Access control lists in password capability environments

Lanfranco Lopriore

Dipartimento di Ingegneria dell'Informazione, Università di Pisa, via G. Caruso 16, 56126 Pisa, Italy. E-mail: l.lopriore@iet.unipi.it

Abstract — With reference to a protection system featuring active subjects that attempt to access passive, typed objects, we propose a set of mechanisms supporting the distribution, verification, review and revocation of access privileges. In our approach, a protection domain is a collection of access rights for the protected objects. An access control list is associated with each object to specify the access rights in each domain. Objects are grouped into clusters. To access the objects in a given cluster, a subject presents a gate referencing this cluster. The gate is a form of password capability that identifies one or more domains. The gate grants the access rights specified for these domains by the access control lists of the objects in the cluster. A subject that holds a gate and is aimed at distributing the access privileges in this gate in restricted form can reduce the gate to eliminate domains; the gate reduction procedure requires no intervention of the protection system. A small set of protection primitives allows subjects to manage objects and access control lists. Forms of revocation of access permissions are supported, at both levels of gates and access control lists.

Keywords: access control list; access right; domain; one-way function; password capability; protection.

1 INTRODUCTION

In a classic protection system paradigm, a set of active *subjects* (users, processes) attempts to access a set of passive, typed entities called *objects* [21], [38]. The type of a given object states the set of the *operations* that can be executed on this object, and the *access rights* that are necessary to accomplish each operation successfully. A subject aimed at executing an operation on a given object must possess the access rights required by this operation, as is stated by the object type. A *protection domain* is a collection of access rights for the protected objects. At any given time, each subject is executed in a protection domain, and can take advantage of the access rights included in this domain to operate on the protected objects. In the course of execution, the subject can change domain, according to the access right requirements of the different execution phases.

1.1 Capabilities and access control lists

Let $b_0, b_1, \ldots$ be a set of objects, and $d_0, d_1, \ldots$ be a set of protection domain. In a well-known representation, the state of the protection system takes the form of a matrix, called the *access matrix*, featuring a row for each domain and a column for each object [16], [28]. Element $AM_{i,j}$ of the access matrix, corresponding to row i and column j, contains the specification of the access rights included in domain d_i for object b_j. An important aspect is that a domain can also be a protected object, and in this case the domain corresponds to both a row and a column of the access matrix.

The access matrix tends to be large and sparse. Most elements of the access matrix are likely to be empty, so storage in matrix form is usually inadequate. Two alternative, well-known approaches to represent the access matrix are *capability lists* and *access control lists* [30]. In the capability list approach, the access matrix is stored by rows. A capability list is a collection of capabilities, which is associated with each domain, and specifies the access rights included in this domain. A *capability* is a pair (b, ar), where ar specifies a set of access rights for object b [18]. In the access control list approach, the access matrix is stored by columns. An access control list ACL_b is associated with each given object b. Each element of ACL_b has the form (d, ar), and specifies the set ar of access rights for b that is included in domain d.

Capabilities need to be segregated into protected memory regions [5], [19], [37]. This is necessary to prevent a subject that holds a given capability from modifying this capability, e.g. the access right field, to obtain an undue amplification of access rights, or the object identifier field, to forge a capability referencing a different object. In a segmented memory environment, the capability segregation problem can be solved by reserving *ad hoc* memory segments for capability storage, the *capability segments* (in contrast, the *data segments* will be reserved for storage of ordinary information items) [6], [13]. Each capability segment can contain a capability list. Capability segments can only be accessed in a strictly controlled fashion, by executing special machine instructions, the *capability instructions*. In an alternative approach, in a system featuring a form of tagged memory, a tag can be associated with each memory cell to specify whether this cell contains a capability or an ordinary information item [2], [14], [24], [36]. A cell tagged to contain a capability can be accessed only by using the capability instructions.

Password capabilities are an alternative, effective solution to the capability segregation problem [4], [12], [20], [25]. In a password capability environment, one or more passwords are associated with each protected object. Each password corresponds to a set of access rights for this object. A password capability is a pair (b, w), where b identifies an object, and w is a password. If w matches one of the passwords associated with object b, the password capability grants the corresponding access rights on b. If passwords are large and sparse, the probability that a malevolent subject guesses a valid password to forge a password capability is vanishingly low [3]. It follows that password capabilities can be mixed in memory with ordinary information items, and can be manipulated by using standard machine instructions.

In this paper, we propose the organization of a protection system that takes advantage of both access control lists, for the protection of ordinary objects, and a form of password capability, called *gate*, for the protection of domains.

1.2 Clusters

We group objects into *clusters*. For each object, the cluster includes the corresponding access control list. To access the objects in a given cluster, a subject must demonstrate the right to take advantage of all, or part of, the domains specified by the access control lists in that cluster. This is a problem of certified identity whose solution is the main contribute of this paper, and is based on gates. A gate referencing a given cluster identifies one or more domains in this cluster. A subject that possesses the gate can access the objects in the cluster with the access rights specified for these domains by the access control lists of these objects.

The rest of this paper is organized as follows. Section 2 introduces the gate concept with special reference to the relation existing between a gate for a given cluster and a *base gate* generated when the cluster is created. Gate validation and reduction, to eliminate access permissions, are analyzed in special depth. Section 3 presents a small set of primitives, the *protection primitives*, which allows subjects to access the objects and to manage their access control lists. Section 4 considers the problems related to the review of access permissions, with special reference to gate revocation. Section 5 discusses the motivations of the gate paradigm in reference to a number of important viewpoints, which include fraudulent gate forging, gate equivalency, and the grouping of objects into clusters. A few considerations concerning performance are presented, in both terms of the memory requirements for gate storage and the execution times for gate validation. Section 6 discusses the relation of our work to previous work. Section 7 gives concluding remarks.

2 GATES

Let C be a cluster, and let $b_0, b_1, \ldots$ be the protected objects included in C. The cluster implements an object protection technique based on domains and access control lists. Let d_0, d_1, $\ldots$, d_{n-1} be the protection domains for $b_0, b_1, \ldots$. An access control list ACL_b is associated with each given object b. Each element of ACL_b has the form (d, ar), where ar denotes the set of access rights for b that is included in domain d.

