Representation and management of multiple keys

Lanfranco Lopriore

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

Abstract—We refer to a key-based protection environment featuring active subjects and protected objects. A subject that possesses a key for a given object is allowed to access this object to carry out actions that depend on the specific application. We propose a new key model, called multiple key, or m-key for short. In this model, an m-key consists of a name, a value and a map. The name specifies a set of objects, and the value is used to validate the m-key. Possession of a valid m-key is equivalent to possession of a key for all the objects specified by the m-key name, or for a subset of these objects, as is stated by the map. A subject that holds an m-key referencing a given set of objects can transform this m-key into a weaker m-key referencing only a subset of these objects. M-key revocation is supported.

Keywords: key; one-way function; protection; revocation.

1 INTRODUCTION

We shall refer to a protection system paradigm whereby active entities called *subjects* generate access attempts to passive, protected entities called *objects* [1]. Objects can be *simple* or *typed*. A single operation is permitted on a simple object (*s-object* from now on, for short), i.e. to access that s-object, whereas for a typed object a set of operations is defined, which depends on the type. The type specifies a set of *access rights*, and determines the association between the operations and the access rights. A subject can execute a given operation on a typed object only if it possesses the access rights required for that operation and that typed object [2].

In a classical representation, this protection paradigm takes the form of a matrix, called the *access matrix* AM, featuring a row for each subject and a column for each object [3]. Let $S_0, S_1, \ldots$ denote the subjects, and $b_0, b_1, \ldots$ denote the objects. Element $AM_{i,j}$ of the access matrix specifies whether subject S_i holds an *access privilege* for object b_j. If b_j is an s-object, the privilege permits accesses in the form specified by the s-object definition, whereas if b_j is typed, the privilege specifies a set of access rights that determines the permitted operations.

1.1 Keys

An important aspect of every protection paradigm is the method to encode access privileges in memory. A solution is to associate *keys* with objects and privileges. A subject that possesses a key for a given object can use this key to access the object to exercise the corresponding privilege.

As an example involving s-objects, let us consider a binder containing a set of documents. In this example, the subjects are the potential readers and the s-objects are the documents in the binder. In the access matrix model, we have a row for each reader, and a column for each document. The matrix element corresponding to a given reader and a given document specifies whether that reader possesses the privilege to access the document. In this case, in a key-based implementation of the access matrix, the reader possesses a key for the document.

As a further example, let us refer to an environment supporting enciphered messages [4], [5]. Each message consists of a header and a body. The header is in plaintext, the body is enciphered by using a symmetric-key algorithm and a cryptographic key whose name is written in the header. A subject that possesses a given cryptographic key can read the contents of all the messages enciphered by using that key. This example shows that a single key can grant an access privilege for several s-objects. In the access matrix model, each element in a column corresponding to a message enciphered by using that key may specify the access privilege to decipher the message.

As a final example, let us refer to a protection environment in which objects are typed. In a key-based approach, a key is associated with each pair (*typed object*, *access privilege*). A subject that possesses a given key is allowed to access the named typed object to carry out the operations permitted by the access rights in the privilege. In the access matrix, the elements in the column of that typed object specify the subjects that the hold access privileges for that object.

1.1.1 Proliferation

In a key-based protection environment, subjects tend to possess varieties of keys. In the aforementioned examples, we have a key for each document in the binder; several keys are necessary to decipher messages from different senders; and, for a typed object, a key is associated with each object and each meaningful access privilege. The necessity to maintain collections of different keys leads to undesirable complications in key management, and to inefficiencies in terms of the memory requirements for key storage.

1.1.2 Weakening

A subject that possesses a given key can transfer a copy of this key to another subject. This is equivalent to granting the full access privilege corresponding to that key. In fact, a copy of a key is indistinguishable from the original, and possession of the copy is equivalent to possession of the original. On the other hand, we may well be aimed at granting only a fraction of the privilege. This means that the protection system should permit actions of key *weakening*. In the example of the typed objects, weakening means to eliminate access rights from the access privilege associated with the given key, so that the recipient of the key copy can access the corresponding typed object only for a subset of the operations.

1.1.3 Revocation

A subject that received a key can distribute the key further. In fact, keys tend to spread throughout the system. A related requirement is the possibility for the original subject to revoke the key, so that it will no longer possible to use this key to exercise the corresponding access privilege [6]. Revocation should transitively extend to every subject that received a key copy. Transitivity should also apply to all the weakened versions of the original key, so that it is impossible to take advantage of the weakened copies to circumvent revocation.

The key revocation mechanism should be non-invasive. In fact, an important requisite for a key model is simplicity and efficiency in key verification and transmission between subjects. Revocation should not affect the overall system performance negatively. This would be the case if, for instance, every activity involving a given key, e.g. generation of a key copy, should be validated by a pervasive protection monitor [7].

1.2 Capabilities

Capabilities are an effective implementation of the key concept [8], [9]. A capability is a pair (ID, AP), where ID is the unique identifier of a protected object, and AP is the specification of an access privilege for this object. In a capability environment, each subject holds a collection of capabilities, which specifies the objects it can access and the operations it can execute on these objects, as made possible by the access privileges.

1.2.1 Segregation and proliferation

Segregation is a peculiar problem of capability systems. We should prevent undue modifications of capabilities, to add access rights to the access privileges, or to change the identifiers to reference different objects (which is equivalent to forge new capabilities).

Several solutions have been proposed to the capability segregation problem [10]. In a segmented memory system, special segments, which we shall call *capability segments*, can be reserved for capability storage (in contrast, *data segments* will contain ordinary information items) [11]. The instruction set of the processor is augmented to include special instructions for capability processing. These *capability instructions* make it possible to access the contents of capability segments; if an ordinary data instruction is used, an exception of violated protection is raised, and execution fails.

