On an Authorization Mechanism

RONALD FAGIN
IBM Research Laboratory

Griffiths and Wade (*ACM Trans. Database Syst. 1*, 3, (Sept. 1976), 242–255) have defined a dynamic authorization mechanism that goes beyond the traditional password approach. A database user can grant or revoke privileges (such as to read, insert, or delete) on a file that he has created. Furthermore, he can authorize others to grant these same privileges. The database management system keeps track of a directed graph, emanating from the creator, of granted privileges. The nodes of the graph correspond to users, and the edges (each of which is labeled with a timestamp) correspond to grants. The edges are of two types, corresponding to whether or not the recipient of the grant has been given the option to make further grants of this privilege. Furthermore, for each pair *A*, *B* of nodes, there can be no more than one edge of each type from *A* to *B*. We modify this approach by allowing graphs in which there can be multiple edges of each type from one node to another. We prove correctness (in a certain strong sense) for our modified authorization mechanism. Further, we show by example that under the original mechanism, the system might forbid some user from exercising or granting a privilege that he "should" be allowed to exercise or grant.

Key Words and Phrases: authorization, protection, security, privacy, access control, database, revocation, proof of correctness
CR Categories: 4.30, 4.31, 4.33, 4.34, 4.35, 5.24, 5.32

1. INTRODUCTION

Griffiths and Wade [1] have defined an authorization mechanism (which we will henceforth call the GW mechanism) for granting and revoking privileges. Their procedure has attracted attention [4, 5] as an extension beyond traditional password approaches. This paper can be viewed as an extended corrigendum to the Griffiths-Wade paper. Therefore, this paper focuses primarily on technical issues, and leaves the reader to refer to the earlier paper for a historical perspective and for a discussion of implementation issues.

As an example of a grant in the GW mechanism, user *A* may grant user *B* the right to read a certain file, say file *F*. Optionally, user *A* may make his grant to user *B* with grant option, which means that user *B* can also grant others the right to read file *F*, with or without grant option (as *B* chooses).

If one user revokes a privilege that he gave to a second user, then the second user should not necessarily lose that privilege. This is because the second user may have been granted this privilege "independently" by yet another user. As an example, due to Griffiths and Wade [1], consider the situation in Figure 1, in

Author's address: IBM Research Laboratory K53/282, 5600 Cottle Road, San Jose, CA 95193.

which user A (the creator of file F) grants users B and C the privilege to read file F, with grant option (the numbers written on the edges are timestamps of the grants). Assume that users B and C each grant user D that privilege, and then B revokes the privilege. User D retains the privilege, because of the grant from C.

As a constrast, consider the situation in Figure 2. Here, when B revokes the privilege he granted to D, then D should lose the privilege, even though, as in the previous situation, D has another grant of this privilege from C. This is because the path back to A has been cut.

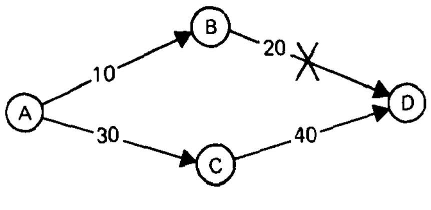

Fig. 1

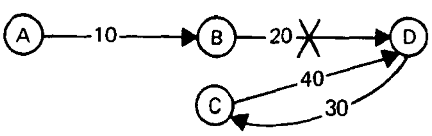

Fig. 2

The GW mechanism maintains an authorization table. For grants of those privileges that have not yet been revoked, the table contains the timestamp of the grant, the i.d. of the grantor, the i.d. of the recipient, a description of the privilege (such as to read file F), and a notation as to whether or not this grant is with grant option. We can think of this authorization table as being a directed graph, where the nodes correspond to users and the edges (each of which is labeled with a timestamp) correspond to grants. The edges are of two types, corresponding to whether or not the grant is with grant option.