A subject S, aimed at accessing cluster C to execute operation op on object b, must present a gate referencing C. This gate gives the right to take advantage of one or more domains in C. The access control list ACL_b of b specifies a set of access rights for each of these domains. The access will be accomplished successfully only if the *union* of these sets of access rights includes the access rights that are required to execute *op*.

The gate concept is a variant of the classical password capability concept. A gate G for cluster C has the form $G = (W, R)$. Quantity W is a password; as will be made clear shortly, this password univocally identifies the cluster. Quantity R, called the *domain selector*, identifies one or more domains in C, as follows. R is partitioned into $n - 1$ subfields, called *primary selectors* and denoted by r. Thus, $R = (r_{n-2}, r_{n-3}, \ldots, r_0)$. The size of a primary selector is n bits, one bit for each domain (the least significant bit, bit 0, corresponds to the first domain, d_0). For each bit that is asserted in a primary selector, the corresponding domain is eliminated from the gate. It follows that the domains referenced by G are those corresponding to cleared bits in quantity $r_{n-2} \vee r_{n-3} \vee \ldots \vee r_0$. A primary selector whose value is 0 is called *null.* All null primary selectors are placed in the most significant positions of domain selector R, at the highest order numbers. In a given gate, if all primary selectors are null, $R = 0$ and the gate references all the domains in the corresponding cluster. We wish to point out that $n - 1$ primary selectors allow us to specify any combination of active domains, even if each non-null primary selector has a single bit asserted.

Figure 1 shows two different configurations of domain selector R of gate $G = (W, R)$. In both cases, $n = 4$, that is, cluster C referenced by G includes four domains, and consequently R is formed by three primary selectors. Thus we have $R = (r_2, r_1, r_0)$. The size of each primary selector is four bits. In the configuration of Figure 1a, bits 0 and 2 of primary selector r_0 are asserted to indicate that domains d_0 and d_2 are laking from G. This is similar to the meaning of primary selector r_1 for domain d_1. The remaining primary selector r_2 is null, so it eliminates no domain. We may conclude that gate G references a single domain, d_3. In the configuration of Figure 1b, bits 0, 1 and 2 of primary selector r_0 are asserted to indicate that domains d_0, d_1 and d_2 are excluded from gate G. The remaining primary selectors r_1 and r_2 are null. In this case, too, the resulting gate G includes a single domain, d_3. However, as will be illustrated shortly, the two cases correspond to different passwords. In fact, in the given cluster we may have several distinct passwords for the same set of domains.

2.1 Password derivation

A password, called the *base password*, is associated with each given cluster C when the cluster is created. This password is denoted by BW_C. A gate expressed in terms of BW_C is called the *base gate* and is denoted by BG_C. In the base gate, the domain selector is always null, thus we have $BG_C = (BW_C, 0)$. The base gate references all the domains in cluster C; possession of the base gate is equivalent to possession of the access privileges in all these domains.

In gate $G = (W, R)$, if one or more primary selectors are non-null, $R \neq 0$ and the non-null primary selectors identify a password generation path that proceeds from BW_C to produce password W. This password generation path is constructed by using a parametric one-way function.

Function f is *one-way* if given a value w it is easy to compute $f(w)$, and given a value y it is computationally unfeasible to determine w such that $y = f(w)$ [1], [15]. Function $g_r(w)$ is a *parametric one-way function* if given a value y and a parameter r, it is computationally unfeasible to determine a value w such that $y = g_r(w)$ [34]. Thus, a parametric one-way function corresponds to a family of one-way functions, a one-way function for each value of the parameter [29].

In our protection system, we take advantage of a parametric one-way function $g_r(w)$, called the *generation function*, where argument w is a password and parameter r is a primary selector. In gate $G = (W, R)$, if $R \neq 0$ then password W is produced by a password generation path that starts from base password BW_C and uses g iteratively. Let $R = (r_{n-2}, r_{n-3}, \ldots, r_0)$ be the composition of the domain selector. We have $W_{j+1} = g_{r_j}(W_j)$, $j = 0, 1, \ldots, n-2$, where $W_0 = BW_C$ and $W = W_{n-1}$. If one or more primary selectors are null, the iterations terminate at the primary selector, say r_k, that precedes the first null primary selector, and in this case $W = W_{k+1}$.

Figure 2 shows the evaluation of password W for a specific configuration of domain selector R of gate $G = (W, R)$. In this example, $n = 4$, that is, cluster C referenced by G includes four domains, and consequently R is formed by three primary selectors. Thus we have $R = (r_2, r_1, r_0)$. The size of each primary selector is four bits. If $R = (0000\ 0010\ 0101)$, bit 3 is cleared in all the primary selectors. Thus, possession of gate G allows us to take advantage of the access rights in domain d_3. In the evaluation of password W, at the first step we have $W_0 = BW_C$. Primary selector r_0 is 0101 (5 in decimal notation), thus we have $W_1 = g_5(W_0)$. In the second step, primary selector r_1 is 0010, thus we have $W_2 = g_2(W_1)$. Primary selector r_2 is null; this terminates the iterations, and $W = W_2$.

2.2 Gate validation

Let S be a subject that possesses gate $G = (W, R)$. The gate is valid and it references cluster C if password W is derived from base password BW_C of C, as has been illustrated in Section 2.1. Subject S can take advantage of G to execute operation *op* on object b if the domains of C specified by G (as indicated by domain selector R) collectively include the access rights required by *op*.

In more detail, let $R = (r_{n-2}, r_{n-3}, \ldots, r_0)$, where each r denotes a primary selector.

If $W = BW_C$ and $R = 0$ (that is, all primary selectors are null), then $G = BG_C$, and it specifies all the domains of cluster C. If $R \neq 0$, validation of G proceeds as follows. Parametric generation function g and the values of the primary selectors are used to evaluate password W. We have $W_0 = BW_C$ and $W_{j+1} = g_{r_j}(W_j), j = 0, 1, \ldots$ etc. The sequence terminates at the primary selector, say r_k, that precedes the first null primary selector. We define $W' = W_{k+1} = g_{r_k}(W_k)$. If no primary selector is null, the sequence terminates at the last primary selector r_{n-2}, and in this case $W' = W_{n-1}$. Gate G is valid if $W = W'$. If G is valid, we evaluate quantity $r_{n-2} \vee r_{n-3} \vee \ldots \vee r_0$. The bits of the result that are cleared specify the domains of cluster C that are actually referenced by G.

