CMU-ITC-84-016

Authentication Server Reference

David King

Information Technology Center

1. Overview

This document describes the Authentication Server for the VICE system, providing user authentication, key distribution and usage accounting functions for the file system and other system services. A separate document describes the internal organization and functioning of the authentication programs, and another describes the authentication message algorithms in greater detail.

Authentication, in the context of a complex of distributed services provided by VICE, is the mechanism by which servers identify the users of the workstations making use of their services, establishing their rights to the services they request; these rights depend on the personal identity of the workstation user. It starts when the user workstation VIRTUE software makes a network connection to the Authentication Server in the nearest cluster machine and a call to the *Connect* service, and is ended by a call to the *Disconnect* service. In between the VIRTUE software makes many requests for service, for instance, *Fetch* and *Store* calls to the file server, or *Attach* calls to the printer server, each of which provides a service; in each case, the service-providing server will ask the Authentication Server for the encryption key which VIRTUE will use.

Accounting, as a function of the authentication system, is the mechanism used by the system administration to distribute the available network computing resources among the potential users, and to maintain records of the resources used by them. It involves computing resources allocated to users, expended by them through the use of system services, and accounted for in a usage transaction file.

The Authentication Server will be implemented as a set of processes running in each cluster machine, one of which is designated the Central Authentication Server for accounting data base purposes. There will be some network-wide data bases, used by all of the servers and maintained by the Central Server. There will be some other cluster-specific data bases, used and maintained independently by each of the servers.

1.1. Definitions

cluster machine	A machine running the VICE software, generally considered to be the primary system for a number of VIRTUE workstations. In it run the file system software processes, the printer server process, perhaps a bulletin board server process, and the Authentication Server processes.
file server	The VICE file system software running in a cluster machine.
cluster server	This is an ambiguous expression and should not be used in this document.
job number	A number, unique within a single cluster machine, to distinguish the workstations authenticated to that machine. This is necessary to allow multiple jobs for a single user to be recognized.
username	The computer-oriented user identification, as assigned by the central administration. These are presently four-character "man numbers," consisting of the users' two initials followed by two more characters for uniqueness; my username is 'DK32'. If another username policy is proposed, it should have usernames short enough to be conveniently specified by users but long enough to be unique over the lifetime of the username. Throughout this design they are limited to eight characters, which is enough for any reasonable username.
account number	A computer-oriented code identification for a computer usage account. At present account numbers are such things as "R602" which has the meaning "Computation Center Systems Software." Throughout this design they are limited to four characters.
erg	A unit of work; the work expended in applying a force of one dyne over a distance of one centimeter. This is an experimental term to denote a unit of computer network utilization, in an attempt to use a non-money term to keep money and computing usage distinguished.
commodity	A service provided by the system which is to be accounted for.
commodity code	A number assigned by the system administration to each service provided by the system; e.g. "disk storage," "network packets," "printer #2 pages," "printer #5 characters."

1.2. Authentication

Before using any servers on a particular cluster machine, each user workstation will make a secure RPC connection to the Authentication Server on that machine and use the *Connect* service to identify itself to VICE. It must provide as an argument to the *RPC_Bind* call a block containing the user's username, and an encryption key generated by "crunching" the user's password (see "key crunching" in the appendix).

The Authentication Server will receive the connection and be asked for matching key; it will look up the user's crunched password in the User Authorization File. It will also generate a session key for use in messages after this exchange. After this the Remote Procedure Call library will encrypt and sequence-check all messages between the workstation and the Authentication Server. The data provided in this initial message will include the user's username again, the account number, the password, and other information.

The Server will check the password and account, and internally register the workstation as "logged in" as that user, with that encryption key. It will return a unique "job number" to the user.

When a workstation wishes to use a particular VICE service, the workstation will establish an authenticated connection to it, providing the username, the job number, and the assigned session key. The server will receive the connection and will ask the Authentication Server for the session key assigned to that user. After that the workstation and server can authenticate and encrypt whatever messages they wish.

When a workstation wishes to stop using the services of a cluster machine, it should call that Authentication Server's *Disconnect* service. It will be charged for cluster machine service until it does.