Segment-based capability segregation forces subjects to adhere to complicated schemes of object management. For instance, for a given object, at least one capability segment is necessary to contain the capabilities for the data segments storing the internal representation of the object. Undesirable capability proliferation and memory wastes may follow. This is especially true if protection should be exercised at a high granularity level, for small-sized objects.

In a different approach, each memory cell includes a one-bit *tag*, which specifies whether this cell contains a capability, or an ordinary information item [12], [13]. Only capability instructions can be executed on a cell tagged to contain a capability. This approach implies *ad hoc* memory modules designed to contain the tags; this is contrary to the requisite of hardware standardization. Complications arise in memory management, e.g. for the necessity to save, and then to restore, the tags whenever a segment is swapped to the secondary memory.

1.2.2 Weakening

Weakening is easy in capability systems. In a simple implementation, the access privilege field includes one bit for each access right; if asserted, the given bit specifies that the capability grants the corresponding access right. The instruction set of the processor includes a capability instruction that makes it possible to clear (but not to set) these bits. This instruction will be used to weaken the given capability, by clearing the bits that correspond to the unwanted access rights.

1.2.3 Revocation

A subject that granted a given capability should be in the position to revert the grant, and retract the corresponding access privilege from the recipient [10]. Solutions have been devised to the capability revocation problem, which include a propagation graph associated with the given capability, to keep track of all copies of this capability throughout the memory system [8]; a reference monitor associated with the given protected object, to mediate all accesses to this object [7]; and short-lived capabilities whose lifetime should be periodically renewed [14]. These solutions are in sharp contrast with the requirement of simplicity of access privilege transmission between subjects, which was one of the original motivations for the introduction of the capability concept [15].

1.3 Password capabilities

Password capabilities are an important improvement on the capability paradigm [10], [16], [17]. In a password capability environment, a set of passwords is associated with each given object; each password corresponds to a specific access privilege for this object. A password capability is a pair (ID, W), where ID is an object identifier, and W is a password. If W matches one of the passwords associated with the object identified by ID, then the password capability grants the access privilege for the object, which corresponds to this password.

1.3.1 Segregation and revocation

If passwords are large, sparse and chosen at random, a malevolent subject will not be able to forge a password capability from scratch, or to modify an existing password capability to include the password for a stronger privilege [18]. This means that password capabilities can be mixed in memory with ordinary information items, and are an effective solution to the segregation problem. Furthermore, if we change one of the passwords associated with a given object, we revoke all the password capabilities defined in terms of that password, and this is an effective solution to the revocation problem.

1.3.2 Proliferation and weakening

Granularity in the specification of access privileges implies that many passwords are associated with the same given object, one password for each significant privilege. This leads to password proliferation, with negative effects on simplicity in password management, and on the memory requirements for password storage.

A related problem is that of password capability weakening. A subject that holds a given password capability is not in the position to weaken this password capability autonomously. Instead, it should ask for the intervention of an *ad hoc* component of the protection system, that we shall call the *object manager*, associated with the given object, and responsible for password capability weakening. The object manager receives a password capability, and returns a new password capability expressed in terms of a weaker password. An alternative, less complicated solution would be desirable, whereby a subject that holds a password capability can weaken this password capability autonomously, with no intervention of the protection system.

1.4 Multiple keys

With reference to an environment featuring subjects and protected objects, this paper is aimed at presenting effective solutions to the problems, outlined above. We shall introduce a new key model, called *multiple keys*, or *m-keys* for short. In this model, an m-key consists of a *name*, a *value* and a *map*. The name specifies a set of s-objects, and the value is used to validate the m-key. Possession of a valid m-key is equivalent to possession of a key for each s-object specified by the m-key name, or for a subset of these s-objects, as is stated by the map. Thus, the m-key construct allows us to encode several distinct keys in a single m-key, and m-keys are an effective solution to the proliferation problem. We shall show that:

- M-keys give effective support to segregation, unforgeability, weakening and revocation.

- The memory requirements for storage of an m-key is comparable to that of a single key.
- Any illegitimate attempt to modify an m-key to include a stronger privilege is hampered. In particular, a subject that holds an m-key generated by weakening is prevented from amplifying this m-key to reference more s-objects, and possibly return to the original, unweakened m-key.
- The m-key concept is flexible, and can be used efficiently to solve a variety of protection problems, which correspond to different meanings associated with the concept of an object.

The rest of this paper is organized as follows. Section 2 introduces the m-key model with special reference to the relation existing between the value of an m-key and a *master value* permanently associated with the name of this m-key. The problems of m-key validation, weakening, and revocation are considered in special depth. Section 3 illustrates a few examples of application of the m-key concept, which include the protection of collections of strictly-related homogeneous objects, the management of sets of cryptographic keys, and the access privileges for typed objects. Section 4 discusses the m-key model from a number of important viewpoints, which include implementation issues, the memory requirements for m-key storage, and the relation of our work to previous work. Section 5 gives concluding remarks.

2 THE MULTIPLE KEY MODEL

Let $b_0, b_1, \ldots, b_{n-1}$ be a set of n s-objects, and let $B = (b_{n-1}, b_{n-2}, \ldots, b_0)$ be the quantity obtained by concatenating (joining) the names of these s-objects. M-key K is a triple (B, V, M) where B is the m-key name, V is the m-key value, and M is a map. We say that the m-key *references* a given s-object if the validity of the m-key extends to this s-object. For each given s-object b_i in B, the map M indicates whether m-key K references b_i, or not; in the affirmative case, possession of K implies possession of a key for b_i. Furthermore, as will be illustrated shortly, the map describes the relation existing between quantity V and an m-key value, called the *master value* and denoted by $\underline{V}$, which is permanently associated with m-key name B.