Let ACL_b denote the access control list of object b. Let ar_{op} denote the set of access rights which is required to execute operation *op*, as is specified by the type of object b, and let ar_G denote the union of the access rights specified by the entries of ACL_b corresponding to the domains referenced by gate G. Operation *op* can be executed successfully on b if $ar_{op} \subseteq ar_G$. If this is not the case, execution raises an exception of violated protection, and *op* terminates with failure.

We wish to remark that our protection system is an extension of the classical access control list paradigm. Suppose that subject S possesses a gate G referencing cluster C. When S presents G, if G specifies a single domain, then the subject enters this domain, as is the case in a traditional access control list environment. If G specifies two or more domains, then the subject enters a *virtual* domain whose access rights are the *union* of the access rights in the component domains. The gate mechanism allows a subject to change the cluster (and consequently, the virtual domain) dynamically in the course of execution. A subject that holds gates for different clusters can determine the present virtual domain by simply presenting a gate for the corresponding cluster. This is an efficient implementation of the concept of a domain switch in direct subject control.

2.3 Gate reduction

A subject S_1 that holds a gate G referencing domains in a given cluster can transfer a copy of this gate to another subject S_2. Consequently, S_2 acquires all access permissions granted by G. Subject S_1 can also transform G into a new gate G' for the same cluster and a subset of the domains. This process is called *gate reduction*, and can be especially important to limit the access permissions of the recipient subject S_2.

Let $G = (W, R)$ be a gate defined in terms of password W and domain selector R, let $R = (r_{n-2}, r_{n-3}, \ldots, r_0)$ be the composition of R in terms of its primary selectors, and let D denote the set of the domains referenced by G, as is specified by the bits that are cleared in quantity $r_{n-2} \vee r_{n-3} \vee \ldots \vee r_0$. Subject S that holds G can reduce this gate into a gate $G' = (W', R')$ referencing only part of the domains in D. To this aim, let r_k

be the first primary selector that is null in R. We transform R into R' by setting the bits of r_k that correspond to the domains to be eliminated. Afterwards, we generate the new password W' by using parametric generation function g. We have $W' = g_{r_k}(W)$.

Figure 3 shows the reduction of gate $G = (W, R)$ in a specific configuration of domain selector R. In this example, $n = 4$, that is, cluster C referenced by G includes four domains. Consequently, R is formed by three primary selectors, $R = (r_2, r_1, r_0)$, and the size of each primary selector is four bits. If $R =$ (0000 0000 0011), bits 2 and 3 are cleared in all primary selectors. This means that gate G references domains d_2 and d_3. Gate G can be reduced to reference a single domain, say domain d_3. To this aim, we modify the least significant null primary selector, r_1, by setting bit 2 to indicate that domain d_2 is no longer referenced by the gate; the resulting configuration will be $R' =$ (0000 0100 0011). The new password W' will be obtained by applying parametric generation function g using primary selector r_1 as a parameter. In decimal notation, the value of this primary selector is 4, so we have $W' = g_4(W)$.

3 THE PROTECTION PRIMITIVES

The protection system defines a set of primitives, the *protection primitives*, aimed at the management of objects and access control lists (Table 1). This section illustrates the effects of the execution of each of these primitives from the point of view of the subject that issues the primitive. We shall hypothesize that all object types include access rights *own* and *copy*. Access right *own* for a given object makes it possible to delete the object; this access right may also grant other type-specific prerogatives. Access right *copy* makes it possible to create an object copy. Let ACL_b denote the access control list of object b. To simplify the presentation, we shall say that a given gate G includes access right ar for b if at least one element of ACL_b includes ar, and G specifies the domain d of this element.

3.1 Allocating and deleting objects

A cluster is itself an object. In every given cluster, domain d_0 is called the *owner domain* and includes the *own* access right for this cluster. A subject that possesses a gate encompassing domain d_0 is called the *cluster owner*; it can take advantage of the *own* access right to allocate new objects in the cluster, and to delete the cluster. (The cluster owner is also allowed to change the base password; this issue will be considered in subsequent Section 4.)

Object allocation is made possible by the $b \leftarrow newObject(G, d)$ protection primitive. Let C be the cluster referenced by gate G. Execution of this primitive allocates a new object b in C. An access control list ACL_b is created for b. ACL_b contains a single element having the form (d, ar_{T_b}), where ar_{T_b} denotes full access rights for object b, as stated by the type T_b of b. Execution terminates successfully only if gate G references both domain d and the owner domain d_0 (that is, the subject that executes this primitive must be the owner of cluster C).

Object deletion is supported by the $deleteObject(G, b)$ primitive. Execution of this primitive deletes object b from the cluster referenced by gate G. Execution terminates successfully only if G includes access right *own* for b (that is, G specifies the domain of an element of ACL_b that includes access right *own*).

3.2 Accessing an object, and creating an object copy

Let T_b be the type of object b of cluster C, and let *op* be the generic operation defined by T_b. Execution of *op* on b is made possible by protection primitive $res \leftarrow operation(G, b, op)$. Argument G is a gate referencing C. Execution of this primitive returns the result *res* of *op*. Let *ar* denote the set of access rights required by *op*. Execution terminates successfully only if G includes access rights *ar* for b.

Creation of a copy of a given object in cluster C is made possible by protection primitive $b' \leftarrow copyObject(G, b)$. Argument G is a gate referencing C. Execution of this primitive allocates a new object b' in C; the value of b' is equal to the value of b. Execution terminates successfully only if G includes access right *copy* for b.

3.3 Adding and revoking access rights

Access control list modifications are made possible by two protection primitives called *addAr* and *removeAr*. Let G be a gate referencing cluster C, let b be an object in this cluster, and let ACL_b be its access control list. Primitive $addAr(G, d, b, ar)$ accesses the element of ACL_b reserved for domain d to add the access right specified by argument *ar*. Execution of this primitive terminates successfully only if gate G includes access right *ar* for b. This means that a subject can grant an access right for a given object only if it possesses a gate including this access right.