1.3. Accounting

The central administration will allocate resources to users by creating computer usage accounts and allocating various amounts of computer usage to the accounts; account administrators will give allocations to their users. The Authentication Server system will keep track of the usage allocation remaining to each user, as workstations logged in as that user make make use of services on a session-to-session basis. As servers receive workstation requests, they will tell their Authentication Server what sort of service was provided and "how many" of that service; the Authentication Server will deduct the proper number of "ergs" from the user's allocation. When the user logs out from the network, these deductions eventually make their way to a central Authentication Server which applies the deduction to a central data base. If the user has exceeded his allocation at this point, the several Authentication Servers are told not to accept *Connect* requests for that user. It would be possible to have a much more complex system which would keep careful track of deductions and cut off service the instant the user reached his limit; this design does not do this.

This system will allow system usage to be rationed as the administration wishes. All usage transactions will be saved in a file, which can be processed afterward to provide usage distribution statistics. It might be desirable to occasionally inform the user, by mail, of his activity (say over the last month) and remind him of his balance. All of this is outside the scope of this system.

1.4. Assumptions

This design assumes that the Authentication Server processes can make ordinary file references, in the traditional Unix manner, to certain system files resident on the local cluster machine, and of concern only to the Authentication Server.

It is assumed that workstation and system server processes can communicate with the Authentication Server using a Remote Procedure Call (RPC) mechanism. Connections between system servers and the Authentication Server, all running on the same machine, are assumed to be secure against user interference or observation.

Some improvements may appear in the network communications area over the life of the project:

- Unlike the standard Ethernet system, a process sending a message will eventually need not have a complete network address for the server with which it wishes to, although it may specify a complete address if it wishes. It can say "send this message to the service-17 socket on whatever network host wants to receive it"; such a generic connection will be answered by one cluster machine, probably the nearest one to the connection initiator.

- There may be a special "high-security" mode available for network connections, which automatically guarantees to both parties that all messages received are from the other party, and that no other party can eavesdrop on the conversation.

- There may be high-speed encryption hardware available on workstations and cluster machines.

- There may be a Unix network domain developed, to allow communication between processing running on the same machine. This would be useful for system servers talking with the Authentication Server.

- There may be a secure network ring, connecting together the cluster machines only; this would make secure communication between the Authentication Servers easier.

The current design takes none of these things for granted; as more advanced features become available, they can be integrated into the server design.

2. Unresolved Issues

This design assumes that a user has only one "computing usage" allocation per account. In the long run this simplifying assumption may not be adequate for a proper rationing of scarce resources; however, at this point the dimensions along which the overall network resource might be divided are unclear. It should be simple enough to break up the allocation into categories, such as "disk access" and "printing," as the system matures.

This system takes for granted a strict separation between "system money" and real money. If a stronger linkage between system usage allocations and real money is required, several parts of the design will be inadequate, particular in security and accountability.

Cutting off service when a user runs out of money has in the past been a politically sensitive issue; there are no users who wish this to happen, but occasionally some departmental business managers have complained about their users using more money than they were budgeted for. This design assumes that the ITC management wishes to actually cut off users.

There is no provision here for a "maximum usage rate" once proposed, settable by the user, to tell the system the fastest he wishes to expend ergs.

It will probably be unacceptable to charge all disk usage to one account; if user files could have account attributes, they could be charged to different accounts.

This design does not provide a general-purpose user-to-user accounting facility. One has been proposed, and is awaiting discussion.

3. Externals

In this section I will describe those parts of the authentication system which will be used by user workstations and system servers, giving enough detail for a program writer to be able to use the system. A later section will describe how the server's services will work.

In the messages and other formats described, int means a 32-bit integer, char means an 8-bit byte, and char[] means a sequence of 8-bit bytes, padded with nulls after the last significant byte if a variable-length string is involved. In integers, there should be an understanding that, when transmitting a 16-bit integer to a network, the first byte transmitted is the high order byte, followed by the lower order byte; for a 32-bit integer, the order should be highest (2^24) byte, next highest (2^16), next-to-lowest (2^8), and lowest. (This is a natural arrangement for the M68000.) Other arbitrary constants, such as service code numbers, will be denoted by upper-case names.

All of the arbitrary numbers which a user or system server need to use are defined in a header file, "authuser.h." This file also contains structure definitions for all of the messages which a user need send or receive. In the rest of the document, message and object formats are described in structure form; in all cases, the definition in the header file is the standard.

3.1. Data bases and blocks

3.1.1. User Authorization File

There will be a User Authorization File (UAF), with a copy on every cluster machine, in a system directory; it will be read and written only by Authentication facilities. They should be everywhere identical, but will be updated by each individual Authentication Server, through a maintenance function. (Or, ultimately, it might be a "replicated" file maintained by the file system, so that a copy modified at a central site would automatically move throughout the network.) It will contain such things as the user's username, crunched password, and valid accounts.

