International Journal of Information Security manuscript No.
(will be inserted by the editor)

Access right management by extended password capabilities

Lanfranco Lopriore

Received: date / Accepted: date

Abstract With reference to a classic protection system featuring active subjects that reference protected objects, we approach the important problem of identifying the objects that each subject can access, and the operations that the subject can carry out on these objects. Password capabilities are a classical solution to this problem. We propose a new form of password capability, called extended password capability (or e-capability, for short). An e-capability can specify any combination of access rights. A subject that holds a given e-capability can generate new e-capabilities for reduced sets of access rights. Furthermore, a subject that created a given object is in a position to revoke the access permissions granted by every e-capability referencing this object, completely or in part. The size of an e-capability is comparable to that of a traditional password capability. The number of passwords that need to be stored in memory permanently is kept to a minimum, and is equal to a single password for each object.

Keywords Access right · Distribution · Password · Reduction · Revocation

1 Introduction

We shall refer to a classic protection system featuring active entities, called *subjects*, that reference passive entities, the protected *objects* [15], [22]. In a system of this type, an important problem is to identify the objects that each subject can access, and the operations that the subject can carry out on these objects [24]. We shall refer to typed objects, so that each given object B is associated with a type T. The definition of T states a set of operations $op_0, op_1, \ldots, op_{p-1}$, and a set of access rights $ar_0, ar_1, \ldots, ar_{n-1}$, where quantities p and n are type-specific. The type definition also states the subset of all access rights that is necessary to accomplish each given operation successfully.

L. Lopriore
Dipartimento di Ingegneria dell'Informazione, Università di Pisa, via G. Caruso 16, 56126 Pisa, Italy.
Tel.: +39-050-2217511
Fax: +39-050-2217522
E-mail: l.lopriore@iet.unipi.it

1.1 Capabilities

In a classical approach, the protection problem is solved by taking advantage of capability-based techniques [14]. A *capability* C is a pair (B, AR), where B is the identifier of a protected object, and AR is a set of access rights for this object. A subject that holds capability C can take advantage of this capability to access object B to carry out the operations that are made possible by the access rights in AR. In a typical implementation, the AR field is encoded in n bits, one bit for each access right defined by the type T of B; if a given bit is asserted, the capability grants the corresponding access right.

Capabilities should be segregated in protected memory regions [4]. This means that a process should be prevented from using ordinary machine instructions to modify a capability, for instance, to set bits in the AR field to obtain an undue amplification of access rights, or even to modify the object identifier to forge a new capability referencing a different object. Solutions have been devised to the capability segregation problem. In a segmented memory environment, special segments, which we shall call *capability segments*, can be used for capability storage [5], [11], [27]. The instruction set of the processor is augmented with a set of special instructions for capability processing, the *capability instructions*. If an ordinary machine instruction is used to access the contents of a capability segment, an exception of violated protection is raised, and execution of the instruction terminates with failure. This approach leads to segment proliferation and an undue complication of the structure of the protected objects, e.g. at least one capability segment is necessary for each given object to store the capabilities referencing the data segments reserved for the internal representation of this object.

In a different approach, capability segregation is obtained by taking advantage of a tagged memory [9], [19]. In this approach, a one-bit tag is associated with each memory cell. The tag specifies whether this cell contains a capability, or an ordinary information item. A capability instruction can be executed on a given memory cell only if this cell is tagged to contain a capability; if this is not the case, execution of the capability instruction generates an exception of violated protection. This approach requires special memory modules apt to store the cell tags, and is contrary to the requisite of hardware standardization.

1.2 Password capabilities

Password capabilities are an important improvement on the classical capability concept [1], [3], [8], [16], [20]. In the password capability approach, a collection of passwords is associated with each object. Each password corresponds to a set of access rights for this object. A password capability P is a pair (B, W), where B is the identifier of a protected object, and W is a password. A subject that possesses P can access object B to carry out the actions permitted by the access rights associated with W. If passwords are large and randomly distributed, it is virtually impossible to guess a password to forge a valid password capability [2]. It follows that password capabilities do not need to be segregated in memory. They can be freely mixed with ordinary information items, and are an effective solution to the segregation problem.[1]

A peculiar problem of password capabilities is password proliferation. If we are aimed at distributing several distinct subsets of access rights, a password should be associated with each of these subsets. Alternatively, we shall reserve a password for each access right, and in this case, several password capabilities will be necessary to express a complex access permission defined in terms of several access rights. Of course, this is an undue complication of access right management.

As an example of an object type, let us refer to the *File* type that defines four access rights, namely *delete*, *write*, *read* and *execute*. The *delete* access right for a given file F makes it possible to delete F; as will be illustrated later, this access right is also necessary to revoke the access permissions for the file. The *write* access right allows us to access F for write. This is similar to the *read* access right for read access, and to the *execute* access right for the execution of the file contents (supposedly, machine code). In this example, if subject S should possess access rights *write*, *read* and *execute*, and a password exists that corresponds to these three access rights, S will be granted a password capability defined in terms of this password. If the password does not exist, it is generated [1], [2]. Alternatively, we can associate a password with each access right, for a total of four passwords; in this case, S will possess three password capabilities, defined in terms of the read password, the write password, and the execute password, respectively.