Primitive $removeAr(G, d, b, ar)$ accesses the element of ACL_b reserved for domain d to eliminate the access right specified by argument *ar*. Execution of this primitive terminates successfully only if gate G includes access right *own* for b. This means that the owner of a given object is the only subject allowed to remove access rights for this object.

4 GATE REVOCATION

The gate revocation problem consists of giving the owner of a given cluster the ability to review and revoke the gates referencing this cluster. A requirement is that the effects of revocation should extend system-wide.

Our system supports gate revocation through the base password and domain naming. We take advantage of the fact that a cluster is itself an object. As seen in Section 3.1, in each cluster, domain d_0 includes the *own* access right for the cluster. The cluster owner, which possesses a gate encompassing domain d_0, can take advantage of the *own* access right to change the base password of the cluster.

Let $G = (W, R)$ be a gate referencing cluster C, let $R = (r_{n-2}, r_{n-3}, \ldots, r_0)$ be the composition of the domain selector in terms of its primary selectors, and let BW_C be the base password of C. As seen in Section 2.2, when a subject presents G to access C, the validity of the gate is verified by an iterative procedure, which starts from BW_C to determine a value W'. The iterations use parametric generation function g and involve the primary selectors. Gate G is valid if W' matches password W specified by G. Now suppose that the cluster owner changes base password BW_C, and let BW'_C be the new base password. So doing, the cluster owner revokes the validity of base gate $BG_C = (BW_C, 0)$ and of all the gates derived from BG_C by reduction. This means that it will no longer possible to use these gates to reference the domains of C. The cluster owner possesses the new base gate, $BG'_C = (BW'_C, 0)$, and can carry out a new distribution of access rights in the form of gates derived from BG'_C by reduction.

It is even possible to associate multiple base passwords with the same given cluster. Let $BW_{C,0}$ and $BW_{C,1}$ be two base passwords for cluster C, for instance. The corresponding base gates are $BG_{C,0} = (BW_{C,0}, 0)$ and $BG_{C,1} = (BW_{C,1}, 0)$. The cluster owner is in the position to use these base gates to generate new gates by reduction. Let G be a gate generated in this way. When G is validated, the result W' of the iterative validation procedure of Section 2.2 is compared with both $BW_{C,0}$ and $BW_{C,1}$. If a match is found, G is valid. In a situation of this type, the cluster owner can revoke the gates for the cluster selectively, simply by eliminating a base password. For instance, by eliminating $BW_{C,1}$, the cluster owner revokes the validity of base gate $BG_{C,1}$ and of all the gates derived from this base gate by reduction. Of course, validity of $BG_{C,0}$ and its derived gates is not affected by the revocation. These considerations can be extended to an arbitrary number of base gates.

The gate revocation procedure, illustrated above, is valid at gate level and applies to all domains. It is even possible to restrict revocation to a single domain, as follows. In cluster C, we associate two different names, say d' and d'', with a given domain d. By doing so, we can have different gates referencing d, which use different domain names. The cluster owner can eliminate a domain name, e.g. d''. An action of this type produces a selective gate revocation involving all the gates for cluster C that reference domain d using name d''.

5 DISCUSSION

5.1 Forging gates

The security of a cluster depends on the infeasibility of guessing any gate for this cluster. Let us suppose that subject S is fraudulently aimed at forging the base gate of cluster C from scratch. This base gate has the form $BG_C = (BW_C, 0)$, where BW_C is the base password of C, and the domain selector is set to 0 to reference all domains. Subject S does not hold BW_C, and consequently, it will use a casual password. If base passwords are large, sparse, and chosen at random, the probability of a casual match is vanishingly low, and the gate forging attempt is destined to fail. Similar considerations can be made for a gate that is not the base gate.

The size of the base password will be determined by taking the overall security requirements into account. The use of large passwords, e.g. 128 bits, induces a very low probability that a valid password be guessed by a force brute attack. A significant improvement of the password scheme would be the introduction of some form of penalty to the guessing subject, so that the cost of any form of systematic attack becomes prohibitive [35]. In our system, we can implement an extension of this type by taking advantage of the fact that object accesses are mediated by clusters. In a possible approach, suppose that a subject presents a gate to a given cluster to access an object in this cluster, e.g. by executing the *operation* protection primitive. If the access fails owing to an invalid password, the subject incurs a small penalty [3], e.g. a delay, whose extent is increased at each subsequent access attempt. In this way, a casual error, e.g. a cluster access using a gate expressed in terms of a revoked password, produces insignificant consequences, but the cost of a systematic, deliberately harmful attack becomes prohibitive.

If the base password of a cluster is derived from some implicit or explicit cluster property, e.g. the cluster composition, then the system can be more easily violated [3]. This is a reason to generate base passwords at random, by using a pseudo-random number generator, for instance. Of course, a significant enhancement would be a hardware device producing truly random bitstrings [33].

Let us now consider a subject S that possesses a gate $G = (W, R)$ referencing a set of domains in cluster C. Suppose that S is aimed at transforming G into a new, stronger gate $G' = (W', R')$. To this aim, S modifies domain selector R to form the new domain selector R' referencing more domains. A result of this type can be simply obtained by clearing the most significant primary selector that is non-null in R. The next step is to derive password W' from password W. In fact, W' *precedes* W in the iterative generation procedure, illustrated in Section 2.1, which starts from base password BW_C of cluster C to produce W. But parametric generation function $g_r(w)$, used to generate W from W', is one-way. This means that it is computationally unfeasible to invert $g_r(w)$ to obtain W' from W. Consequently, subject S will have to use a random W', and the probability of success is virtually null.

We assume that $g_r(w)$ is one-way for each value of parameter r. Furthermore, let $g_{r_1}(w)$ and $g_{r_2}(w)$ be one-way functions derived from $g_r(w)$ for different values of the parameter; we require that there should be no tractable method to compute $g_{r_2}(w)$ given $g_{r_1}(w)$. We can construct $g_r(w)$ starting from a block cipher to minimize the design and implementation efforts. A practical example is $g_r(w) = E_w(r)$, i.e. the encryption of r using symmetric cipher E with key w [34]. The block cipher must be adapted to guarantee that the resulting function is not invertible. Only block ciphers that are secure against known plaintext attacks are appropriate, e.g. DES [22], [26], [29].