3.2. Workstation and Server Access

To make calls on Authentication Server services, the user workstation software, or the system server, must use the Remote Procedure Call mechanism to establish a connection to server "authrpc" on the desired host, make remote procedure calls over the connection, and close the connection when the communication is complete.

Workstations should use secure connections. In the initial connect for *Connect*, the RPC "self-identification" should contain just the user's username;.

on the initial connect for other purposes, the self-identification should contain the username and the assigned job number. (See below for details.)

The request headers of the messages contain in the "Opcode" field a number to indicate the service being requested; any arguments are contained in the "Parameters" area of the RPC request block. Response messages will contain a status code in the "ReturnCode" field of the response header, with additional information in the "Results" area of the RPC response block.

Messages sent between workstations and cluster machines are generally protected by encryption and by message sequence numbers, by the RPC facility.

Each request a workstation or server can make consists of a set sequence of message exchanges, where the caller sends a request and the server sends a response. When the messages used for a particular service have been exchanged (this is one exchange for everything besides Connect) each side will close the RPC connection.

Workstations and system servers are distinguished in the services which they can request. To designate itself as a system server, the server will (eventually) be required to make an RPC to a "Privilege" service, supplying a password, unless some means can be found for processes in the same machine with the Authentication Server to otherwise identify themselves (Unix domain sockets would serve this purpose).

3.3. Workstation services

3.3.1. Connect service

The user workstation VIRTUE software will first obtain from the user his computer username, his password, and his billing account number. It will convert the user's password with one of the password crunching algorithms (the algorithms are described in an appendix). It will establish a secure RPC connection to the nearest cluster machine's Authentication Server, and send an AAS_CONNECT request, providing as arguments the account number and the password.

```
struct ms_connect {
    char account[4];          /* The account number, or zero to default */
    char password[16];        /* The password, in the clear */
};
```

Additionally, the *RPC_Bind* will need the user's username as the 'self-identification':

```
    struct ms_connect_si {
        char username[8];            /* The username */
    };
```

The server will look up the user's entry in the User Authorization File; if it does not exist it will return with an RPC error. Otherwise, it will extract the crunched password stored there, and tell RPC to use that for the encryption key for the data of this message. It will pick an encryption key suitable for RPC, and return it to the RPC package; this key will be used for all further communications over this connection, and indeed for all future connections. It will then read the message body, and check the username again. It will compare the password with the one stored in the UAF, and check the account number against the list of accounts valid for the user (obtaining the default if necessary). It will internally register the user as logged in, with his job number, account number, and encryption key. It can then return code Success.

The reply block will contain

```
    struct ms_connect_r {
        int job;                     /* The job number */
        int key[8];                  /* The session key */
    };
```

The possible return codes are

AAR_SUCCESS	Success.
AAR_BADMSG	The message was badly formatted.
AAR_BADPASS	The password was incorrect.
AAR_BADDACT	The account was defaulted but there is no default.
AAR_BADACCT	The account was not valid.
AAR_BADALLOC	The account has exceeded its allocation.

Once this is done, the workstation knows his encryption key, and can close the connection to the Authentication Server.

3.3.2. Disconnect service

To disconnect from VICE, the VIRTUE software should establish a secure connection to the Authentication Server again, providing to RPC the 'self-identification' of the user's username and job number, and the session encryption key. It should then request the *Disconnect* service, making an RPC AAS_DISCONNECT request, providing no parameters.

The self-identification block this time should look like

```
struct ms_service_si {
    char username[8];            /* The username */
    int job;                     /* The job number */
};
```

The server will perform various cleanup functions and return a confirmation with return code Success, and no data.

The workstation should have first used *DisconnectFS* to disconnect from any file servers which it may have been using.

3.3.3. Change Authorization service (CHGAUTH)

A user can call Change Authorization, operation code AAS_CHGAUTH, to have the authentication system change some things stored in his User Authorization File entry. It must have established a secure connection in the same manner as for *Disconnect*. The request block contains basic identification information, and the request details.

```
struct ms_chgauth {
    char username[8];            /* The username again */
    char oldpwd[16];             /* The user's current password */
    int item;                    /* The function (CHGAPWD,CHGAACT) */
    char newpwd[16];             /* The new password */
    char newdacct[4];            /* The new default account */
    int checksum;                /* Checksum */
};
```

The function code is a number indicating the thing to be changed.