In a different approach, we can modify the definition of a password capability to contain several passwords. In this approach, a generalized password capability G assumes the form $(B, W_0, W_1, \ldots)$, where $W_0, W_1, \ldots$ are passwords for object B. The access rights granted by G on B are those specified by these passwords. Of course, we can define a generalized password capability corresponding to an arbitrary combination of access rights. However, the size of generalized password capabilities is variable, and can be very large for several access rights. In our example of the *File* type, a generalized password capability specifying access rights *write*, *read* and *execute* will contain three passwords. If the size of an object identifier is 64 bits, and the size of a password is 128 bits, the size of this generalized password capability is 56 bytes.

1.3 Reduction

An important feature of capability-based protection is simplicity in access right transmission between subjects. A subject S that holds capability $C = (B, AR)$ specifying a set AR of access rights for object B can grant these access rights to a different subject S' simply by transferring a copy of C to S'. A related aspect is that

[1] If a subject steals a password capability, it can take advantage of this password capability, to access the object it references illegitimately. In fact, the validity of a password capability is independent of the subject that holds this password capability and extends system-wide, and a copy of a password capability cannot be distinguished from the original. This is a different aspect of the segregation problem. Password capability stealing can be precluded by a separation of the address spaces enforced by the underlying operating system kernel [2]. Alternatively, we can assign a cryptographic key to each application; the password capabilities held by the subjects of a given application are encrypted by using the key of this application [16]. This mechanism prevents stealing between subjects of different applications, but cannot protect the subjects of the same application, which should be considered mutually trustworthy.

of capability *reduction*, i.e. to transform C into a new capability $C' = (B, AR')$, where AR' specifies a subset of the access rights contained in AR, so that subject S can transfer only part of the access rights in the original capability C. To this aim, the instruction set of the processor will be extended to include capability instructions aimed at altering the access right field in a strictly controlled fashion, so that any form of access right amplification is prevented.

Access right reduction is arduous in password capability environments [16]. Suppose that subject S holds password capability $P = (B, W)$ referencing object B with password W, and let AR be the set of access rights associated with W. Suppose also that S is aimed at granting subject S' a password capability $P' = (B, W')$ for the same object B, where W' corresponds to a subset AR' of AR. In a situation of this type, intervention of a software component is necessary, which we shall call the *password capability manager* M_B, associated with B and aimed at password capability reduction. Subject S sends password capability P to M_B. In turn, M_B transforms P into P', and returns P' to S. If no password exists that corresponds to the reduced set of access rights AR', a new password is generated, and is associated with the object [1], [2].

In our example of the *File* type, suppose that subject S holds password capability $P = (F, W)$ for file F, where W corresponds to access rights *write*, *read*, and *execute*. Suppose also that S is aimed at granting subject S' permission to read and to execute (but not to write) the file. To this aim, S will ask for intervention of the password capability manager M_F of file F. M_F will transform password capability P into a new password capability $P' = (F, W')$ for file F, where password W' corresponds to access rights *read* and *execute*. If no such password exists, it is generated. Alternatively, M_F will return two password capabilities, one for access right *read* and one for access right *execute*. This is an undue complication of the whole password management process.

Of course, generalized password capabilities featuring several passwords are an effective solution to the problem of access right reduction. Let $G = (F, W_{write}, W_{read}, W_{execute})$ be a generalized password capability for file F, where W_{write}, W_{read} and $W_{execute}$ are the passwords for access rights *write*, *read* and *execute*, respectively. Subject S that holds G can apply reduction to eliminate access right *write* simply by removing password W_{write} from G to obtain generalized password capability $G' = (F, W_{read}, W_{execute})$.

In this paper, we approach the problem of access right representation in password capabilities. We propose a new model of password capability, which we call *extended password capability*, or *e-capability* for short. In this model:

- An e-capability can specify any combination of access rights;
- The size of an e-capability is comparable to that of a traditional password capability featuring a single password, and is independent of the number of access rights granted by this e-capability;
- A subject that holds a given e-capability is in a position to reduce this e-capability to eliminate one or more access rights. The subject can carry out this action autonomously; no intervention is required of a form of password capability manager.

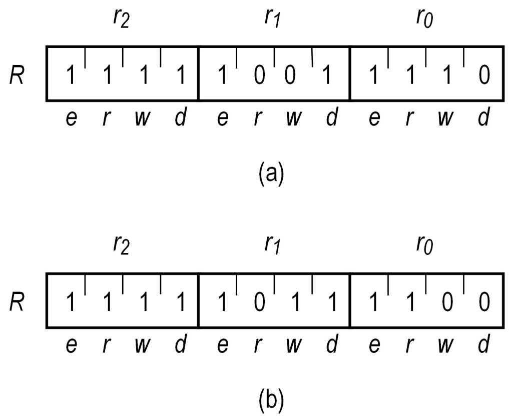
The rest of this paper is organized as follows. Sect. 2 introduces the e-capability model with special reference to the relation existing, in an e-capability referencing a given object, between the password and an *owner password* generated when the object is created. E-capability validation and reduction are analyzed in depth. Sect. 3 extends the e-capability concept by *classes* aimed at supporting the review of access permissions, so that the owner of a given object is given the ability to revoke the validity of the e-capabilities referencing this object. Sect. 4 discusses the motivations for the e-capability model with respect to a number of important viewpoints, which include the fraudulent forging of e-capabilities, the properties of the proposed form of access right revocation, e-capability reduction, and the memory requirements for password storage. Sect. 5 gives concluding remarks.