5.2 Equivalent gates

Let C be a cluster, let $G = (W, R)$ be a gate, and let $R = (r_{n-2}, r_{n-3}, \ldots, r_0)$ be the composition of the domain selector of this gate in terms of its primary selectors. As seen in Section 2, the set of domains in C that are referenced by G is specified by the bits that are cleared in quantity $r_{n-2} \vee r_{n-3} \vee \ldots \vee r_0$. It follows that different configurations of the primary selectors may well correspond to the same set of domains. The corresponding gates are said to be *equivalent*. For instance, let us consider gates $G_1 = (W_1, R_1)$ and $G_2 = (W_2, R_2)$, where R_2 is obtained by changing the order of the non-null primary selectors in R_1. In this case, G_1 and G_2 are equivalent, but the iterative password generation procedure will produce different results, so $W_1 \neq W_2$.

It is important to note that no additional cost follows from the point of view of the memory requirements for password storage. In fact, a single password needs to be stored for each given cluster, the base password. Password validation is not based on comparison with a pre-existing set of passwords, as is the case in password capability environments [12], [20], [25]. Instead, as seen in Section 2.2, password validation is the result of a dynamic evaluation procedure, which starts at the base password and terminates at the password to be validated.

5.3 Object clusters

In our protection model, objects are grouped into clusters. The cluster concept is flexible, and well suited to be adapted to a variety of protection problems. We shall now consider a few significant examples of these problems.

5.3.1 Linear hierarchy

Let us refer to a protection system that organizes documents into security classes. The classes are structured into a linear hierarchy, whereby each class can have only one parent and one child in the hierarchy [11]. Each subject is assigned to a range of adjacent classes. The system implements an extended form of mandatory access control [30], [32], in which a subject in a given class range may write to documents in each class in this range and above, and may read documents in each class in this range and below. For instance, for n security classes numbered from 1 (the highest level) to n, a subject in class range i to j can write documents in classes 1 to j, and can read documents in classes i to n (thus, the subject can access the documents in classes i to j both to read and to write).

The *Document* data type defines access rights *read*, which makes it possible to access the given document to read its contents, and *write*, which makes it possible to modify these contents. All documents are grouped into a cluster featuring n domains, one domain for each security class. An access control list is associated with each document. The access control list of a document in class c specifies access right *read* for domains 1 to c, and access right *write* for domains c to n. The cluster owner holds a gate expressed in terms of the base gate of the cluster, which specifies all domains. Suppose that a subject S should be inserted into class range i to j. The cluster owner will use the generation function to derive a gate from the base gate. This new gate will specify the domains from i to j, and will be granted to S.

5.3.2 Tree shaped hierarchy

Let us now refer to a tree shaped hierarchy [11] in which documents are placed at the lowest hierarchical level, and are grouped into subclasses according to contents. In turn, subclasses are grouped into classes, and the set of all classes forms the protection system. A subject in a given subclass can access all the documents in this subclass; a subject in a given class can access the documents in all the subclasses of this class; and finally, a subject at the root of the hierarchy can access all documents.

In our protection environment, we associate a domain with each subclass. The access control list of a document in the i-th subclass specifies both access rights *read* and *write* for the i-th domain. A subject in a given subclass holds a gate specifying a single domain, i.e. the domain of this subclass. A subject in a given class holds a gate specifying the domain of each subclass of this class. Finally, a subject in the root holds the base gate, which specifies all domains.

5.3.3 Network

As a final example, let us consider a distributed environment consisting of a network of nodes grouped into applications. The nodes in the same application cooperate in the same task. In each application, a node assumes the role of the application server, which collects data from all the other nodes of this application. The application server transforms these data into a form suitable for transmission to the base station, which is responsible for the final presentation and delivery of the results of the elaborations of the entire network to the external environment.

The primary memory of each node hosts a segment, which is shared with the other nodes of the same application, and is reserved for communications with these nodes. Two operations are possible on a shared segment, to read its contents and to modify these contents. These operations are made possible by access rights *read* and *write*, respectively. The segments shared by the nodes of a given application form a cluster. Each application defines a set of protection domains. An access control list is associated with each segment, which specifies the access rights included for that segment in each domain of the cluster. Each node holds a gate that allows it to access a subset of the segments in the same application, as is necessary for internode cooperation. The application server holds the base gate, which allows it to access the shared segments of all the application nodes.

In turn, the shared segments of all application servers are grouped to form a cluster. Each application server holds a gate that allows it to access a subset of the shared segments of the other application servers, as is required for application servers to communicate. The base station possesses the base gate that allows it to access the shared segment of all application servers, as is necessary to communicate with the servers.

5.4 Considerations concerning performance

5.4.1 Storage requirements

The memory requirements for storage of a gate are the result of two components, the size of the password and the size of the domain selector. The size of the password is determined by the overall security requirements, e.g. 128 bits. The size of the domain selector is a function of the number of the domains in the corresponding cluster. In fact, as seen in Section 2, for n domains we have $n-1$ primary selectors, and the size of each primary selector is n bits.

In a possible, effective implementation of a protection system using gates, we shall define a limited number of gate formats. We may have a *short gate* supporting up to four domains ($n = 4$). In this case, the domain selector consists of three primary selectors of four bits each, which fit into two bytes. Then we have a *standard gate* for up to eight domains ($n = 8$). Here, the domain selector consists of seven primary selectors of eight bits each, for a total of seven bytes. Finally, a *long gate* can support up to 16 domains ($n = 16$). In this case, we have 15 primary selectors of size 16 bits, for a total of 30 bytes. If the size of a password is 128 bits, the memory requirements for storage of a gate are 18 bytes for a short gate, 23 bytes for a standard gate, and 46 bytes for a long gate. (In contrast, for 64-bit object identifiers and 128-bit passwords, the size of a classical password capability is 24 bytes.)