Map M is divided into $n-1$ *submaps*, i.e. $M = (m_{n-2}, m_{n-3}, \ldots, m_0)$, where m_i denotes the i-th submap. The size of each submap is n bits, and the i-th bit corresponds to the i-th s-object, b_i. If the i-th bit is cleared in all the submaps of m-key K, then K references b_i, and possession of K implies possession of a key for b_i. Conversely, if the i-th bit is asserted in one or more submaps, then K does not reference b_i, even if b_i is named in B.

A submap is *cleared* if its value is 0. Cleared submaps are always placed in the most significant positions of the map, at the highest order numbers. It follows that, if m_j is cleared, then m_k is cleared for all $k > j$. We shall take advantage of this property shortly, in m-key validation. Possession of an m-key whose submaps are all cleared (i.e. $M = 0$) is equivalent to possession of a key for each s-object included in the m-key name. In this case, $V = \underline{V}$, and the resulting m-key has the form $(B, \underline{V}, 0)$. This m-key is called the *master key* of the s-objects named in B, and is denoted by $\underline{K}$. Thus, $\underline{K} = (B, \underline{V}, 0)$.

2.1 M-key evaluation

Function f is *one-way* if given x, it is easy to determine $f(x)$, and given $f(x)$, it is computationally unfeasible to determine x [19]. In our protection system, to generate m-key values, we take advantage of a one-way function, called the *base function* and denoted by f.

Let $K = (B, V, M)$ be an m-key. The value V of the m-key is divided into n *subvalues*, i.e. $V = (v_{n-1}, v_{n-2}, \dots, v_0)$, where v_j denotes the j-th subvalue, and n is the number of s-object names that form B. The relation existing between V and master value $\underline{V}$ associated with B is specified in terms of repeated applications of base function f, one application for each submap of map M that is not cleared. In detail, if $M = 0$ then we have $V = \underline{V}$; otherwise, quantity V is obtained by iteratively evaluating quantity $V_{i+1} = f(U_i), i = 0, 1, \dots$ etc., where $V_0 = \underline{V}$. Quantity U_i is called the *mapped complement* of V_i, and is the result of a bitwise complement of each subvalue of V_i that corresponds to an asserted bit of m_i (i.e. if the k-th bit of m_i is asserted, then we complement v_k). The iterations terminate at the submap, say m_j, that precedes the first cleared submap (i.e. $m_{j+1} = 0$), and in this case $V = V_{j+1}$. If no submap is cleared, the iterations terminate at the last submap m_{n-2}, when $V = V_{n-1}$.

Figure 1 shows the evaluation of quantity V for a specific configuration of map M in m-key $K = (B, V, M)$. In this example, $n = 4$, that is, the m-key name includes four s-object names, which we shall denote by b_0 to b_3. Thus, $B = (b_3, b_2, b_1, b_0)$. Map M consists of three submaps, i.e. $M = (m_2, m_1, m_0)$, and the size of each submap is four bits. If $M = (0000, 0001, 0110)$, bit 3 is the only bit that is cleared in all submaps. Consequently, m-key K references a single s-object, b_3. This means that possession of K is equivalent to possession of a single key, for b_3. Value V is partitioned into four subvalues, i.e. $V = (v_3, v_2, v_1, v_0)$. Let us denote the bit configuration of the generic subvalue by $xx\dots x$, and the corresponding bitwise complement by $\bar{x}\bar{x}\dots\bar{x}$. In the evaluation of V, in the first iteration we have $V_0 = \underline{V}$. Submap m_0 is 0110, and consequently, subvalues v_1 and v_2 are bitwise complemented. The result U_0 is transmitted to function f that produces V_1. In the second iteration, submap m_1 is 0001, and consequently, subvalue v_0 is bitwise

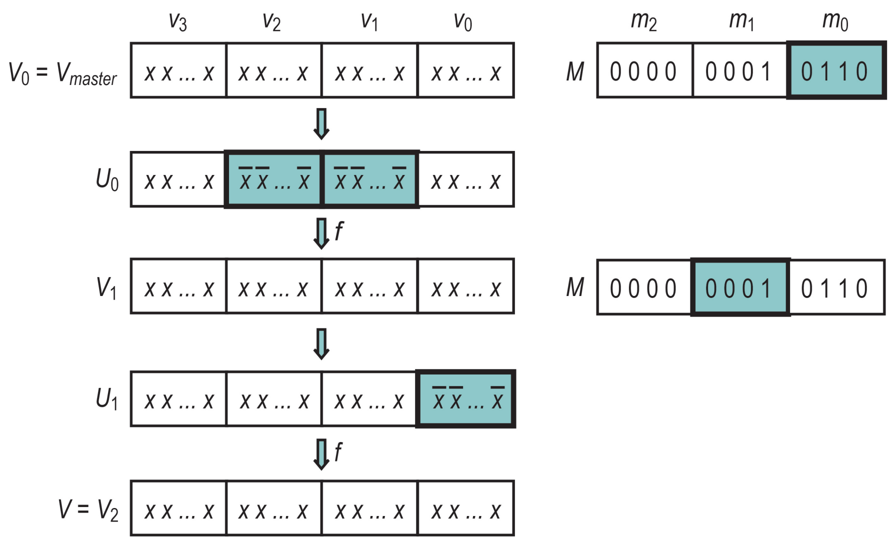

Figure 1: Evaluation of quantity V for a specific configuration of map M in m-key $K = (B, V, M)$. In this example, the m-key name includes four s-object names. Consequently, the value V of the m-key is partitioned into four subvalues, v_0 to v_3, map M is partitioned into three submaps, m_0 to m_2, and the size of each submap is four bits.

complemented. The result U_1 is transmitted to function f that produces V_2. Submap m_2 is cleared; this terminates the iterations, and $V = V_2$.

2.2 M-key validation

Let $K = (B, V, M)$ be an m-key, and let $\underline{V}$ be the master value associated with m-key name B. K is *valid* if its value V corresponds to the value we obtain starting from $\underline{V}$ and using map M in the iterative evaluation process described in Section 2.1.

