Vochteloo, S. Russell, J. Liedtke, “The Mungi single-address-space operating system,” *Software — Practice and Experience*, vol. 28, no. 9 (July 1998), pp. 901–928.

[28] R. Pose, “Password-capabilities: their evolution from the Password-Capability System into Walnut and beyond,” *Proceedings of the Sixth Australasian Computer Systems Architecture Conference*, Gold Coast, Australia, January 2001, pp. 105–113.

[29] L. Lopriore, “Password management: distribution, review and revocation,” *The Computer Journal*, first published online November 9, 2014; doi:10.1093/comjnl/bxu125

[30] A. W. Leung, E. L. Miller, “Scalable security for large, high performance storage systems,” *Proceedings of the Second ACM Workshop on Storage Security and Survivability*, Alexandria, Virginia, USA, October 2006, pp. 29–40.

Gianluca Dini is an associate professor of computer engineering at the Dipartimento di Ingegneria dell'Informazione of the University of Pisa, Pisa, Italy. His research interests include cybersecurity, distributed systems, networked embedded systems and wireless sensor networks. He made research in the areas of key management and group communication.

Lanfranco Lopriore is a full professor of computer engineering at the Dipartimento di Ingegneria dell'Informazione of the University of Pisa, Pisa, Italy. His research interests include protection systems, password-capability systems, single address space architectures, and wireless sensor networks. He made research in the areas of cache memories and program debugging environments.