2 The e-capability model

Let T be an object type, let $ar_0, ar_1, \ldots, ar_{n-1}$ be the access rights defined by T, and let B be an object of type T. An e-capability E is a triple (B, W, R), where W is a password and R is the *reduction field.* The reduction field encodes the access rights granted by E; as will be made clear later, this field also specifies the relation existing between password W and a password, called the *owner password* and denoted by W_{own}, which is associated with B and corresponds to full access rights.

Reduction field R is partitioned into $n-1$ *reduction subfields*, $r_0, r_1, \ldots, r_{n-2}$, of size n bits. In a reduction subfield, the i-th bit corresponds to access right ar_i. The access rights granted by e-capability E are expressed by quantity $AR = r_0 \wedge r_1 \wedge \ldots \wedge r_{n-2}$, i.e. the result of the logical AND of all the reduction subfields; if the i-th bit is set in AR, then E includes access right ar_i. This means that, for each bit that is cleared in a given reduction subfield, this subfield eliminates the corresponding access right from the result of the preceding subfields, and consequently, from E. A reduction subfield of all 1's is called a *flat* subfield and eliminates no access right at all. All flat subfields are always placed at the highest order numbers, in the most significant positions of the R field. If all subfields are flat, i.e. $R = 11\ldots1$, we have a situation of no reduction at all. In this situation, e-capability E grants full access rights, and $W = W_{own}$.

In our example of the *File* type, we have four access rights, so $n = 4$. An e-capability $E = (F, W, R)$ referencing file F with password W features a reduction field R of size 12 bits, which is partitioned into three subfields of size 4 bits (Fig. 1). In each subfield, let us suppose that bit 0 corresponds to access right *delete*, and bits 1 to 3 correspond to access rights *write* (w), *read* (r) and *execute* (e), respectively. If the reduction field is configured as is illustrated in Fig. 1a, the least significant bit of subfield r_0 is cleared to indicate that access right *delete* is lacking from E; this is similar to the meaning of subfield r_1 for access rights *write* and *read.* The remaining subfield r_2 is flat, so it produces no access right reduction. We may conclude that e-capability E grants access right *execute* on file F. In the configuration of Fig. 1b, bits 0 and 1 of subfield r_0 are cleared to indicate that access rights *delete* and *write* are lacking from E; this is similar to the meaning of subfield r_1 for access right *read.* The remaining subfield r_2 is flat. In this case, too, the resulting e-capability grants access right *execute* on F. As will be illustrated shortly, different configurations of the reduction field always correspond to different passwords, even if these configurations specify the same set of access rights. In fact, we may have several passwords for the same set of access rights. However, only one password, the owner password W_{own}, needs to



(a)

(b)

Fig. 1 Two different configurations of the reduction field R, both corresponding to access right *execute*. The bits in each subfield, from bit 0 to bit 3, correspond to access right *delete* (d), *write* (w), *read* (r) and *execute* (e), respectively.

be permanently stored with the object, so the memory requirements for password storage are low.

2.1 Password derivation

Function h is *one-way* if it is easy to compute but hard to invert [12]. This means that given a value x, it is computationally easy to compute $h(x)$, but given a value y, it is computationally unfeasible to determine a value x such that $y = h(x)$. The design and implementation efforts connected with the construction of a one-way funciton can be kept to a minimum starting from a good cryptosystem [17], [21]. For instance, a publicly known constant c is encrypted using x as the key, i.e., if E_x is a symmetric cypher, we have $h(x) = E_x(c)$ [23]. Function $h_r(x)$ is a *parametric one-way function* if, given a value y and a parameter r, it is computationally unfeasible to determine a value x such that $y = h_r(x)$ [26]. Thus, a parametric one-way function is a family of one-way functions, one function for each value of parameter r, and can be implemented as $h_r(x) = E_x(r)$ [23]. In the following, we shall take advantage of a parametric one-way function in password derivation.

When subject S creates an object B, the owner password W_{own} of this object is generated at random. Then, e-capability $E_{own} = (B, W_{own}, 11\ldots1)$ referencing B is assembled and is returned to S. E_{own} is called the *owner e-capability.* In E_{own}, all the subfields of the reduction field are flat, and the password is the owner password. Thus, E_{own} grants full access rights. All the other passwords of B are derived from W_{own} by taking advantage of a password derivation mechanism based on the iterated application of a parametric one-way function $h_r(W)$, which we call the *password generation function.* The parameter of h is a reduction subfield, and the argument is a password.

In detail, let us refer to e-capability $E = (B, W, R)$, and let $r_0, r_1, \ldots, r_{n-2}$ be the subfields of reduction field R. Password W corresponding to R is obtained iteratively, by evaluating $W_{i+1} = h_{r_i}(W_i), i = 0, 1, \ldots, n-2$, where $W_0 = W_{own}$ and $W = W_{n-1}$. If one or more reduction subfields are flat, the iterations terminate at the subfield that precedes the first flat subfield (it should be recalled that all flat subfields are always placed in the most significant positions of the reduction field). Thus, password W is the last of a sequence of passwords that starts from owner password W_{own}. Each password in this sequence corresponds to a set of access rights smaller than that associated with the previous password.

In the example of Fig. 1a, password W is obtained starting from W_{own}. Reduction subfield r_0 is 1110 (14 in decimal notation), thus we have $W_1 = h_{14}(W_{own})$. In the next step, reduction subfield r_1 is 1001, thus we have $W_2 = h_9(W_1)$. Reduction subfield r_2 is flat; this terminates the iterations, and $W = W_2$. In the example of Fig. 1b, we start from W_{own}, then we have $W_1 = h_{12}(W_{own})$ and finally, $W = W_2 = h_{11}(W_1)$. In both cases, the resulting password corresponds to access right *execute*. Thus we have two different passwords for the same access right.