As seen in Section 2, the primary selectors are aimed at specifying domain reductions. If each primary selector eliminates a single domain, we need $n-1$ primary selectors to specify all possible reductions. A gate is often reduced before being transmitted to another subject, to limit the access privileges of the recipient. In fact, it is rarely the case that a gate traverses a number of transmission steps, with reduction, as high as is made possible, for instance, by the long gate format. This suggests us to limit the number of possible reductions to save storage. With a limit of eight reductions, the long gate format features eight primary selectors of size 16 bits, and overall the size of the domain selector is 16 bytes. A subject that has to apply a further reduction when no null primary selector is available will ask the cluster owner to *shrink* the gate, that is, to produce an equivalent gate with a single non-null primary selector. In this primary selector, the cluster owner will set each bit that is asserted in at least one primary selector in the original gate. The password of the equivalent gate will be recalculated starting from the base password, as usual.

5.4.2 Execution times

As seen in Section 2.2, validation of a given gate is an iterative procedure that implies one execution of the parametric generation function g for each non-null primary selector in that gate. Let T_g denote the time necessary for a single iteration, including the execution time of g, and let m be the number of non-null primary selectors in gate G, where $m \leq n-1$, n is the number of the domains, and $n-1$ is the number of the primary selectors. Let T_v denotes the total time necessary for validation of a gate; we have $T_v = m \cdot T_g$. The maximum time cost corresponds to the case of no null component, when $m = n-1$ and $T_{v,max} = (n-1) \cdot T_g$. The time cost is lower for one or more null components, and the minimum cost corresponds to the validation of the base gate, when $m = 0$ and $T_{v,min} = 0$.

As seen in Section 5.4.1, it is always possible to shrink a gate featuring two or more non-null primary selectors into an equivalent gate featuring a single non-null primary selector. A positive side effect is a reduction of the gate validation time. After shrinking the gate, we have $m = 1$ and $T_v = T_g$.

6 RELATION TO PREVIOUS WORK

From an operational point of view, capability lists make it easy to determine the access rights that form a given domain; however, in a capability environment, it is hard if not impossible to determine the domains that include access rights for a given object (an action of this type requires the inspection of all capability lists). Conversely, access control lists facilitate the identification of the access rights associated with a given object for each domain; however, in an access control list environment, it is difficult to determine the access rights included in a given domain (an action of this type requires the inspection of all access control lists).

Our research effort has been aimed at taking advantage of the positive properties of the two mechanisms, capabilities and access control lists, in a hybrid approach designed to minimize the negative effects. We support object protection by reserving an access control list for each given object. The list specifies the access rights included in each domain for this object. The classical password capability paradigm associates passwords with objects; in sharp contrast, we associate passwords with domains. In fact, the gate is a form of password capability extended to contain a domain selector that identifies one or more domains within the same cluster.

We are now in the position to compare the three protection models, capability lists, access control lists, and our hybrid model based on gates. We shall consider a number of important viewpoints that include *segregation*, i.e. the method to prevent access privilege forging; *reduction*, i.e. the ability to reduce the extent of a given access privilege by eliminating access rights; *revocation*, i.e. the ability to prevent further utilization of a given access privilege; and *password proliferation*, i.e. the number of passwords that should be associated with the given object.

6.1 Segregation

In the original formulation, a capability has the form (b, ar), where ar specifies a set of access rights for object b. In a possible implementation, ar features one bit for each access right; if the given bit is set, the corresponding access right is included in the capability. Capabilities should be segregated in memory. This is necessary to prevent a subject that holds a given capability from modifying the ar field to obtain an undue amplification of access rights, or even the b field to forge a new capability referencing a different object. As seen in Section 1, capability segregation is always supported by *ad hoc* mechanisms, e.g. capability segments [6], [13] or a tagged memory [2], [14], [24], [36].

In a password capability system, no form of segregation is necessary, provided that passwords are large, sparse and chosen at random [8], [12], [23]. Being a form of password capability, gates are inherently protected from tampering. Furthermore, as seen in Section 5.1, the non-invertibility property of the parametric generation function guarantees that it is impossible to manipulate a gate to forge a new gate with stronger access rights.

6.2 Reduction

Ease of access right distribution is certainly one of the main advantages of all capability models. A subject that holds a given capability is free to transfer the access rights granted by this capability to another subject; a result of this type can be obtained by a simple action of a capability copy. A related problem is that of access right reduction. As seen in Section 2.3, reduction can be especially important to limit the access permissions of the recipient subject. In fact, reduction can be useful even within the boundaries of the same subject, to access an object with restricted privileges. This can be convenient to minimize the consequences of misbehaviour due to errors or faults, according to the *principle of least privilege* [27], [31].

In a classical capability environment, reduction requires that the *ar* field of the given capability is modified to suppress unwanted access rights. If the *ar* field features a bit for each access right, *ad hoc* mechanisms should be provided to access *ar* to clear the bits corresponding to the access rights to be eliminated. This is an apparent violation of capability segregation. It can be supported by a special capability instruction that allows a form of restricted access to the internal representation of capabilities, i.e. the bits in the *ar* field can be cleared but cannot be set by using this instruction.

Let us now consider a subject S that holds a password capability for a given object, and suppose that the password corresponds to a given set of access rights. If S is aimed at granting another subject a subset of these access rights, the password should be replaced with a new password, corresponding to this subset. In a password capability system in the original formulation, subject S is not in a position to forge the new password capability autonomously, as it does not possess the new password. Instead, S has to ask for intervention of a password manager, associated with the object type, and authorized to access the passwords.

In contrast, in our system, a subject can carry out gate reduction autonomously. No intervention of the protection system is necessary. We have obtained this important result by taking advantage of the parametric generation function, as has been shown in Section 2.3 and is illustrated in Figure 3.

6.3 Revocation

A subject that holds a capability for a given object can transfer this capability to a second subject, which in turn can transmit the capability further. It follows that access privileges tend to spread throughout the system. A related problem is that of access right revocation. Several solutions have been proposed to this problem [21]. A propagation graph can be constructed for each capability, which keeps track of all copies of this capability [7]. This solution tends to subvert one of the main advantages of the capability model, i.e. simplicity in the transmission of access privileges between subjects. We can limit the capability lifetime [17]. This solution tends to overburden the protection system with explicit requests to renew capability validity.

