Authentication Routines [E]

This appendix contains detailed information about the data that is passed during UNICOS, Data Encryption Standard (DES), and Kerberos authentication and validation.

UNICOS authentication
E.1

The client of a remote procedure might want to identify itself as it is identified on a UNICOS system. In this case, the value of the credential's discriminant of an RPC message is AUTH_UNIX. The bytes of the credential's string encode the following (XDR) structure:

```
struct auth_unix {
    unsigned    stamp;
    string      machinename[255];
    unsigned    uid;
    unsigned    gid;
    unsigned    gids[10];
};
```

The stamp value is an arbitrary ID that the caller machine can generate. machinename is the name of the caller's machine (such as krypton). uid is the caller's effective user ID. gid is the caller's effective group ID. gids is a counted array of groups that contain the caller as a member. The verifier that accompanies the credentials should be AUTH_NULL.

The value of the discriminate of the response verifier received in the reply message from the server can be AUTH_NULL or AUTH_SHORT. In the case of AUTH_SHORT, the bytes of the response verifier's string encode an auth_opaque structure. This new auth_opaque structure can be passed to the server instead of the original AUTH_UNIX credentials. The server keeps a cache that maps shorthand auth_opaque structures (passed back by way of an AUTH_SHORT-style response verifier) to the original credentials of the caller. The caller can save network bandwidth and server CPU cycles by using the new credentials.

The server can flush the shorthand auth_opaque structure at any time. If this happens, the RPC message is rejected because of an authentication error. The reason for the failure is AUTH_REJECTEDCRED. At this point, you might want to try the original AUTH_UNIX credentials.

DES authentication
E.2

When you use DES authentication, the authentication and validation data exchanged with each RPC call and reply become markedly more complex. When a client chooses to use DES authentication, it must make a call to authdes_create to build a DES authentication structure. The DES authentication structure has the following form:

```
typedef struct {
        struct  opaque_auth     ah_cred;
        struct  opaque_auth     ah_verf;
        union   des_block       ah_key;
        struct auth_ops {
                void    (*ah_nextverf)();
                int     (*ah_marshal)();        /* nextverf & serialize */
                int     (*ah_validate)();       /* validate verifier */
                int     (*ah_refresh)();        /* refresh credentials */
                void    (*ah_destroy)();        /* destroy this structure
*/
        } *ah_ops;
        caddr_t ah_private;
} AUTH;
```

The first field in the AUTH structure is ah_cred. This is the authentication handle credential field; it contains the information the client will provide the server for authentication. The second field in the AUTH structure is ah_verf. This is the authentication handle verification field; it contains the information the server will return to the client to prove its identity. Both of these fields are of type opaque_auth. An opaque_auth structure has the following form:

```
struct opaque_auth {
        enum_t  oa_flavor;              /* flavor of auth */
        caddr_t oa_base;                /* address of more auth stuff */
        u_int   oa_length;              /* not to exceed MAX_AUTH_BYTES */
};
```

The oa_flavor field specifies the type of authentication or validation being done. It can take the value AUTH_NONE, AUTH_UNIX, AUTH_SHORT, or AUTH_DES.

The oa_base field is a pointer to specific data being used for authentication or validation. In the case of AUTH_DES, the ah_cred.oa_base field is unused, but the ah_verf.oa_base field points to an area that contains an encrypted time stamp filled in by the server and checked by the client.

The oa_length field specifies the number of data bytes to which the oa_base field points.

The third field of the AUTH structure is called ah_key, the authentication handle key. This field contains a DES key used for the duration of the AUTH structure. The ah_key field is of type union des_block, which is specified as follows:

```
union des_block {
        struct {
#ifdef CRAY
                word64 both;
#else
                u_long high;
                u_long low;
#endif
        } key;
        char c[8];
};
typedef union des_block des_block;
```

The des_block is a union of 8 bytes, which constitute the DES session key. This session key exists only for the duration of the AUTH structure, which is no longer than the duration of the CLIENT structure with which this AUTH structure is associated. Typically, a new CLIENT structure is generated each time the client-side application is executed. Thus, a new DES key is generated each time the application is run. This DES session key is never sent across the network in its plain form. Instead, it is sent only after it has been encrypted as part of the ah_private data, described below.

The des_block union contains a conditional compilation statement. This is necessary because, on a Cray Research system, a long value consists of 64 bits. On most other machines that run secure RPC, a long value consists of only 32 bits. Thus, to maintain consistency, the key portion of the union is declared a structure that contains one element of type word64 on the Cray Research system. The word64 type is defined in the <rpc/types.h> file, and it is merely a 64-bit entity that allows the user to address the high or low 32 bits of it separately.

The ah_key field of the AUTH structure is currently used only when performing DES authentication and validation. For other types of authentication, its contents are undefined.