We now describe the GW mechanism (and our modification). Assume that at time t, user X grants user Y privilege P over file F, with or without grant option. Let us call this grant G. If X is the creator of file F, then grant G is recorded into the authorization table. If not, but if at time t there appears in the authorization table a grant of privilege P to X, with grant option, then once again grant G is recorded. Otherwise, grant G is invalid, and so it is not recorded (thus, it is ignored, that is, treated as a NO-OP).

There is one exception to the above set of rules for recording grants. The manner in which this exception is handled is the difference between the original GW mechanism and our modified version. Assume that just before time t, there already appears in the authorization table an earlier grant G' of the same privilege P from the same grantor X to the same recipient Y, and with the same grant option as G (that is, grants G and G' are either both with grant option or both without grant option). In the original GW mechanism the new grant G is not recorded [1, p. 249]. In our modified version the new grant G is recorded and the old grant G' is also left recorded. Thus, in this case, just after time t, the authorization table contains at least two grants of the same privilege from the same grantor to the same recipient, with the same grant option (but with different timestamps). We note that grants with grant option and grants without grant option are recorded separately in the original GW mechanism. Therefore, under the original GW mechanism, it is possible that at a given time there is recorded in the grant table two grants of the same privilege, from the same grantor to the same recipient, as long as the two grants have different grant options. By contrast, under the modified GW mechanism, an arbitrary number of grants of each type, from the same grantor to the same recipient of the same privilege, may appear in the authorization table. (A natural scenario in which A might grant the same privilege P twice to B is if, say, he grants privilege P to B, and then, at a later time, he grants privilege P to everyone.)

Our revocation algorithm is the same as that of Griffiths and Wade, except that their rules for deleting a grant by X' to Y' of privilege P from the authorization table may lead us to delete several (but as we will see, not necessarily all) such grants from X' to Y'. Assume that grantor X revokes privilege P from Y. If the authorization table at that time contains no grant of privilege P from X to Y, then the revocation is ignored. Otherwise, on step 1 of the revocation algorithm, each recorded grant of privilege P from X to Y is deleted from the table. If one of the deleted grants from X to Y was with grant option, then what happens to Y's grants of privilege P to others? Let t be the minimum timestamp for which there still appears in the authorization table a grant to Y of privilege P, with grant option (if no such grant still appears, then let $t = \infty$). On step 2, each grant by Y of privilege P, with timestamp smaller than t, is deleted from the table. Grants continue to be deleted from the table recursively by this procedure. For example, consider the situation in Figure 2. Assume that all grants in Figure 2 are with grant option. At time 50, user B revokes privilege P from D. In step 1 of the revocation algorithm, the grant from B to D is deleted from the authorization table. On step 2, the grant from D to C is deleted. On the third (and this case, final) step, the grant from C back to D is deleted.

In Section 3 we prove "correctness" for the modified GW mechanism. To explain what we mean by "correctness," we must first explain some auxiliary concepts.

Assume that $G_1, \ldots, G_n$ are grants of privilege P over file F. We assume that $G_1, \ldots, G_n$ are each timestamped, and each annotated as to whether or not it is with grant option. We say that $\langle G_1, \ldots, G_n \rangle$ is an *authorization chain* of privilege P if

(a) the grantor of G_1 is the creator of file F;
(b) the timestamp of G_{i+1} is larger than the timestamp of G_i ($i = 1, \ldots, n-1$);
(c) the grantor of G_{i+1} equals the recipient of G_i ($i = 1, \ldots, n-1$);
(d) grants $G_1, \ldots, G_{n-1}$ are with grant option.

For example, assume that A is the creator of file F, and that G_1, G_2, G_3, G_4 are, respectively, grants from A to B at time 10, from B to C at time 20, from C to D at time 30, and from D to E at time 40. Assume that each of the four grants is a grant of privilege P, which is to read file F, and that the first three grants are with grant option. Then $\langle G_1, G_2, G_3, G_4 \rangle$ is an authorization chain of privilege P.