In a password capability system, if we change one of the passwords associated with a given object, we revoke all the password capabilities referencing this object in terms of that password. In our system, we support gate revocation through the base passwords (see Section 4). In a given cluster, if we change one of the base passwords associated with the cluster, we revoke the base gate defined in terms of this base password, and also all the gates derived from this base gate by reduction. In spite of its simplicity, this gate revocation mechanism possesses a number of interesting properties. Revocation is [7]:

- *selective*, that is, it can be limited to a subset of the subjects that possess a gate for the given cluster, if this subset has been associated with a specific base password and we change this base password;
- *independent*, that is, gates for the same set of domains can be revoked independently of each other, if these gates derive from base gates defined in terms of different base passwords;
- *transitive*, that is, if a gate is revoked, the effects of the revocation propagate to all the subjects that received a copy of this gate, independently of the distribution path followed by the gate copy to reach the corresponding recipient (and in fact, a copy of a gate is indistinguishable from the original, and gates have no memory of subsequent copy actions);
- *temporal*, and in fact, a revocation obtained by changing a given base password can be reversed through the same mechanism used for revocation, by restoring the original value of that base password.

6.4 Password proliferation

In a traditional password capability environment, several passwords are usually associated with each given object. In a possible approach, we have a password for each access right. A subject executing an operation permitted by a given access right must present a password capability to the protection system; this capability must be expressed in terms of the password associated with that access right. In this approach, a subject aimed at using a given object by issuing operations permitted by distinct access rights has to hold several password capabilities for this object. This places inappropriate burden on subjects. Furthermore, consider an operation whose execution requires two distinct access rights. The parameters of this operation will include two password capabilities, both referencing the same object. This undue complication deviates from the basic model of a single password capability for each operation.

In a different approach, a single password can be associated with more access rights. This approach enhances the object interface, at the expense of an increment in the number of passwords. Multiple passwords tend to become a security hazard; passwords can be lost or stolen, for instance [9], [10].

In our protection system, passwords are only used to select subsets of the domains of the given cluster. As seen in Section 3, each cluster is associated with a single access right, the *own* access right, which makes it possible to allocate new objects in the cluster, to delete the cluster, and to change its base password. This access right is encoded by taking advantage of owner domain d_0. A single password, the base password, must be maintained in memory for each cluster. All the other passwords, which are necessary to express gates in terms of subsets of the domains, can be generated dynamically starting from the base password, by means of the iterative procedure for password derivation, which has been outlined in Section 2.1. The number of access rights defined by the given object type is irrelevant to password proliferation. In fact, for each object in the given cluster, access rights are codified within that cluster, by the access control list associated with the object.

Thus, in a traditional password capability system we have several passwords for each object, whereas in our gate-based system we have a single password, the base password, for each object collection, i.e. a cluster. Significant advantages follow in terms of memory space requirements, simplicity in password management, and security. In fact, we take advantage of a synergy between the two traditional techniques for access right representation, access control lists and capability lists. We use access control lists to encode the access rights for general objects, and gates to encode the access rights for domains. As a result, we are in the position to ascertain the access rights included in a given domain for a general object at little effort, as is a characteristic of access control lists, while being able to determine the access rights held by a given subject for domains, as is specified by the gates held by that subject.

7 CONCLUDING REMARKS

We have considered an important problem in the design of a protection system, i.e. the definition of the mechanisms supporting the distribution, verification, review and revocation of access permissions. We have proposed the organization of a protection system that takes advantage of both access control lists, for the protection of ordinary objects, and a new form of password capability, the gate, for the protection of domains. In our approach:

- Ordinary objects are grouped into clusters. In a cluster, an access control list is associated with each object. The access control list specifies the access rights for this object that are included in each protection domain.
- A subject can access the objects in a given cluster only it possesses a gate for this cluster. The gate is a form of password capability, extended to contain a domain

selector that identifies one or more domains. A relation exists between the password in a given gate and a base password associated with the cluster when the cluster is created. This relation is expressed in terms of the domain selector, by application of a parametric one-way generation function.
- A gate for a given cluster identifies a virtual protection domain in terms of the domains of that cluster. The access rights in the virtual domain are the union of the access rights in the component domains.
- A small set of protection primitives allows subjects to manage objects and access control lists.

We have obtained the following results:

- A subject that holds a gate for a given cluster can generate a new gate for this cluster, to restrict the gate extent by eliminating domains. The gate reduction procedure can be iterated to eliminate more domains. Gate reduction is usually aimed at distributing restricted access privileges to another subject, but can be also used within the boundaries of the same subject, to access a cluster with less privileges to minimize the consequences of errors or faults. Subjects can carry out gate reduction autonomously; no intervention of the protection system is required.
- The duality of gates and access control lists allows us to carry out orthogonal forms of review and revocation of access privileges. A change of the access privileges in a given access control list affects only the gates referencing the domains involved in the change. On the other hand, the owner of a given cluster can replace the base password of that cluster; as a consequence, validity of all the gates referencing that cluster is revoked. The cluster owner can subsequently grant gates derived from the new password, thereby producing a new distribution of access rights. This gate revocation mechanism results to possess a number of interesting properties; revocation is selective, independent, transitive and temporal.
- If base passwords are large and sparse, it is impossible for a malevolent subject to forge valid gates from scratch. Transformation of a valid gate into a stronger gate referencing more domains is prevented by the one-way property of the parametric generation function.
- A gate can be shrunk into an equivalent gate featuring a single non-null primary selector. So doing, we keep the memory requirement for gate storage low, by reserving less space for the reduction field while not limiting the number of possible reductions. Furthermore, shorter execution times are required for gate validation.

ACKNOWLEDGEMENT

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

FUNDING

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] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: a survey. Technical report, Centre for Computer Security Research, Department of Computer Science, University of Wollongong, Australia, 1995.

[2] N. P. Carter, S. W. Keckler, and W. J. Dally. Hardware support for fast capability-based addressing. *ACM SIGPLAN Notices*, 29(11):319–327, November 1994.

[3] M. D. Castro, R. D. Pose, and C. Kopp. Password-capabilities and the Walnut kernel. *The Computer Journal*, 51(5):595–607, 2008.

[4] 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.

[5] M. de Vivo, G. O. de Vivo, and L. Gonzalez. A brief essay on capabilities. *ACM SIGPLAN Notices*, 30(7):29–36, July 1995.

[6] 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.

[7] V. D. Gligor. Review and revocation of access privileges distributed through capabilities. *IEEE Transactions on Software Engineering*, SE-5(6):575–586, November 1979.