2.2 E-capability validation

In our system, all authentication and authorization decisions are taken locally to the protected object being accessed [10]. In detail, when e-capability $E = (B, W, R)$ is presented to object B to execute operation *op*, execution includes the actions necessary to validate E. The e-capability is *valid* if password W matches the password that is obtained by applying password generation function h iteratively, starting from the owner password W_{own} of B and using the reduction subfields that are not flat. Execution of *op* is as follows:

1. The access rights granted by E on B are evaluated. To this aim, quantity $AR = r_0 \wedge r_1 \wedge \ldots \wedge r_{n-2}$ is computed, where r_i denotes the i-th subfield of reduction field R.
2. If AR does not include the access rights that are necessary to execute operation *op*, an exception of violated protection is raised, and execution of *op* terminates with failure; otherwise
3. The subfields $r_0, r_1, \ldots, r_{n-2}$ of reduction field R are used to evaluate password W' corresponding to R, by applying password generation function h iteratively, starting from the owner password W_{own} of B, as follows: $W_0 = W_{own}$, $W_1 = h_{r_0}(W_0)$, $W_2 = h_{r_1}(W_1)$, ... etc. The sequence terminates at the reduction subfield, say r_i, that precedes the first flat subfield, and in this case $W' = W_{i+1}$; if no reduction subfield is flat, the sequence terminates at the last reduction subfield r_{n-2}, when $W' = W_{n-1}$.
4. If $W' \neq W$, i.e. the password evaluated at step 3 does not match the password in e-capability E, an exception of violated protection is raised, and execution of *op* terminates with failure; otherwise
5. The actions involved in the execution of *op* are carried out.

2.3 E-capability reduction

Let us consider a subject S that holds e-capability $E = (B, W, R)$ for object B of type T, where W is a password, and R is a reduction field. Let $ar_0, ar_1, \ldots, ar_{n-1}$ be the access rights defined by T, let $r_0, r_1, \ldots, r_{n-2}$ be the subfields of R, and let AR be the set of access rights specified by E, which is given by relation $AR = r_0 \wedge r_1 \wedge \ldots \wedge r_{n-2}$. Subject S can grant these access rights to a different subject S' by transferring a copy of E to S'. Subject S may even grant a reduced set of access rights to S'. To this aim, S will forge a new e-capability $E' = (B, W', R')$ corresponding to this reduced set. An action of this type is called an e-capability *reduction*.

In detail, access right ar_i can be eliminated from E by carrying out the actions that follow:

1. Quantity R' is obtained by modifying the first subfield of R that is flat, say subfield r^*, to clear the i-th bit, which corresponds to ar_i;
2. Quantity W' is obtained by applying password generation function h, i.e. $W' = h_{r^*}(W)$.

At step 1, an e-capability can be reduced only if it specifies at least two access rights, and consequently, it contains at least one flat reduction subfield; in fact, we have n access rights and $n - 1$ reduction subfields. Of course, reduction may well involve more than a single access right, and in this case more bits will be cleared in subfield r^*, i.e. the bits corresponding to each of these access rights.

3 Access right revocation

As seen in Sect. 1, a salient feature of capability-based protection is simplicity in capability transmission between subjects. If e-capabilities are used, this feature is made even more evident by the fact that e-capabilities do not need to be segregated in memory. It follows that access rights tend to spread throughout the system. A related problem is access right *revocation*; a subject that holds the *delete* access right for a given object should be granted the ability to revoke the validity of the e-capabilities referencing this object, and the effects of the revocation should extend system-wide.

3.1 Classes

We shall now extend the e-capability model, presented in the foregoing sections, to introduce effective support to e-capability revocation. As seen in Sect. 2 and illustrated in Fig. 1, different e-capabilities may co-exist in memory, which grant the same set of access rights and originate from different sequences of reduction subfields. The extension we are introducing now is based on the concept of e-capability *class*: different e-capabilities can originate from the *same* sequence of reduction subfields, if these e-capabilities belong to different classes.

Let us refer to the generic e-capability $E = (B, W, R, C)$. In this new form, we have a *class field*, which is denoted by C. As seen in Sect. 2, the set AR of the access rights granted by E is expressed by relation $AR = r_0 \wedge r_1 \wedge \ldots \wedge r_{n-2}$, i.e. the logical AND of all the subfields of reduction field R. These access rights are *nominal*, and can be revoked. The class field is used to determine the set EAR of the *effective* access rights corresponding to the nominal access rights specified by AR, as follows. Each object maintains a table, called the *revocation table RT*, featuring one entry for each value of the class field. The size of an entry is equal to

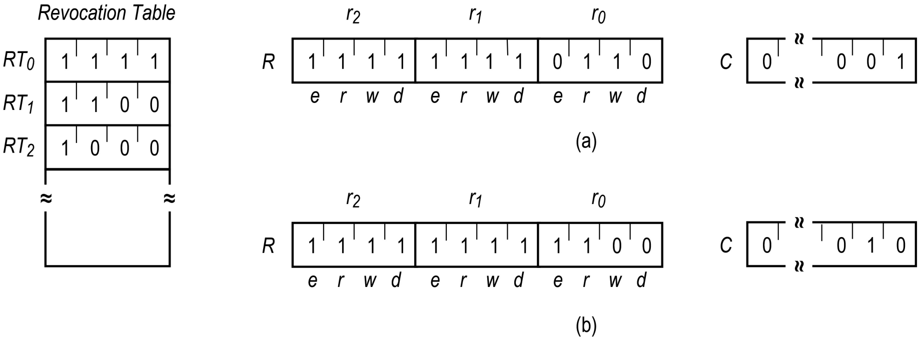


Fig. 2 Revocation table of an object of the *File* type, and different configurations of the R and C fields.