Assume that a fixed finite sequence of grants and revocations has taken place (where the grants and revocations may be interleaved). Let GRANT be the set of grants only (each timestamped and annotated as to grant option). Let REVOKED be the set of all grants in GRANT of privileges that are later revoked by users. That is, if in our fixed sequence of grants and revocations, user X revokes privilege P from user Y at time t, then REVOKED contains all grants of privilege P from X to Y with timestamp less than t. Note that REVOKED is *not* defined to contain all grants that are deleted from the authorization table by the recursive revocation procedure; instead, REVOKED contains only those grants of privileges that are explicitly revoked by users. Let UNREVOKED be the set difference GRANT – REVOKED. Thus, UNREVOKED is the set of all grants (in our fixed finite sequence) of privileges that are not later explicitly revoked by users. Define VALID to be the subset of UNREVOKED of all grants G for which there is an authorization chain $\langle G_1, \ldots, G_n \rangle$ of grants in the set UNREVOKED, with $G_n = G$. (It is possible that the chain is of length 1; then the grantor of G is the creator of the file F over which the privilege is being granted.) So a grant is in VALID if it is the final grant in an authorization chain of grants in UNREVOKED. Note that the authorization chain that shows that a given grant is in VALID is not necessarily unique. That is, it is possible for a grant to be the final grant in more than one authorization chain. Intuitively, we think of VALID as the set of grants in our sequence that would be valid if the grants of privileges that were later revoked had never taken place. Hence, VALID is the set of grants that the system "should" honor. We will prove (in Section 3) that if the modified GW mechanism is continually applied (after each grant and revocation in our fixed finite sequence), then the following desirable property holds.

CORRECTNESS PROPERTY. *The final resulting authorization table contains precisely the set VALID of grants.*

This property is, as its name implies, what we mean by "correctness" of the modified GW mechanism.

The actual manner in which the authorization table is utilized in practice is twofold. First, a user (such as user X) is allowed to exercise privilege P over file F if and only if either (1) X is the creater of file F, or else (2) there is an entry in the authorization table in which user X is granted privilege P. If either (1) or (2) holds, then we say that "the system authorizes user X to exercise privilege P." Second, user X is allowed to grant privilege P to others if and only if either (1′) X is the creator of file F, or else (2′) there is an entry in the authorization table in which user X is granted privilege P with grant option. If either (1′) or (2′) holds, then we say that "the system authorizes user X to grant privilege P." It is easy to see that the correctness property above implies the following two properties.

EXERCISABILITY PROPERTY. *The system authorizes user X to exercise privilege P (over file F) if and only if either* (1) *X is the creator of file F, or else* (2) *X is the recipient of a grant that is in the set VALID.*

The exercisability property is desirable since, as we saw, VALID is the set of all grants that the system should honor.

GRANTABILITY PROPERTY. *The system authorizes user X to grant privilege P (over file F) if and only if either* (1′) *X is the creator of file F, or else* (2′) *X is the recipient of a grant, with grant option, that is in the set VALID.*

In Section 2 we show that the original GW mechanism does not obey the correctness property, the exercisability property, or the grantability property.

2. EXAMPLES AND COUNTEREXAMPLES

In this section we show that the original GW mechanism does not perform as it should. Recall that the only difference between the original GW mechanism and our modified version is that if the same privilege from the same grantor to the same recipient, with the same grant option, is granted more than once (without an intervening deletion of the first grant from the authorization table), then the original GW mechanism ignores all but the first such grant (that is, does not record the later grants in the authorization table). However, under the modified GW mechanism, all legal grants are recorded.