In detail, let $B = (b_{n-1}, b_{n-2}, \ldots, b_0)$, where b_i denotes an s-object name, let $V = (v_{n-1}, v_{n-2}, \ldots, v_0)$, where v_i denotes a subvalue, and let $M = (m_{n-2}, m_{n-3}, \ldots, m_0)$, where m_i denotes a submap. If $M = 0$ (i.e. all the submaps are cleared) and $V = \underline{V}$, then m-key K is valid and it references $b_{n-1}, b_{n-2}, \ldots, b_0$ (that is, possession of K is equivalent to possession of a key for each s-object named in B). If $M \neq 0$, the m-key is validated by using base function f and the submaps of M to evaluate quantity V' corresponding to M. Thus we have $V_0 = \underline{V}$, and $V_{i+1} = f(U_i), i = 0, 1, \ldots$ etc., where U_i is the mapped complement of V_i. The sequence terminates at the submap, say m_j, that precedes the first cleared submap (i.e. $m_{j+1} = 0$), and in this case $V' = V_{j+1}$. If no submap is cleared, the iterations terminate at the last submap m_{n-2}, when $V' = V_{n-1}$. M-key K is valid if $V' = V$. If K is valid, and the i-th bit is cleared in all the submaps, then K references s-object b_i (that is, possession of K is equivalent to possession of a key for b_i).

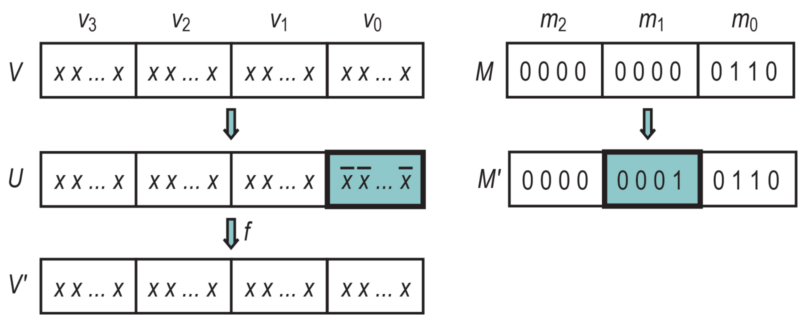

Figure 2: Weakening of m-key $K = (B, V, M)$ that references two s-objects, b_0 and b_3, into m-key $K' = (B, V', M')$ that references a single s-object, b_3.

2.3 M-key weakening

Let us consider a subject S that holds m-key $K = (B, V, M)$, let $B = (b_{n-1}, b_{n-2}, \dots, b_0)$, where b_i denotes an s-object name, and let I be the set of all the s-objects referenced by K, according to map M. Subject S may wish to transfer a copy of K to a different subject S'. This action is equivalent to the grant of a key for each s-object in I. Subject S can even transfer an m-key referencing a subset I' of the s-objects in I. To this aim, S converts K into a weaker m-key $K' = (B, V', M')$ that references only the s-objects in I'. This action is called *m-key weakening*, and can be carried out by subject S autonomously, as is detailed below.

Suppose that m-key K references s-object b_i, and we are aimed at excluding this s-object from a weaker m-key K'. Let $m_0, m_1, \dots, m_{n-2}$ be the submaps of map M, and let m_j be the first cleared submap (i.e. $m_k = 0$ for all $k \geqslant j$). Map M is transformed into map M' by setting the i-th bit of m_j, which corresponds to b_i. Then, value V is transformed into value V' by applying base function f. Thus we have $V' = f(U)$, where U denotes the mapped complement of V obtained by using m_j.

It should be noted that, for n s-objects and $n - 1$ submaps, the map of an m-key referencing two or more s-objects always contains at least one cleared submap, and can be weakened (conversely, if no map is cleared, then the m-key contains a single s-object, and cannot be weakened).

Figure 2 shows the weakening of m-key $K = (B, V, M)$ for a specific configuration of map M. In this example, $n = 4$, that is, the m-key name includes four s-object names. Thus, $B = (b_3, b_2, b_1, b_0)$. Map M consists of three submaps, i.e. $M = (m_2, m_1, m_0)$, and the size of each submap is four bits. If $M = (0000, 0000, 0110)$, bits 0 and 3 are cleared in all the submaps, and consequently, K references b_0 and b_3 (i.e. possession of m-key K is equivalent to possession of a key for b_0 and a key for b_3). Let us suppose that we are aimed at weakening m-key K into m-key $K' = (B, V', M')$ to eliminate b_0, so that K' will reference a single s-object, b_3. We modify the first cleared submap of M, i.e. submap

m_1; the new value is 0001 to indicate that s-object b_0 is excluded from K'. Quantity U is obtained by bitwise complementing subvalue v_0 of V, corresponding to bit 0 of m_1, which is asserted. Base function f is applied to U to obtain $V' = f(U)$.

2.4 M-key revocation

Let $B = (b_{n-1}, b_{n-2}, \ldots, b_0)$ be an m-key name, where b_i denotes an s-object name, and let $\underline{V}$ be the master value associated with B. As seen in Section 2.2, validation of m-key $K = (B, V, M)$ is an iterative procedure that starts at $\underline{V}$. The procedure determines a value V' by successive transformations involving map M and base function f. M-key K is valid if V' matches m-key value V. It follows that if we modify $\underline{V}$, we invalidate all the m-keys that originate from $\underline{V}$, that is, master m-key $\underline{K} = (B, \underline{V}, 0)$ and all the m-keys derived from $\underline{K}$ by weakening. It will no longer be possible to use these m-keys to reference the corresponding s-objects.