the size of a reduction subfield, i.e. n bits. Entry RT_c, corresponding to a specific value c of the class field C, specifies a possible revocation of access rights, as results from relation $EAR = AR \wedge RT_c$. This means, in particular, that if RT_c contains all 0's, e-capability E is completely revoked (it cannot be used for successful object access). A constraint for RT is that RT_0 always contains all 1's. This means that an e-capability in class 0 (i.e. with $C = 0$) grants all the access rights in AR (all the nominal access rights are effective), and cannot be revoked.

With reference to our example of the *File* type and an object of this type, Fig. 2 shows the revocation table and two configurations of the reduction and class fields. Entry RT_0 contains all 1's, and in fact, an e-capability in class 0 cannot be revoked. In entry RT_1, two bits are cleared, i.e. bit 0 corresponding to access right *delete*, and bit 1 corresponding to access right *write*; it follows that these access rights are revoked in all the e-capabilities in class 1. Finally, in entry RT_2 a single bit is asserted, i.e. bit 3 corresponding to access right *execute*; it follows that all access rights except *execute* are revoked in the e-capabilities in class 2. In the configuration of the reduction and class fields shown in Fig. 2a, the nominal set of access rights AR, given by relation $r_0 \wedge r_1 \wedge r_2$, includes access rights *write* and *read.* The value of the class field is 1, and consequently, access right *write* is revoked. In the configuration of Fig. 2b, AR includes access rights *read* and *execute.* The value of the class field is 2, and consequently, access right *read* is revoked.

3.2 E-capability generation

When subject S creates a new object B, all the entries of the revocation table of this object are initialized to all 1's. Subject S receives the owner e-capability E_{own} that grants all access rights for B, including the *delete* access right. The class of E_{own} is 0; its composition is $E_{own} = (B, W_{own}, R, C)$, where $R = (r_0, r_1, \ldots, r_{n-2})$, all r_i's are flat, and $C = 0$. E_{own} cannot be revoked, as follows from the fact that, for class 0, the corresponding entry RT_0 of the revocation table always contains all 1's. The *delete* access right in E_{own} makes it possible to change the contents of the revocation table.

Owner e-capability E_{own} can be used to generate e-capabilities for object B in different classes. In detail, an e-capability $E = (B, W', R', C')$ with full access rights and a class $c' \neq 0$ can be generated starting from E_{own}, as follows:

1. The class field C' contains quantity c';
2. All the subfields of reduction field R' are flat;
3. Quantity W' is obtained by using password generation function h, i.e. $W' = h_{c'}(W_{own})$.

We wish to remark that both the owner e-capability E_{own} and the new e-capability E include full access rights, however E_{own} is in class 0 and is not subject to revocation, whereas E is in class $c' \neq 0$ and can actually be revoked. Furthermore, e-capability E cannot be used to generate an e-capability in a different class, as this action requires a knowledge of the owner password W_{own}, which is only contained in E_{own}. As will be shown shortly, this important requisite prevents actions of e-capability forging.[2]

3.3 E-capability validation

When a subject presents e-capability $E = (B, W, R, C)$ to object B to execute a given operation *op*, execution of *op* includes the actions necessary for validation of E. Let c be the class of E, as is specified by the class field C. Execution is as follows:

1. Quantity $EAR = AR \wedge RT_c$ is evaluated, as illustrated in Sect. 3.1.
2. If EAR does not include the access rights required by *op*, execution of *op* fails; otherwise
3. The contents c of class field C and subfields $r_0, r_1, \ldots, r_{n-2}$ of reduction field R are used to evaluate password W' corresponding to C and R, as follows. First we have $W_0 = W_{own}$, or, if $c \neq 0$, $W_0 = h_c(W_{own})$. Then, we apply password generation function h iteratively, starting from W_0 to obtain W', as follows: $W_1 = h_{r_0}(W_0)$, $W_2 = h_{r_1}(W_1)$, $\ldots$ etc. The sequence terminates at the reduction subfield, say r_i, that precedes the first flat subfield, and in this case $W' = W_{i+1}$; if no reduction subfield is flat, the sequence terminates at the last subfield r_{n-2}, when $W' = W_{n-1}$.
4. If $W' \neq W$, i.e. the password evaluated at step 3 does not match the password in e-capability E, then execution of *op* fails; otherwise
5. The actions involved in the execution of *op* are carried out.

[2] Suppose that subject S transfers a copy of the owner e-capability referencing object B to subject S'. As a result, S' acquires full access rights for B, including the *delete* access right that makes it possible to delete the object and to modify its revocation table. In fact, there is no way to distinguish the original owner e-capability from its copy. Furthermore, S' will be able to generate e-capabilities for B in different classes, as it possesses the owner password. If this should not be the case, S will preventively transform the owner e-capability into a different class, thereby changing the password.