We now demonstrate an example of a problem with the original GW mechanism. Assume that user *A* is the creator of file *F*, and that a number of grants, each with grant option, of privilege *P* over file *F* take place as in Figure 3(a). For example, user *A* grants user *B* privilege *P* at time 10, user *B* grants user *C* privilege *P* at time 20, and so on. Note that *C* grants *D* privilege *P* both at time 30 and at time 60. Under the modified GW mechanism, all grants appearing in Figure 3(a) are contained in the authorization table after time 60. However, under the original GW mechanism, the second grant from *C* to *D* is ignored, and so, after time 60, only the grants in Figure 3(b) are contained in the authorization table. Assume now that at time 70, user *B* revokes privilege *P* from user *C*. What is the effect on the authorization table in the original GW mechanism? On step 1 of the revocation algorithm, the grant from *B* to *C* is deleted from the authorization table. Then the earliest remaining grant to *C* has timestamp 40. So on step 2, the revocation algorithm deletes from the authorization table the grant from *C* to *D*, since the timestamp (30) of this latter grant is smaller than 40. Similarly, on the last step, the grant from *D* to *E* is deleted. Thus, under the original GW mechanism, the final resulting authorization table contains the grants in Figure 3(c). In particular the system no longer authorizes user *D* to exercise privilege *P*. However, *D* should be allowed privilege *P*, since there was a grant from *A* to *C* at time 40, and from *C* to *D* at time 60.

Under the modified GW mechanism, after the revocation by *B* at time 70, the revocation algorithm first (on step 1) deletes from the authorization table the grant from *B* to *C*. Then the earliest remaining grant to *C* has timestamp 40. So on step 2, the grant from *C* to *D* with timestamp 30 is deleted (but the grant from *C* to *D* with timestamp 60 is *not* deleted). On the last step, the grant from *D* to *E* is deleted. The final resulting authorization table contains the grants in Figure 3(d). So after time 70, user *D* is authorized to exercise (and to grant) privilege *P*, as he should be, whereas under the original GW mechanism, as we saw, he is not.

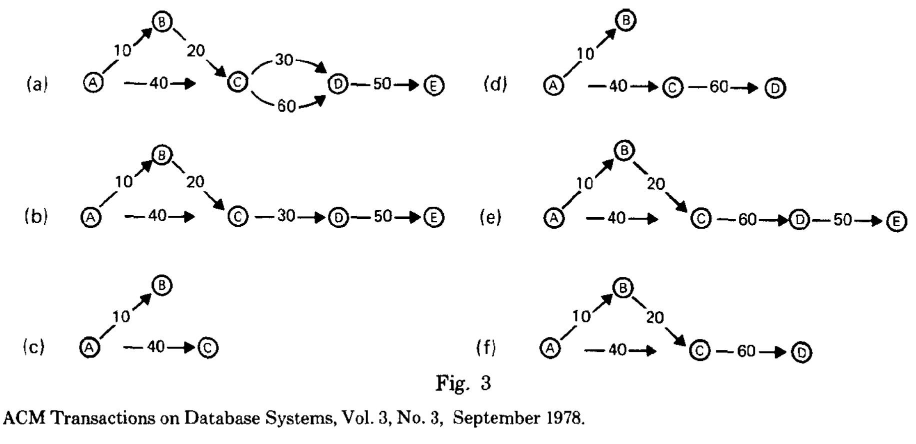

Fig. 3

We have shown that the original GW mechanism violates the correctness property, as defined in Section 1, because the grant from C to D at time 60 should be honored by the system, that is, this grant is in the set VALID. The exercisability and grantability properties are violated, since after time 60, user D should be allowed to exercise and grant privilege P, yet the system forbids this.

What would have happened if the original GW mechanism were modified so that after the grant from C to D at time 60, rather than the second grant being ignored, instead the *first* grant were deleted and the second grant were recorded? Then after time 60, the authorization table would contain the grants in Figure 3(e). It is easy to see that the system can "garbage collect" away the grant from D to E, since the incoming grant (that has timestamp 60) to D has timestamp larger than that of the grant from D to E (that has timestamp 50). Then the authorization table would contain the grants in Figure 3(f). In particular the system would not authorize user E to exercise or grant privilege P. However, user E should be so authorized, because of the grants from A to B at time 10, from B to C at time 20, from C to D at time 30, and from D to E at time 50. In fact under the *modified* GW mechanism, the final authorization table contains the grants in Figure 3(a). In particular the system then *does* authorize user E to exercise and grant privilege P, as it should.