Now suppose that we associate two master values with B, say $\underline{V}_1$ and $\underline{V}_2$. The corresponding master m-keys are $\underline{K}_1 = (B, \underline{V}_1, 0)$ and $\underline{K}_2 = (B, \underline{V}_2, 0)$. If we change a master value, say $\underline{V}_1$, we invalidate the corresponding master m-key $\underline{K}_1$ and all the m-keys derived from this master m-key by weakening. On the other hand, validity of $\underline{K}_2$ and all the m-keys derived from $\underline{K}_2$ is unaffected. These considerations can be extended to an arbitrary number of master values.

3 EXAMPLES OF APPLICATION

M-keys can be used in a variety of applications, corresponding to different meanings associated with the concept of an s-object. This section presents a few examples of these applications, where s-objects correspond to strictly related entities, to cryptographic keys, and to different access rights for a typed object. This is by no means exhaustive, but it gives an indication of the flexibility of the m-key concept.

3.1 Homogeneous objects

As an example of application involving a collection of strictly related, homogeneous objects, let us consider a binder B containing n documents. An m-key for the binder has the form (B, V, M), where $B = (d_{n-1}, d_{n-2}, \ldots, d_0)$, and d_i denotes the name of a document. A master value $\underline{V}$ is associated with the binder. Map M states the relation existing between $\underline{V}$ and m-key value V. The master m-key for B has the form $\underline{K} = (B, \underline{V}, 0)$, where the map equal to 0 indicates that $\underline{K}$ references all the documents in the binder (i.e. possession of $\underline{K}$ is equivalent to possession of a key for each document in the binder). As shown in Section 2.3, $\underline{K}$ can be transformed into a weaker m-key $K = (B, V, M)$. In this case, map

M has a non-zero value that indicates the relation existing between master value $\underline{V}$ and m-key value V, and determines the documents that are actually referenced by the m-key. Possession of m-key K is equivalent to possession of a key for each of these documents.

Let S be a subject aimed at accessing document d_i in binder B. S should possess an m-key $K = (B, V, M)$ for the binder, and this m-key should reference d_i. When the m-key is sent to the binder, the i-th bit of each submap of map M is inspected. If one or more of these bits are asserted, K does not reference document d_i, and the access to this document is denied. If all these bits are cleared, master value $\underline{V}$ is used to validate K. This means that quantity V' corresponding to M is evaluated by carrying out the iterative procedure delineated in Section 2.2, which involves base function f and the submaps of M. If $V' = V$, validation terminates successfully, and the access to d_i is finally permitted.

3.2 Cryptographic keys

Let us consider a message-passing environment, where messages are enciphered by a symmetric-key algorithm. A message consists of a header in plaintext, and a body in ciphertext. The message header includes the name k of a cryptographic key; the value of this key has been used to encipher the message body. Subject S that enciphered a message by using k can authorize a recipient subject R to read this message by granting k to R. The authorization can be transitively extended to other subjects by copying k to these subjects. M-keys can be effectively used to simplify these copy actions. In particular, we can transfer several cryptographic keys by a single m-key copy.

In this application, in m-key $K = (B, V, M)$, we have $B = (k_{n-1}, k_{n-2}, \ldots, k_0)$, where k_i denotes a cryptographic key. A *key manager* maintains a key table featuring an entry for each cryptographic key. The entry for key k_i contains the value v_i of this key. Furthermore, the key manager associates a master value $\underline{V}$ with B. In m-key K, map M indicates the relation existing between $\underline{V}$ and m-key value V. If $M = 0$, we have the master m-key $\underline{K} = (B, \underline{V}, 0)$; possession of $\underline{K}$ implies possession of all the cryptographic keys. If $M \neq 0$, possession of m-key $K = (B, V, M)$ implies possession of a subset of all the cryptographic keys, and the composition of this subset is stated by M.

A subject that holds m-key K can ask the key manager for the value v of the i-th cryptographic key k_i by executing operation $v \leftarrow keyValue(K, i)$. Execution of this operation of the key manager is as follows. The i-th bit is inspected in each submap of map M; if this bit is asserted in one or more submaps, then K does not reference k_i, and operation *keyValue* fails. Otherwise, master value $\underline{V}$ is used to validate K, as has been illustrated in Section 2.2. If validation fails, then *keyValue* fails; otherwise, the key table is accessed to find the value v_i of k_i, which is finally returned to the caller.

A subject S that holds an m-key K referencing a set of cryptographic keys can transfer these cryptographic keys to another subject S' by simply copying K to S'. Subject S can also transfer an m-key for a subset of the cryptographic keys, e.g. a single key. To this aim, S weakens K, as has been illustrated in Section 2.3, to eliminate the unwanted keys; no intervention of the key manager is necessary.

3.3 Typed objects

Let us refer to a protection environment in which objects are typed. Subject S can access object G of type T to execute operation *op* only if it possesses the access privilege associated with *op*. The access privilege is expressed in terms of a collection of access rights. As seen in Section 1, a classical protection problem is to represent the access privileges. M-keys and our protection system are a solution to this problem.

Let $ar_0, ar_1, \ldots, ar_{n-1}$ be the access rights defined by type T, and let an s-object correspond to a pair (*typed object*, *access right*). Thus, for a given object G of type T, we define n s-objects, one s-object for each access right defined by T. Let b_i be the s-object associated with access right ar_i. An m-key referencing G is a triple (B, V, M), where $B = (b_{n-1}, b_{n-2}, \ldots, b_0)$. This means that the name of the m-key includes the s-objects for all the access rights. The map states the relation existing between value V of the m-key and master value $\underline{V}$ associated with B. If $M = 0$, we have the master m-key $\underline{K} = (B, \underline{V}, 0)$, which grants full access rights for G. If $M \neq 0$, possession of m-key $K = (B, V, M)$ is equivalent to possession of a subset of the access rights, and the composition of this subset is stated by the map.