4 Discussion

4.1 Forging e-capabilities

Let us consider a malevolent subject S that holds no access permission for object B, and intends to forge a valid e-capability $E = (B, W, R)$ referencing B (we shall consider the class field later). To this aim, S generates a value for the R field that corresponds to the intended set of access rights, e.g. all 1's for an e-capability featuring all access rights. Password W will be chosen at random. Of course, if passwords are large and sparse, the probability to guess a valid password is vanishingly low, and the e-capability forging attempt is destined to fail.

Let us now suppose that subject S holds a valid e-capability $E = (B, W, R)$ for object B, and intends to forge a new e-capability $E' = (B, W', R')$ for B, where the set of access rights included in E' is *larger* than that in E. A result of this type implies that reduction field R is transformed into reduction field R' corresponding to more access rights; this an easy task, which can be accomplished by modifying of one or more subfields of R into flat subfields, for instance. However, password generation function h cannot be inverted. It follows that it is not possible to evaluate password W' corresponding to R' starting from password W corresponding to R. Once again, if passwords are large and sparse, a valid password cannot be guessed, and the access right amplification attempt will fail.

Let us now consider the case that subject S holds a valid e-capability $E = (B, W, R)$ for object B, and tries to take advantage of this e-capability to forge a new e-capability $E' = (B', W, R)$ for a different object B' of the same type as B. Let us suppose that S is aimed at using E' to access B' to execute operation *op*, in the hypothesis that the access rights corresponding to reduction field R make it possible to accomplish *op* successfully. In this hypothesis, the access right check in the first phase of the validation of E', as is delineated at step 2 of Sect. 2.2, will be successful. However, at step 3, password W' corresponding to R will be evaluated starting from the owner password of B' instead of the owner password of B that was used to generate W. Consequently, W' will not match W, and execution of *op* will fail.

Further e-capability forging attempts could be conceived, which involve e-capability classes. Let us consider a subject S that holds a valid e-capability $E = (B, W, R, C)$, let c denote the value of the class field C, and suppose that c corresponds to a form of revocation, i.e. one or more bits are cleared in entry RT_c of the revocation table. Suppose that subject S replaces value c with a new value c', corresponding to a class with a less stringent revocation or even no revocation at all (as is always the case for class 0). An action of this type would imply a password change; the new password W' should be the last password of the sequence $W_0 = h_{c'}(W_{own})$, $W_1 = h_{r_0}(W_0)$, $W_2 = h_{r_1}(W_1)$, $\ldots$etc., which involves quantity c' and all the reduction subfields that are not flat. This requires a knowledge of owner password W_{own}, which is only the case if subject S holds the owner e-capability.

4.2 E-capability revocation

Several solutions have been proposed in the past to the problem of access privilege revocation, with reference to classical capability and password-capability environments. A propagation graph can be associated with every given capability to keep track of all copies of this capability [6], [7]. This solution is contrary to a main requisite of the capability-passing model, i.e. simplicity in access privilege transmission between subjects. Capabilities can be short-lived, and in this case, a process that holds a given capability maintains the corresponding access privilege for a limited time interval [13]. This solution tends to overburden the protection system with explicit requests to extend the capability lifetime. A resource monitor can be associated with a protected object to mediate with the subjects that hold access privileges for this object, so that the object owner may ask for actions of selective revocation of access privileges [18], [25]. This approach complicates the structure of the protected objects, and is prone to affect the system performance adversely, owing to the need of mediated access.

As seen in Sect. 3, in our system, a subject that possesses the owner e-capability for a given object is in a position to review and revoke the access permissions granted by the existing e-capabilities for this object, completely or in part, a single exception being the owner e-capability itself, which cannot be revoked. To the aim of revocation, the subject modifies the contents of the revocation table, according to the e-capability classes to be actually revoked. Despite its simplicity, this e-capability revocation mechanism possesses a number of interesting properties. Revocation is [6]:

- *Partial.* A subject that possesses the owner e-capability for a given object can revoke any subset of the access rights for this object. To this aim, the subject accesses the entry of the revocation table corresponding to the intended class to clear the bits corresponding to the access rights to be revoked in this class.
- *Independent.* Different e-capabilities for the same object can be revoked independently of each other, if these e-capabilities belong to different classes.
- *Transitive.* The effects of the revocation of a given e-capability propagate to all the copies of this e-capability, transitively. In fact, a copy of an e-capability cannot be distinguished from the original, and e-capabilities have no memory of successive copy actions.
- *Temporal.* The effects of an e-capability revocation obtained by modifying an entry of the revocation table can be reversed simply by restoring the original value of this entry, through the same mechanism as for revocation.

4.3 E-capability reduction

In Sect. 2.3, we have seen that an e-capability can be transformed into a new e-capability for a reduced access privilege. This capability reduction process may well be iterated. In fact, we may have different reduction paths that start from the owner e-capability to generate different e-capabilities granting the same set of access rights; these e-capabilities are expressed in terms of different passwords.