We have shown that if only one grant at a time of the same privilege, from the same grantor to the same recipient, with the same grant option, is contained in the authorization table, then the system may not authorize some user to exercise or grant some privilege that the user should be authorized to exercise or grant.

3. PROOF OF CORRECTNESS OF THE MODIFIED GW MECHANISM

In this section we prove that the correctness property, defined in Section 1, holds for our modified GW mechanism. That is, we show that after a sequence of grants and revocations (where the grants and revocations may be interleaved), the resulting authorization table contains precisely those grants in the set VALID. Recall that a grant G of privilege P over file F from user X to user Y (with or without grant option) is in the set UNREVOKED if grant G appears in our sequence, and if, furthermore, our sequence does not contain a revocation, of privilege P by user X from user Y, that occurred after the time of grant G. The grant is in the set VALID if it is also the last grant in an authorization chain of grants in UNREVOKED. So VALID contains precisely those grants that the system "should" recognize.

We now show that the modified GW mechanism obeys the correctness property. The exercisability and grantability properties of Section 1 then follow, since they are consequences of the correctness property.

THEOREM. *After a sequence of (possibly interleaved) grants and revocations in which the modified GW mechanism is applied after each grant and revocation, the resulting authorization table contains precisely those grants in the set VALID.*

PROOF. Let AUTH be the set of grants in the resulting authorization table. We must show that AUTH = VALID.

We first show that VALID is a subset of AUTH. If not, then of all the grants that are in VALID but not in AUTH, let G be the one with the smallest timestamp (there is at least one such grant by assumption). Since G is in VALID, there is an authorization chain $\langle G_1, \ldots, G_n \rangle$, where each grant in the chain is in UNREVOKED, and where $G_n = G$. Grant G_{n-1} is in VALID, because of the authorization chain $\langle G_1, \ldots, G_{n-1} \rangle$. Now, the timestamp of G_{n-1} is smaller than that of G_n, that is, smaller than that of G. Therefore, since G is the grant with minimal timestamp that is in VALID but not in AUTH, it follows that G_{n-1}, being in VALID, is also in AUTH. Hence, G_{n-1} is never deleted from the authorization table during the execution of the sequence of grants and revocations (because it is in the final version AUTH of the authorization table).

Assume for definiteness that G is a grant of privilege P by user X to user Y, with timestamp t. Now there are only two ways that G can be deleted from the authorization table. The first way is if X revokes privilege P from Y at some time after time t. But this is not the case, since G is in VALID. The only other way that G can be deleted from the authorization table is if at some time after time t, a grant to X of privilege P is deleted from the authorization table and if there is then no remaining grant of privilege P to X in the authorization table with timestamp smaller than t. But this cannot happen either, since, as we saw, grant G_{n-1} is never deleted from the authorization table.

Hence, G is never deleted from the authorization table, and so G is in AUTH. This is a contradiction.

We have shown that VALID is a subset of AUTH. To conclude the proof, we must show that AUTH is a subset of VALID.

Let $H_1, \ldots, H_k$ be a sequence of (possibly interleaved) grants and revocations. We wish to show that AUTH is a subset of VALID (where AUTH and VALID are each based on $H_1, \ldots, H_k$). We call each H_j ($j = 1, \ldots, k$) a *command*; a command is either a grant or a revocation. For $j = 1, \ldots, k$, we denote by AUTH_j the resulting authorization table when the modified GW mechanism has completed its actions after command H_j. Similarly, define UNREVOKED_j and VALID_j to be based on the sequence $H_1, \ldots, H_j$. That is, a grant G is in UNREVOKED_j if G appears in $H_1, \ldots, H_j$, and if $H_1, \ldots, H_j$ does not contain a revocation of the privilege granted in G by the grantor of G from the recipient of G, after the time that G was granted. Furthermore, a grant is in VALID_j if it is the last grant in an authorization chain of grants in UNREVOKED_j.