Therefore, subject S that possesses an m-key K for a given typed object G is in the position to access G to perform the operations permitted by the access rights included in K, according to the m-key map. Subject S may even transfer these access rights to a different subject S', completely or in part. A simple m-key copy action is sufficient to transfer all the access rights. For a partial access right transfer, S will generate a weaker version of K, say $K' = (B, V', M')$, where M' is obtained by modifying M to exclude the s-objects corresponding to the unwanted access rights. To this aim, as seen in Section 2.3, subject S modifies the first submap of M whose value is 0, say submap m_j, to set the bits corresponding to the s-objects to be excluded. Then, S uses m_j and base function f to evaluate V' starting from V. The new m-key K' will be transferred to subject S'.

As an example of implementation, let us consider the *Buffer* object type. The definition of this type includes four access rights, namely *delete*, *copy*, *insert*, and *extract*. Possession of the *delete* access right for a given buffer allows us to delete the buffer, *copy* makes it possible to generate a buffer copy, *insert* allows a producer subject to add new data items into the buffer, and *extract* allows a consumer subject to extract data items from the buffer. Thus, for a given instantiation of the *Buffer* type, we have four access rights, which correspond to four s-objects, namely b_0 for access right *delete*, b_1 for *copy*, b_2 for *insert*, and b_3 for *extract*. When subject S creates a buffer, a master value $\underline{V}$ is generated at random and is permanently associated with the buffer as part of its internal representation. Subject S receives a master m-key for the buffer, $\underline{K} = (B, \underline{V}, 0)$, where $B = (b_3, b_2, b_1, b_0)$, and the map equal to 0 indicates that all access rights are included in the master m-key. Subject S can transfer these access rights to a different subject S', completely or in part, by granting S' a copy of $\underline{K}$, or an m-key generated from $\underline{K}$ by weakening.

As seen in Section 2.4, m-keys can be revoked by changing the corresponding master value. In our example of the *Buffer* type, this action is made possible by a specific operation, which has the form $\underline{K}' \leftarrow revoke(\underline{K})$. This operation receives the old master m-key $\underline{K}$ as a parameter; this is necessary to verify that the caller holds full access rights for the buffer. Execution replaces $\underline{V}$ with a new value $\underline{V}'$, and returns a master m-key $\underline{K}'$ defined in terms of $\underline{V}'$. As a result, all the m-keys originated from $\underline{V}$ (i.e. $\underline{K}$, as well as all the m-keys derived from $\underline{K}$ by weakening) are invalidated; it will no longer possible to use these m-keys to access the buffer. After execution of *revoke*, $\underline{K}'$ is the single valid m-key for the buffer. The subject that executed *revoke* has the opportunity to carry out a new distribution of access rights, by granting copies of $\underline{K}'$ as well as of m-keys derived from $\underline{K}'$ by weakening.

4 DISCUSSION

4.1 M-key forging

Let us suppose that subject S attempts to forge an m-key for a set of s-objects from scratch. Let $K = (B, V, M)$ be the result of this attempt. M-key name B will be simply obtained by joining the s-object names, i.e. $B = (b_{n-1}, b_{n-2}, \ldots, b_0)$. Map M will be set to 0, so that the m-key will reference all the s-objects. However, S does not know the master value $\underline{V}$ associated with B, and consequently, it will use a value chosen at random. If master values are sparse and large, the probability of a casual match is virtually null, and the m-key forging attempt is destined to fail. Of course, similar considerations also apply to any weaker m-key, corresponding to a non-zero value of map M.

Let us now suppose that subject S possesses a valid m-key $K = (B, V, M)$. If $M \neq 0$, this m-key references only a subset of the s-objects named in B. Suppose that S is aimed at amplifying K to obtain m-key $K' = (B, V', M')$, where M' is less restrictive than M (e.g. if submap $m_j \neq 0$ and submap $m_k = 0$ for $k > j$, we clear m_j). In this case, V' should *precede* V in the iterative procedure illustrated in Section 2.1, and used to evaluate V starting from master value $\underline{V}$. However, base function f is one-way, and can cannot be inverted. This means that it is impossible to use m-key value V corresponding to map

M to determine m-key value V' corresponding to map M'. In this case, too, if S uses a random V', the probability of a casual match is vanishingly low.

4.2 M-key revocation

As seen in Section 2.4, by changing the master value $\underline{V}$ associated with m-key name B, we revoke the corresponding master m-key $\underline{K} = (B, \underline{V}, 0)$, and all the m-keys derived from $\underline{K}$ by weakening. If more master values are associated with B, and we change one of them, the m-keys derived from the other master values are not affected by the change.

Despite its simplicity, this mechanism for m-key revocation possesses several interesting properties [8]. Revocation is *selective*, that is, it can be limited to a subset of the m-keys for a given set of s-objects, if a specific master value is associated with this subset, and we change this master value. Revocation is *transitive*, that is, the effects of the revocation of a given m-key automatically extend to all the copies of this m-key (in fact, the copy of an m-key is indistinguishable from the original). Revocation is *independent*, that is, m-keys for the same collection of s-objects can be revoked independently of each other, if these m-keys derive from different master m-keys. Revocation is *temporal*, that is, it can be reversed automatically through the same mechanism used for revocation, by restoring the master value preceding revocation.

4.3 Implementation issues

In a possible, effective implementation of an m-key system, we shall define a limited number of m-key formats, corresponding to prefixed values of the number n of s-objects in an m-key. We may have a *short* m-key for up to four s-objects ($n = 4$), a *standard* m-key for up to eight s-objects ($n = 8$), and a *long* m-key for up to 16 s-objects ($n = 16$). For instance, in the standard m-key format, the m-key value is partitioned into eight subvalues, v_0 to v_7. The map consists of seven submaps, m_0 to m_6, and the size of each submap is eight bits.

4.3.1 Clusters

As seen in Section 2, the name of an m-key is obtained by joining the names of the component s-objects. Let m be the size of an s-object name, in bytes; for n s-objects, an m-key name is $n \cdot m$ bytes wide. For instance, $m = 4$ makes it possible to generate a large number of s-object names. In the standard m-key format, introduced above, we have eight s-objects, and the size of an m-key name is 32 bytes.