Let us refer again to our example of the *File* type, and let us suppose that subject S_0 creates file F and consequently receives owner e-capability $E_{own} =$

$(F, W_{own}, 1111\ 1111\ 1111)$ (to simplify the presentation, here we ignore the class field). In E_{own}, all the reduction subfields are flat to indicate that the password is the owner password. Let us now suppose that S_0 wants to transfer all access rights for F except the *delete* access right to subject S_1. To this aim, S_0 transforms e-capability E_{own} into e-capability $E_1 = (F, W_1, 1111\ 1111\ 1110)$. In E_1, the value 1110 of reduction subfield r_0 indicates that access right *delete* is lacking. Password W_1 is given by relation $W_1 = h_{14}(W_{own})$, where h is the password generation function. Let us now suppose that in turn S_1 wants to transfer a single access right for F, the *execute* access right, to subject S_2. To this aim, S_1 reduces e-capability E_1 further, to obtain e-capability $E_2 = (F, W_2, 1111\ 1001\ 1110)$, where $W_2 = h_9(W_1)$. Now consider the case of a direct transmission of access right *execute* from S_0 to S_2. A single reduction is necessary, which starts from e-capability E_{own} to generate e-capability $E'_2 = (F, W'_2, 1111\ 1111\ 1000)$, where $W'_2 = h_8(W_{own})$. Thus, we have two e-capabilities, E_2 and E'_2, which correspond to the same access right *execute*. They originate from different reduction paths, and consequently, they are expressed in terms of different passwords.

A salient feature of our approach to e-capability reduction is that a subject S can carry out reduction autonomously. This is in sharp contrast with classical password capability environments. As seen in Sect. 1.3, in an environment of this type, the intervention an external entity is necessary, which we have called the password capability manager. It receives a password capability and returns a new password capability for the reduced set of access rights.

4.4 Considerations concerning performance

4.4.1 Execution times

Let $E = (B, W, R)$ be an e-capability for object B of type T, let n be the number of access rights defined by T, and let m be the number of subfields of reduction field R that are not flat, where $m \leq n - 1$. As seen in Sect. 2.2, validation of E requires m subsequent evaluations of password generation function h, starting from owner password W_{own}. The result will be compared with password W in E; if a match is found, E is valid, and it grants the access rights specified by R. Thus, the cost T_v in terms of execution time of an e-capability validation is a function of m and is given by relation $T_v = m \cdot T_h$, where T_h denotes the execution time of h. The maximum cost, corresponding to $m = n - 1$, is $(n - 1) \cdot T_h$. The cost is lower if one or more reduction subfields are flat, and the lowest limit corresponds to a flat R (i.e. all 1's), when $m = 0$, $W = W_{own}$, and $T_v = 0$.

As seen in Sect. 2, we may have different reduction fields that correspond to the same set of access rights. Consequently, we may have different password validation costs for e-capabilities expressing the same access permission. Consider, for instance, a situation of four access rights, ar_0 to ar_3, i.e. $n = 4$. In this case, the reduction field consists of three subfields. Examples of reduction fields are $R' = (1011\ 1101\ 1110)$ and $R'' = (1111\ 1111\ 1000)$, both corresponding to a single access right, ar_3. In R', no subfield is flat, and $m = 3$; it follows that validation of an e-capability featuring this reduction field implies the maximum cost in terms of execution times, $T_v = 3 \cdot T_h$. In R'', two subfields are flat, $m = 1$, and $T_v = T_h$.

4.4.2 Memory requirements

In a classical password capability environment, a set of passwords is associated with each object, and each password corresponds to an access permission defined in terms of a subset of all access rights. If a password is necessary that corresponds to a given access permission, and this password is not available, it is generated [1], [2]. Thus, the number of passwords associated with each given object tends to be high.

In contrast, in our e-capability environment, a *single* password, the owner password, is permanently stored in memory for each given object. Every other password is evaluated dynamically, when an e-capability is generated by reduction, or is validated. In fact, e-capability reduction processes that originate from the owner password can generate e-capabilities for all possible combinations of access rights.

As seen in Sect. 4.3, we may have two or more passwords that correspond to the same set of access rights. Situations of this type can occur, in particular, as a consequence of independent reduction activities, carried out by different subjects. No memory cost is associated with these multiple passwords, which in fact are not stored in memory permanently, but are evaluated dynamically, as part of actions of e-capability validation.

A final consideration is relevant to the memory requirements for capability storage. For 64-bit object identifiers and 128-bit passwords, the size of a classical password capability is 24 bytes. In an e-capability, for four access rights, the reduction field consists of three subfields of size 4 bits. A 4-bit class field makes it possible to define up to 16 different classes, which allow an accurate control over password review and revocation. In a configuration of this type, the size of an e-capability is 26 bytes. We may conclude that the memory size increase for storage of an e-capability with respect to a classical password capability is a negligible fraction of the total. Conversely, as seen above, we have noticeable memory space savings for password storage, and this is especially true if we are aimed at expressing several different access permissions. For instance, permanent storage of 15 passwords is necessary in a classical password capability system for a complete coverage of all possible combinations of four access rights; in contrast, a single password, i.e. the owner password, is sufficient in our e-capability environment.

5 Concluding remarks

We have considered an important protection problem, i.e. to identify the objects that each subject can access, and the operations that the subject can carry out on these objects. We have proposed a new model of password capability, called e-capability. In this model, an e-capability for a given object has the form of a password capability, i.e. an object name and a password, extended to contain a reduction field and a class field. The reduction field specifies the relation existing between the password and an owner password, generated when the object is created. This relation is expressed in terms of access rights. The class field is used to link each given password with a class, to the aim of the review and revocation of access permissions. The following is a summary of the main results we have obtained:

- A subject that holds a given e-capability is in a position to generate new e-capabilities for subsets of the access rights. This capability reduction process can be iterated to eliminate more access rights.
- A subject that holds an e-capability defined in terms of the owner password of a given object is in a position to revoke the access permissions granted by every other e-capability referencing this object, completely or in part. The e-capability revocation mechanism is based on e-capability classes. Revocation results to possess a number of interesting properties; it is partial, independent, transitive and temporal.
- If owner passwords are large, sparse and chosen at random, it is impossible for a malevolent subject to forge new e-capabilities. Any attempt to amplify the access permission granted by a given e-capability by adding new access rights, or to change the e-capability class, is destined to fail.
- The size of an e-capability is comparable to that of a traditional password capability.
- The number of passwords that need to be stored in memory permanently is kept to a minimum, and is equal to a single password, the owner password, for each object. Every other password is evaluated dynamically, as part of the actions involved in password validation and reduction.