We wish to show that AUTH_k is a subset of VALID_k. Our proof proceeds by induction on k. To begin the induction, assume that $k = 1$. In this case both AUTH_1 and VALID_1 are empty (and hence equal) unless (a) H_1 is a grant, and (b) the grantor of H_1 is the creator of the file over which the privilege is granted. If (a) and (b) both hold, then AUTH_1 and VALID_1 both contain precisely H_1, and so once again, AUTH_1 and VALID_1 are equal. This concludes the proof in the $k = 1$ case.

Assume inductively that AUTH_{k-1} is a subset of VALID_{k-1}. We wish to show that AUTH_k is a subset of VALID_k.

Since AUTH_{k-1} is a subset of VALID_{k-1}, and since (by the first part of the proof) VALID_{k-1} is a subset of AUTH_{k-1}, it follows that $\text{AUTH}_{k-1} = \text{VALID}_{k-1}$.

It is possible that command H_k is either a grant or a revocation. We assume first that H_k is a grant. For definiteness, assume that H_k is a grant by user X to user Y of privilege P over file F. Recall that by definition of the modified GW mechanism, grant H_k is in AUTH_k, that is, H_k is recorded into the authorization table, if and only if either

(1) X is the creator of file F, or
(2) AUTH_{k-1} contains a grant to X of privilege P, with grant option.

It is easy to see that because H_k is a grant and not a revocation, it follows that VALID_{k-1} is a subset (not necessarily proper) of VALID_k. Furthermore, we know that $\text{AUTH}_{k-1} = \text{VALID}_{k-1}$. So AUTH_{k-1} is a subset (not necessarily proper) of VALID_k. Now our goal is to show that AUTH_k is a subset of VALID_k, and we know that AUTH_{k-1} is a subset of VALID_k. But the only possible difference between AUTH_{k-1} and AUTH_k is that AUTH_k might contain H_k, while AUTH_{k-1} does not. So we need only show that if H_k is in AUTH_k, then also H_k is in VALID_k. Therefore, let us assume that H_k is in AUTH_k; we will show that H_k is in VALID_k. Since H_k is in AUTH_k, we know that either (1) or (2) above holds. If (1) holds, that is, if the grantor of H_k is the creator of file F, then H_k is certainly in VALID_k. If (2) holds, then let G be a grant, contained in AUTH_{k-1}, of privilege P to user X, with grant option. Since $\text{AUTH}_{k-1} = \text{VALID}_{k-1}$, it follows that G is in VALID_{k-1}. So there is an authorization chain $\langle G_1, \ldots, G_n \rangle$ of grants in UNREVOKED_{k-1}, where $G_n = G$. Now UNREVOKED_{k-1} is a subset of UNREVOKED_k, since H_k is a grant. So each of the grants $G_1, \ldots, G_n$ is in UNREVOKED_k. Furthermore, it is clear that H_k is in UNREVOKED_k, since H_k is the last command in the sequence $H_1, \ldots, H_k$. But then $\langle G_1, \ldots, G_n, H_k \rangle$ is an authorization chain. Therefore, H_k is in VALID_k, as desired.

Now we assume that the final command H_k is a revocation. Again, we wish to show that AUTH_k is a subset of VALID_k. We assume not, and derive a contradiction. Since AUTH_k is not a subset of VALID_k, let G be the grant with minimal timestamp that is in AUTH_k but not in VALID_k. It is clear that AUTH_k is a subset of AUTH_{k-1}, since H_k is not a grant (it is a revocation). In particular, grant G, being in AUTH_k, is also in AUTH_{k-1}. Since $\text{AUTH}_{k-1} = \text{VALID}_{k-1}$, it follows that grant G is in VALID_{k-1}.

For definiteness, let us assume that this grant G is a grant from user X to Y of privilege P over file F with timestamp t.

Since G is in VALID_{k-1}, there is an authorization chain $\langle G_1, \ldots, G_n \rangle$ of grants in UNREVOKED_{k-1}, where $G_n = G$. Assume first that length n of this authorization chain is 1. Then the grantor X of G is the creator of file F. However, since grant G is in VALID_{k-1} but not in VALID_k, we know that revocation H_k must be a revocation by X from Y of privilege P (because X is the creator of file F). But then G is not in AUTH_k, which is a contradiction.