AAO_CHGAPWD	Change the password.
AAO_CHGAACT	Change the default account.

The server will check the checksum and sequence numbers. The server will check the old password; if it is correct, it will make the change and return a confirmation with no data. The return code will contain

AAR_SUCCESS	The change has been made.
AAR_BADUSER	The username was invalid.
AAR_BADNPASS	The new password is unacceptable (criteria to be determined, currently it must be at least 6 characters long).
AAR_BADACCT	The new default account is not among the allowed accounts.
AAR_BADFUNC	The function code was not recognized.
AAR_BADPASS	The old password is incorrect.

3.3.4. Assign Key

Two users can use this facility to establish secure and authenticated communications, using the protocol of Bauer, Berson, and Feiertag (/fIA Key Distribution Protocol Using Event Markers, ACM TOCS 1:249). In this protocol, user A tells user B his identity and an *event marker*, a never-reused unpredictable number generated secretly by A (a recommended algoritm is to step a counter by one, and pass it through DES to generate the next event marker). User B asks the Authentication Server to create a *conversation key* for this one interaction, providing A's identity and event marker, B's identity, and another event marker for B. The server returns to B a message with two parts, the first encrypted with A's session key and the second with B's session key (these are the keys assigned when the user logged in to his Authentication Server). B's message portion contains the conversation key, A's identity for crosschecking, and B's event marker. B is responsible for passing A's message portion, containing complementary information, to A. With these messages exchanged, the two users can now be sure that they can communicate privately, and that the other partner knows the key only if he is the user he purports to be. (See the article for argumentation on this.)

A's message portion will be encrypted with the Authentication Server's own encryption algorithm, as described in the appendix. The encryption facility of RPC cannot be used for this, although the entire message returned to B is encrypted for him.

Since there are multiple Authentication Servers, there is the complications that user A must also provide the name of the Server to which he is logged in, and that B must pass this information on to B's own Server. With this information B's Authentication Server can contact A's Server and get the session key. If A and B have some Authentication Server in common, it would be more efficient to use it, but that is an efficiency problem for them to consider.

User B obtains this conversation key from his Authentication Server by opening a connection to the Server and making the AAS_ASSIGNKEY request, with arguments

```
struct ms_assignkey {
    char hisname[8];              /* Partner's username */
    int hisjob;                   /* Partner's job number */
    char myevent[8];              /* Requesting user's event marker */
    char hisevent[8];             /* Partner's event marker */
    char hisserver[20];           /* Partner's home cluster, or empty */
};
```

The server will look up the users, requesting information from the partner's home Authentication Server if necessary, assign the conversation key, and return the message

```
struct ms_assignkey_r {
    struct ms_assignkey_rd {      /* Encrypted block */
        char otheruser[8];        /* The other user's username */
        int otherjob;             /* The other user's job number */
        char myevent[8];          /* This user's event marker */
        char key[8];              /* Conversation key */
        int dummy;                /* Space for DES, must be zero */
    } mydata, hisdata;            /* One for each user */
};
```

The server may return codes

AAR_SUCCESS	Success.
AAR_BADMSG	The message was incomprehensible.
AAR_BADUSER	Your username or job number were not valid.
AAR_NOPART	The other user's username or job number were not valid, or for other reasons it was impossible to get information from the other user's Authentication Server.

3.4. System services

These are always requested by processes running on the same system as the responding Authentication Server. Accordingly, integers in the following messages need not be forced to the network standard, but should be in the format appropriate to the machine.

3.4.1. Get User Data service (GETUSER)

When a user workstation requests a service from a VICE server, it will establish an authentication connection, and provide the server with the user's self-identification, presumably the user's username and job number. The server will need to know the user's encryption key. To do this the server will make an RPC call to the Authentication Server's AAS_GETUSER service, providing as parameters the username and job number.

```
        struct ms_getuser {
            char username[8];          /* The username */
            int job;                   /* The job number */
        };
```

The Authentication Server will look the username and job number up in its local database, and if found will return to the server the user's account number and the user's encryption key.

```
        struct ms_getuser_r {
            char account[4];           /* The account number */
            char key[8];               /* The encryption key */
        };
```

The possible return codes are

AAR_SUCCESS	The user is logged in, the reply data contains details
AAR_BADMSG	The message was incomprehensible.
AAR_BADUSER	The user is not logged in.