[8] 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.

[9] E. Gudes. The design of a cryptography based secure file system. *IEEE Transactions on Software Engineering*, SE-6(5):411–420, 1980.

[10] L. Harn and H.-Y. Lin. A cryptographic key generation scheme for multilevel data security. *Computers & Security*, 9(6):539–546, 1990.

[11] H. R. Hassen, A. Bouabdallah, H. Bettahar, and Y. Challal. Key management for content access control in a hierarchy. *Computer Networks*, 51(11):3197–3219, 2007.

[12] 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.

[13] 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.

[14] A. Kwon, U. Dhawan, J. M. Smith, T. F. Knight Jr, and A. DeHon. Low-fat pointers: compact encoding and efficient gate-level implementation of fat pointers for spatial safety and capability-based security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pages 721–732, Berlin, Germany, November 2013. ACM.

[15] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[16] B. W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, January 1974.

[17] 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.

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

[19] L. Lopriore. Encrypted pointers in protection system design. *The Computer Journal*, 55(4):497–507, April 2012.

[20] L. Lopriore. Password capabilities revisited. *The Computer Journal*, 58(4):782–791, April 2015.

[21] L. Lopriore. Password management: distribution, review and revocation. *The Computer Journal*, 58(10):2557–2566, October 2015.

[22] 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.

[23] D. Mossop and R. Pose. Security models in the Password-Capability System. In *Proceedings of the TENCON 2005 – 2005 IEEE Region 10 Conference*, pages 1–6, Melbourne, Australia, November 2005. IEEE.

[24] 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.

[25] 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.

[26] 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.

[27] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[28] 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.

[29] R. S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27(2):95–98, 1988.

[30] R. S. Sandhu and P. Samarati. Access control: principles and practice. *IEEE Communications Magazine*, 32(9):40–48, September 1994.

[31] F. B. Schneider. Least privilege and more. *IEEE Security & Privacy*, 1(5):55–59, September 2003.

[32] 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*, Washington, DC, USA, October 2003. IEEE.

[33] M. Stipčević and Ç. K. Koç. True random number generators. In *Open Problems in Mathematics and Computational Science*, pages 275–315. Springer, 2014.

[34] 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.

[35] C. S. Wallace and R. Pose. Charging in a secure environment. In *Proceedings of the International Workshop on Computer Architectures to Support Security and Persistence of Information*, pages 85–96, Bremen, Germany, May 1990. Springer.

[36] R. N. Watson, J. Woodruff, P. G. Neumann, S. W. Moore, J. Anderson, D. Chisnall, et al. Cheri: a hybrid capability-system architecture for scalable software compartmentalization. In *Proceedings of the IEEE Symposium on Security and Privacy*, San Jose, California, USA, May 2015.

[37] M. V. Wilkes. Hardware support for memory protection: capability implementations. *ACM SIGARCH Computer Architecture News*, 10(2):107–116, March 1982.

[38] X. Zhang, Y. Li, and D. Nalla. An attribute-based access matrix model. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 359–363, Santa Fe, New Mexico, USA, March 2005. ACM.

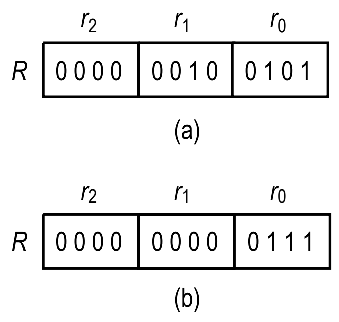

Figure 1: Two different configurations of domain selector R, corresponding to a single domain, d_3. In both cases, the cluster is supposed to include four domains. Consequently, the domain selector is partitioned into three primary selectors, $R = (r_2, r_1, r_0)$, and the size of each primary selector is four bits.

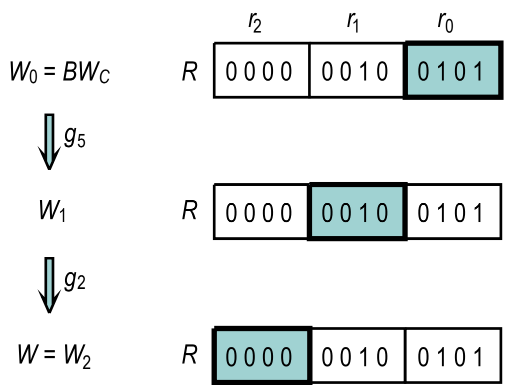

Figure 2: Evaluation of password W for a specific configuration of domain selector R of gate $G = (W, R)$.

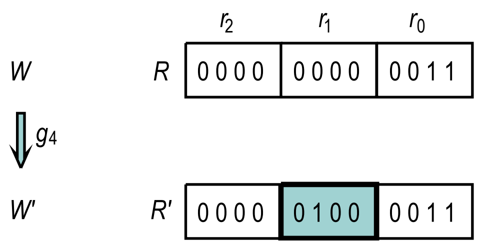

Figure 3: Reduction of gate $G = (W, R)$. In this example, the cluster defines four domains, and gate G references two domains, d_2 and d_3. After reduction, G references a single domain, d_3.

Table 1: The protection primitives.

$b \leftarrow newObject(G, d)$
Allocates a new object in the cluster referenced by gate G, and returns the name b of this object. The access control list ACL_b of b contains a single element, which includes full access rights for b in domain d. Requires that G references both d and the owner domain d_0.

$deleteObject(G, b)$
Deletes object b from the cluster referenced by gate G. Requires that G includes access right *own* for b.

$res \leftarrow operation(G, b, op)$
In the cluster referenced by gate G, executes operation op on object b, and returns the result of this operation. Requires that G includes the access rights for b that are required by op.

$b' \leftarrow copyObject(G, b)$
In the cluster referenced by gate G, creates a copy of object b and returns the name b' of the copy. Requires that G includes access right *copy* for b.

$addAr(G, d, b, ar)$
In the cluster referenced by gate G, adds access right ar to the element reserved for domain d in the access control list ACL_b of object b. Requires that G includes access right ar for b.

$removeAr(G, d, b, ar)$
In the cluster referenced by gate G, removes access right ar from the element reserved for domain d in the access control list ACL_b of object b. Requires that G includes access right *own* for b.