These memory requirements can be significantly reduced if the s-objects in the same m-key are numbered in sequence to form a *cluster*. In the example of a collection of homogeneous objects, considered in Section 3.1, this means that the documents in a given binder will correspond to consecutive objects. In the cryptographic key example of Section 3.2, the keys will be assigned consecutive names. Finally, in the typed object example of Section 3.3, the access rights for the same given typed object will be numbered in sequence.

If s-objects are clustered, the most significant portion of the name of a given s-object identifies the cluster, and the least significant portion is a *local* s-object number, equal to the order number of that s-object in the cluster. The name of an m-key for the cluster is simply given by the cluster name. If the cluster consists of fewer s-objects than the capacity of the m-key format, the redundant s-objects will be excluded from the m-key by taking advantage of the m-key map.

For instance, for 32-bit s-object names and clusters of eight s-objects, the 29 most significant bits of an s-object name identify the cluster, and the three least significant bits are the local name of the s-object in that cluster. In the standard format, the m-key supports up to eight s-objects. For a smaller cluster, e.g. a cluster of six s-objects, b_6 and b_7 will be excluded from the m-key by setting the two most significant bits of submap m_0; the m-key value will be evaluated accordingly, as is the case in m-key weakening (see Section 2.3). The map consists of seven submaps, and the size of each submap is eight bits. If the size of an s-object name is 32 bits and the size of an m-key value is 128 bits, the resulting m-key size is 27 bytes, which is comparable to that of a single key (20 bytes), and is much less than the memory requirements for storage of eight separate keys (160 bytes).

4.4 Previous work

Capabilities and password capabilities have received much attention for many decades [15]. Several systems were conceived that support forms of capability-based protection. This section takes a few significant examples of these systems into consideration. The aim is a comparison with the design principles that inform the m-key protection model.

PSOS [20] is a capability operating system using tagging for capability segregation, in the processor as well as in the primary and secondary memory. The tag bits cannot be altered by ordinary processor operations, a single exception being the two instructions, to create a new capability and to restrict a capability to include less access rights. A capability consists of a unique identifier and a set of access rights taking the form of a Boolean array. The instruction to weaken a capability creates a new capability with the same identifier, and applies a mask to the access rights. The ability to copy a capability can be restricted, e.g. if we are aimed at permitting a user to access a given object but not to transmit this privilege to others. A result of this type is obtained by an access right that grants store permission; each memory segment can be limited to contain only those capabilities that include this access right.

CHERY [13] is a design and implementation effort using a capability coprocessor to extend the 64-bit MIPS instruction set architecture. The MIPS pipeline transmits instructions to the coprocessor, it exchanges operands with the coprocessor, and receives exceptions from the coprocessor. The aim is to give efficient support to a hybrid approach to memory protection whereby a memory management unit supports coarse-grained inter-process separation, and capabilities implement fine-grained intra-process isolation and software compartmentalization. Applications are decomposed into isolated components that are selectively given access to resources. Capability segregation is obtained by a one-bit memory tag for each 256-bit memory cell. The coprocessor includes a set of capability registers; the capability instructions make it possible to access these registers to load and store capabilities, and to dereference capabilities to load and store data.

Annex [6] is a distributed system that unifies addressing and protection by an object model based on password capabilities. In Annex, subjects have no ambient authority. Instead, the entire authority of a subject is that connected with the password capabilities it possesses. This promotes adherence to the *principle of least privilege*: each subject should be granted least possible privileges, and a privilege should be granted to least possible subjects [21]. A password capability consists of a device identifier that univocally specifies a host device, as is required to deliver information requests; an object identifier that specifies an object in the target device; a capability identifier aimed at selecting an access permission; and a password that guarantees capability unforgeability. Password capabilities may only exist within the protective bounds of the kernel. Outside the kernel, a password capability is referenced by using handles mapped to that password capability on a per-object basis. Access privilege revocation uses a kernel-based algorithm, similar to that proposed in [8], in which a graph is used to record the propagation of capabilities within the boundaries of a single device. When a revocation should take place that crosses the device boundaries, all devices are required to cooperate.

Protected pointers [22] were conceived as a solution to the password proliferation problem, consequent to the necessity to specify any arbitrary combination of access rights in a memory system exercising protection at the level of memory segments. A single password is associated with each segment; this keeps the memory requirements for password storage low, and simplifies password management. A segment pointer consists of pair (ID, AR), where ID identifies a segment, and AR specifies a set of access rights for this segment. Segment pointers are never stored in memory in plaintext. Instead, they are stored in the ciphertext form that results from application of a symmetric-key cipher to quantity AR and the password of the corresponding segment. Encryption prevents tampering with the access right field to add new access rights, and is an effective solution to the segregation problem. A segment pointer must be deciphered to verify the access rights when an access is issued to the corresponding segment. Hardware supports are necessary to limit the execution time costs of repeated actions of segment pointer deciphering, e.g. in the case of a sequence of accesses to the same segment. These hardware supports include a set of protection registers inside the processors. A segment pointer must be translated into plaintext and loaded into a protection register to access the corresponding segment. This protection register can be used in every subsequent access to this segment.

The m-key protection paradigm, introduced in this paper, does not rely on any form of support from *ad hoc* hardware. Instead, all the functionalities of the protection system are intended to be implemented by software routines. As seen in Section 4.1, m-keys do not need to be segregated in memory or enciphered, as they are protected from forgery by unguessablef master passwords. We rely on typed object encapsulation for fine-grained process separation (see Section 3.3). M-key revocation requires no intervention of the kernel. In fact, as seen in Section 2.4, our solution to the revocation problem takes advantage of the m-key validation procedure, which uses the universally-known base function. When an access is attempted to a given object to execute a given operation, the master password associated with that object is used to validate the access. If the master password has been changed, validation fails, and the m-key is revoked. Multiple encryption is used to support m-key weakening with no intervention of the protection system (see Section 2.3). Multiple encryption is certainly not a new idea; it has been used to support secure authorized deduplication, for instance [23], [24].