Acknowledgements The author thanks the anonymous reviewers for their insightful comments and constructive suggestions.

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. M. Anderson, R. D. Pose, and C. S. Wallace. A password-capability system. *The Computer Journal*, 29(1):1–8, February 1986.
2. M. D. Castro, R. D. Pose, and C. Kopp. Password-capabilities and the Walnut kernel. *The Computer Journal*, 51(5):595–607, 2008.
3. J. S. Chase, H. M. Levy, E. D. Lazowska, and M. Baker-Harvey. Lightweight shared objects in a 64-bit operating system. *ACM SIGPLAN Notices*, 27(10):397–413, October 1992.
4. M. de Vivo, G. O. de Vivo, and L. Gonzalez. A brief essay on capabilities. *ACM SIGPLAN Notices*, 30(7):29–36, July 1995.
5. D. M. England. Capability concept mechanism and structure in System 250. In *Proceedings of the International Workshop on Protection in Operating Systems*, pages 63–82, IRIA, Paris, France, 1974.
6. V. D. Gligor. Review and revocation of access privileges distributed through capabilities. *IEEE Transactions on Software Engineering*, SE-5(6):575–586, November 1979.
7. D. A. Grove, T. C. Murray, C. A. Owen, C. J. North, J. A. Jones, M. R. Beaumont, and B. D. Hopkin. An overview of the Annex system. In *Proceedings of the Twenty-Third Annual Computer Security Applications Conference*, pages 341–352, Miami Beach, Florida, USA, December 2007. IEEE.
8. G. Heiser, K. Elphinstone, J. Vochteloo, S. Russell, and J. Liedtke. The Mungi single-address-space operating system. *Software – Practice and Experience*, 28(9):901–928, July 1998.
9. M. E. Houdek, F. G. Soltis, and R. L. Hoffman. IBM System/38 support for capability-based addressing. In *Proceedings of the 8th Annual Symposium on Computer Architecture*, pages 341–348, Minneapolis, Minnesota, USA, May 1981. IEEE Computer Society Press.
10. J. King-Lacroix and A. Martin. BottleCap: a credential manager for capability systems. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, pages 45–54, Raleigh, NC, USA, October 2012. ACM.

11. G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. seL4: formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pages 207–220, Big Sky, MT, USA, October 2009. ACM.
12. L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.
13. A. W. Leung and E. L. Miller. Scalable security for large, high performance storage systems. In *Proceedings of the Second ACM Workshop on Storage Security and Survivability*, pages 29–40, Alexandria, Virginia, USA, October 2006. ACM.
14. H. M. Levy. *Capability-Based Computer Systems*. Digital Press, Bedford, Mass., USA, 1984.
15. L. Lopriore. Encrypted pointers in protection system design. *The Computer Journal*, 55(4):497–507, April 2012.
16. L. Lopriore. Password capabilities revisited. *The Computer Journal*, 58(4):782–791, April 2015.
17. R. C. Merkle. One way hash functions and DES. In *Proceedings of the 9th Annual International Cryptology Conference – Advances in Cryptology*, pages 428–446, Santa Barbara, California, USA, August 1989. Springer.
18. M. S. Miller, K.-P. Yee, and J. Shapiro. Capability myths demolished. Technical report, Systems Research Laboratory, Johns Hopkins University. http://srl.cs.jhu.edu/pubs/SRL2003-02.pdf, 2003.
19. P. G. Neumann and R. J. Feiertag. PSOS revisited. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 208–216, Las Vegas, NV, USA, December 2003. IEEE.
20. R. Pose. Password-capabilities: their evolution from the Password-Capability System into Walnut and beyond. In *Proceedings of the Sixth Australasian Computer Systems Architecture Conference*, pages 105–113, Gold Coast, Australia, January 2001. IEEE.
21. B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: a synthetic approach. In *Proceedings of the 13th Annual International Cryptology Conference*, pages 368–378, Santa Barbara, California, USA, August 1993. Springer.
22. P. Samarati and S. De Capitani Di Vimercati. Access control: policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, pages 137–196. Springer, Berlin, Heidelberg, 2001.
23. R. S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, 1988.
24. L. Seitz, J.-M. Pierson, and L. Brunie. Key management for encrypted data storage in distributed systems. In *Proceedings of the Second IEEE International Security in Storage Workshop*, pages 20–30, Washington, DC, USA, October 2003. IEEE.
25. J. S. Shapiro, J. M. Smith, and D. J. Farber. EROS: a fast capability system. *ACM SIGOPS Operating Systems Review*, 34(2):170–185, 2000.
26. W. Trappe, J. Song, R. Poovendran, and K. J. Liu. Key management and distribution for secure multimedia multicast. *IEEE Transactions on Multimedia*, 5(4):544–557, 2003.
27. M. V. Wilkes and R. M. Needham. *The Cambridge CAP Computer and its Operating System*. North-Holland, New York, 1979.