Given this information, the server can tell RPC the key to use to decrypt the user's messages, and assure itself that the sender is the one to whom that encryption key was assigned. It can then use encryption as it wishes for whatever portions of its communications that require it.

3.4.2. Bill User service

After performing a service, a system server make a request with code AAS_BILLUSER to charge the user, providing the user's username and account number, and a list of commodities and usage counts.

```
struct ms_billuser {
    struct ms_billuser_h {          /* Fixed header */
        char username[8];           /* The username */
        char account[4];            /* The account number */
        char count;                 /* The number of commodities which follow */
    } header;
    struct ms_billuser_c {          /* Commodity block, repeated as needed */
        int comcode;                /* Commodity code */
        int comcount;               /* Number used */
    } commodity;
};
```

The server will compute the total usage from the commodities given, write a transaction record to a transaction file, save in a buffer the total charge against the user's account, and return a confirmation to the sender. The return codes are

AAR_SUCCESS	The usage has been charged against the user's allocation.
AAR_BADCOMM	A commodity code was provided which is not defined; all other commodities were billed.
AAR_BADMSG	The message was incomprehensible.

To reduce overhead, high-traffic servers such as the file server may be able to maintain a small number of counters, perhaps of requests made, or files transferred, or blocks transferred, and send it to the Authentication Server in batches.

3.4.3. User Maintenance service

It is to be used by maintenance programs to update the User Authorization File and Accounting Data Base. It would be good if it could be guaranteed that the sender were suitably privileged. The request has code AAS_UMAINT and parameters

```
struct ms_umaint {
    struct ms_umaint_h {         /* Fixed header */
        int function;            /* Function: ADDUSER,DELUSER,CHGAPWD,
                                    CHGAACT,ADDACCT,CHGACCT,DELACCT */
        char username[8];        /* Username to add or delete */
        char password[16];       /* New password, in the clear */
        int pwdtime;             /* Password change timestamp or zero */
        char defacct[4];         /* New default account */
        char flags;              /* Option flags */
#define UMAINT_CPWD 1            /* Password is already crunched */
        char acctcnt;            /* Number of account blocks, for ADDUSER,
                                    ADDACCT,CHGACCT,DELACCT */
    } head;
    struct ms_umaint_a {         /* Account block, repeat as necessary */
        char account[4];         /* Account number */
        int quota;               /* Account allocation */
        char flag;               /* Account flag */
    } acctblk;
};
```

The function field contains the operation to perform:

AAO_ADDUSER	Add the user to the UAF.
AAO_DELUSER	Remove the user from the UAF.
AAO_CHGAPWD	Change the user's password.
AAO_CHGAACT	Change the user's default account.
AAO_ADDACCT	Add the specified list of accounts, quotas, and flags to the user's list.
AAO_CHGACCT	Change the account quotas and flags for the user's accounts specified in the list.
AAO_DELACCT	Delete the specified accounts.

The server will perform the function and return a confirmation with return code

AAR_SUCCESS	Success.
AAR_BADFUNC	The function requested does not exist.
AAR_DUPUSER	The user to be added already exists.
AAR_BADUSER	The user to be changed or deleted does not exist.
AAR_DUPACCT	The account to be added is already defined.

3.4.4. Server Status service

This is to be used by maintenance programs to obtain information on the Authentication Server's status. At present it only returns a list of the users logged in to the server. The request has code AAS_SSTATUS and no parameters, and returns the information

```
    struct ms_sstatus_r {
        struct ms_sstatus_rh {          /* Fixed header */
            int count;                  /* Number of user entries to follow */
        } head;
        struct ms_sstatus_ru {          /* User entry */
            int job;                    /* Job number */
            char username[8];           /* Username */
            char account[4];            /* Account */
            int logtime;                /* Time of login */
        } user;
    };
```

4. Algorithms

4.0.1. Password crunching

This is the way in which passwords are converted into encryption keys. Workstations must know this in order to do the initial handshaking.

Password crunching algorithm zero takes the 16-byte password in 8-byte chunks and passes the 64-bit number through the DES algorithm with a standard key, with chaining, and the last 64 bits resulting from this are taken as the key. (This process courtesy of Konheim, Mack, McNeill, Tuckerman, and Waldbaum, *The IPS cryptographic programs*, IBM Sys. J., 19:253, 1980.)

4.0.2. Data encryption

Data will be encrypted using a corrected form of the Unix crypt(3) implementation of the Data Encryption Standard, using block chaining.