Therefore, the length n of the authorization chain is at least 2. Now G_{n-1} is in VALID_{k-1}, since it is part of an authorization chain of grants in UNREVOKED_{k-1}. Since $\text{AUTH}_{k-1} = \text{VALID}_{k-1}$, it follows that G_{n-1}, being in VALID_{k-1}, is in AUTH_{k-1}. Let us call each grant to X of privilege P, with grant option, with timestamp smaller than t (the timestamp of G), a *supporting* grant for G. Intuitively, these grants "provide support" for G, that is, they are grants of privilege P, with grant option, where the recipient is the grantor of G (and where these grants have timestamps smaller than that of G). We know that AUTH_{k-1} contains a supporting grant for G, namely G_{n-1}. We now show that AUTH_k

contains a supporting grant for G. Assume not. Then the revocation H_k must have caused the revocation algorithm to delete G_{n-1} and all other supporting grants for G. When the last supporting grant for G was deleted from the authorization table, then G should have been deleted, according to the rules of the revocation algorithm. But G was not deleted. Hence, our assumption that AUTH$_k$ does not contain a supporting grant for G is false. Therefore, let G' be a grant that is in AUTH$_k$ and that is a supporting grant for G. Now the timestamp of G' is smaller than that of G, and G' is in AUTH$_k$. Therefore, since G is the grant with minimal timestamp that is in AUTH$_k$ but not in VALID$_k$, we know that G' is in VALID$_k$. So there is an authorization chain $\langle G_1', \ldots, G_m' \rangle$ of grants in UNREVOKED$_k$, with $G_m' = G'$. Now the grantor X of G did not revoke the privilege P from the recipient Y of G after the time t of the grant G, since G is in AUTH$_k$. Therefore, G is in UNREVOKED$_k$. So $\langle G_1', \ldots, G_m', G \rangle$ is an authorization chain of grants in UNREVOKED$_k$. Therefore, G is in VALID$_k$. This is a contradiction.

4. HISTORICAL COMMENTS

We refer the reader to Griffiths and Wade [1] for a historical perspective of their authorization scheme, and for a good bibliography of papers on protection. We note that their scheme differs fundamentally from the recent schemes analyzed by Harrison, Ruzzo, and Ullman [2] and by Lipton and Snyder [3], because of the major role of timestamps in the GW mechanism (there are no timestamps in the other schemes).

This paper originated when the author tried to prove correctness for the (original) GW mechanism. Thus, an attempted proof of correctness led to the discovery of a "bug"!

5. SUMMARY

We have modified the authorization mechanism of Griffiths and Wade by allowing the authorization table to contain simultaneously more than one grant from the same grantor to the same recipient of the same privilege with the same grant option. We have proven correctness for the modified authorization mechanism, whereas we have shown that under the original authorization mechanism the system may not authorize some user to exercise or grant some privilege that he should be authorized to exercise or grant.

ACKNOWLEDGMENTS

The author is grateful to Pat Griffiths and Brad Wade for helpful discussions and suggestions, and to Peter Denning for a historical perspective.

REFERENCES

1. GRIFFITHS, P.P., AND WADE, B.W. An authorization mechanism for a relational database system. *ACM Trans. Database Syst. 1*, 3 (Sept. 1976), 242-255.
2. HARRISON, M.A., RUZZO, W.L., AND ULLMAN, J.D. Protection in operating systems. *Comm. ACM 19*, 8 (Aug. 1976), 461-471.

3. LIPTON, R.J., AND SNYDER, L. A linear time algorithm for deciding subject security. *J. ACM 24*, 3 (July 1977), 455–464.
4. TAYLOR, A. Another security approach steps beyond passwords. *Computerworld 10*, 49, Dec. 6, 1976, p. 13.
5. VOYSEY, H. San Jose: Home of System R. *Computing Europe*, Oct. 28, 1976, p. 8.

Received May 1977; revised February 1978