5 CONCLUDING REMARKS

In a protection system supporting active subjects that produce access attempts to protected objects, we have proposed a new key paradigm called multiple keys. The following is a brief summary of the main results we have obtained:

- A subject that holds an m-key referencing a given set of s-objects can transform this m-key into a weaker m-key referencing only a subset of these s-objects. This transformation requires no intervention of the protection system, and can be iterated to the limit of an m-key that references a single s-object.
- If master values are large, sparse and chosen at random, it is virtually impossible for a malevolent subject to forge a valid m-key. Similarly, it is impossible to modify a valid m-key to reference more s-objects.
- By changing the master value associated with a given m-key name, we revoke the validity of the m-keys defined in terms of that m-key name. It is even possible to associate more master values with a single m-key name, and in this case, revocation can be restricted to the m-keys derived from a subset of these master values. We

have shown that this m-key revocation mechanism possesses a number of interesting properties; it is selective, transitive, independent, and temporal.

- The memory requirements for storage of an m-key are low, and are comparable to those of a single key. This is especially true for clusters of strictly related s-objects, so that these s-objects can be named in sequence.
- The m-key concept is flexible, and can be used to solve a variety of protection problems efficiently. We have illustrated a few examples of applications, including the protection of collections of strictly related s-objects, the management of sets of cryptographic keys, and the access permissions to typed objects.

ACKNOWLEDGEMENT

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

REFERENCES

[1] X. Zhang, Y. Li, and D. Nalla, “An attribute-based access matrix model,” in *Proceedings of the 2005 ACM Symposium on Applied Computing*, (Santa Fe, New Mexico, USA), pp. 359–363, ACM, March 2005.

[2] L. Lopriore, “Password management: distribution, review and revocation,” *The Computer Journal*, vol. 58, pp. 2557–2566, October 2015.

[3] G. Saunders, M. Hitchens, and V. Varadharajan, “Role-based access control and the access control matrix,” *ACM SIGOPS Operating Systems Review*, vol. 35, no. 4, pp. 6–20, 2001.

[4] G. Dini and L. Lopriore, “Distributed storage protection in wireless sensor networks,” *Journal of Systems Architecture*, vol. 61, pp. 256–266, May–June 2015.

[5] S. Zhu, S. Xu, S. Setia, and S. Jajodia, “Establishing pairwise keys for secure communication in ad hoc networks: a probabilistic approach,” in *Proceedings of the 11th IEEE International Conference on Network Protocols*, (Atlanta, GA, USA), pp. 326–335, IEEE, November 2003.

[6] 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*, (Miami Beach, Florida, USA), pp. 341–352, IEEE, December 2007.

[7] J. S. Shapiro, J. M. Smith, and D. J. Farber, “EROS: a fast capability system,” *ACM SIGOPS Operating Systems Review*, vol. 34, no. 2, pp. 170–185, 2000.

[8] V. D. Gligor, “Review and revocation of access privileges distributed through capabilities,” *IEEE Transactions on Software Engineering*, vol. SE-5, pp. 575–586, November 1979.

[9] J. Threet and S. Shenoi, “Capability-based primitives for access control in object-oriented systems,” *Database Security XI: Status and Prospects*, pp. 134–148, 2016.

[10] L. Lopriore, “Password capabilities revisited,” *The Computer Journal*, vol. 58, pp. 782–791, April 2015.

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

[12] J. Brown, J. Grossman, A. Huang, and T. F. Knight Jr, "A capability representation with embedded address and nearly-exact object bounds," tech. rep., Project Aries, ARIES-TM-005, Artificial Intelligence Laboratory, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, 2000. http://www.ai.mit.edu/projects/aries/Documents/Memos/ARIES-05.pdf.

[13] R. N. Watson, J. Woodruff, P. G. Neumann, S. W. Moore, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie, S. Murdoch, R. Norton, M. Roe, S. Son, and V. Munraj, "CHERI: a hybrid capability-system architecture for scalable software compartmentalization," in *Proceedings of the 36th IEEE Symposium on Security and Privacy*, (San Jose, California, USA), IEEE, May 2015.

[14] 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*, (Alexandria, Virginia, USA), pp. 29–40, ACM, October 2006.

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

[16] J. S. Chase, H. M. Levy, E. D. Lazowska, and M. Baker-Harvey, "Lightweight shared objects in a 64-bit operating system," in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, (Vancouver, British Columbia, Canada), pp. 397–413, ACM, October 1992.

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

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

[19] M. Robshaw, "One-way function," in *Encyclopedia of Cryptography and Security*, pp. 446–447, Springer, 2005.

[20] P. G. Neumann and R. J. Feiertag, "PSOS revisited," in *Proceedings of the 19th Annual Computer Security Applications Conference*, (Las Vegas, NV, USA), pp. 208–216, IEEE, December 2003.

[21] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, pp. 1278–1308, September 1975.

[22] L. Lopriore, "Encrypted pointers in protection system design," *The Computer Journal*, vol. 55, pp. 497–507, April 2012.

[23] J. Li, X. Chen, M. Li, J. Li, P. P. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1615–1625, 2014.

[24] J. Li, Y. K. Li, X. Chen, P. P. Lee, and W. Lou, "A hybrid cloud approach for secure authorized deduplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1206–1216, 2015.

Lanfranco Lopriore has been a full professor of computer engineering at the University of Pisa, Italy, since 1991. From 1978 to 1990 he was with the Italian National Research Council. In 1990 he was made a full professor of computer engineering at the University of Calabria, Cosenza, Italy. His research interests include protection systems, capability systems, and single address space systems.