The next field in the AUTH structure is the ah_ops field, which is a pointer to a structure that contains function pointers specific to the authentication method. The functions pointed to are enumerated in the AUTH structure and include functions to get the next verifier, to marshal (generate) credentials, to validate credentials, to refresh credentials, and to destroy credentials.

The last field in the AUTH structure is the ah_private field, which is a generic pointer to data specific to the authentication method. In the case of DES authentication, the ah_private field points to a structure of type ad_private. Users should not manipulate the data within this structure. The contents of the structure are described as follows; this description is only for informational purposes.

```
/*
 * This struct is pointed to by the ah_private field of an “AUTH *”
 * when doing DES authentication.   */
struct ad_private {
        char *ad_fullname;              /* client’s full name */
        u_int ad_fullnamelen;           /* length of name, rounded up */
        char *ad_servername;            /* server’s full name */
        u_int ad_servernamelen;         /* length of name, rounded up */
        u_int ad_window;                /* client specified window */
        bool_t ad_dosync;               /* synchronize? */
        struct sockaddr ad_syncaddr;    /* remote host to synch with */
        struct timeval ad_timediff;     /* server’s time – client’s time
*/
        u_long ad_nickname;             /* server’s nickname for client */
        struct authdes_cred ad_cred;    /* storage for credential */
        struct authdes_verf ad_verf;    /* storage for verifier */
        struct timeval ad_time-stamp;   /* time-stamp sent */
        des_block ad_xkey;              /* encrypted conversation key */
};
```

This structure constitutes the authentication data that actually is sent across the network.

The first four fields of the structure are largely self-explanatory. ad_fullname and ad_servername are strings that contain the client's name and the server's name, respectively. The ad_fullnamelen and ad_servernamelen fields are the lengths of these client and server names, rounded up to a multiple of 4 bytes.

The ad_window field is an unsigned integer that contains the duration (in seconds) of the credentials. By having a small duration during which the authentication credentials are valid, the client protects itself from malicious users who might intercept these credentials and attempt to retransmit them later. If such a scheme were used, the server would detect that the credentials had expired and would deny the request.

The ad_window field is taken directly from the second parameter passed on the authdes_create call. For this reason, you should pass a relatively small number, perhaps 60, as this parameter.

The ad_dosync field is a flag that indicates whether the server and client want to synchronize their concepts of local time. Doing this ensures that the client and server agree on the end of the effective lifetime of a credential. However, synchronizing client and server is a nontrivial procedure and is not recommended. Instead, clients and servers should run an application such as ntpd(8), which implements the Network Time Protocol. By running ntpd, users are assured that the concept of current time on their local machine is essentially the same, at least for DES authentication purposes, as the current time on the server. If the third argument to the authdes_create call is not NULL, the ad_dosync field is set to TRUE.

The ad_syncaddr field is a pointer to the address of the host with whom to synchronize. This value was passed in as the third parameter of the authdes_create call. Again, you should set this parameter to NULL.

The ad_timediff field is a timeval structure, which is defined in the sys/time.h file. It contains the difference between server time and client time, and it is used as part of the synchronization mechanism.

The ad_nickname field is an unsigned long value that the client and server use to speed up validation after initial validation has completed. Essentially, the client specifies in the ad_cred field (described below) whether a “full name” or a "nickname" is being used for the credentials. When a full name is being used, the server must go through the calculations necessary to produce information that allows the client to validate confidently the server's identity. After this is done, the client can specify that, from then on, a nickname credential can be used. This tells the server that there is no need to calculate such complex validation information for the server for each and every RPC request. It is a shorthand mechanism analogous to the AUTH_SHORT mechanism used with UNICOS validation.

The next field in the ad_private structure is the ad_cred field. This is an element of type struct authdes_cred, which is described, as follows:

```
/*
 * A DES authentication credential
 */
struct authdes_cred {
        enum authdes_namekind adc_namekind;
        struct authdes_fullname adc_fullname;
        u_long adc_nickname;
};
```

The adc_namekind field takes either the value ADN_FULLNAME or the value ADN_NICKNAME, depending on whether or not "real" validation is being requested.

The adc_fullname field, which is an authdes_fullname structure, looks like this:

```
/*
 * A fullname contains the network name of the client,
 * a conversation key and the window
 */
struct authdes_fullname {
        char *name;        /* network name of client, up to MAXNETNAMELEN
*/
        des_block key;     /* conversation key */
        u_long window;     /* associated window */
};
```

The types of all fields that have this structure have already been defined.

The last field in the authdes_cred field is adc_nickname, which is just an integer that the client uses to conveniently identify the server.

The ad_verf field of the ad_private structure is of type struct authdes_verf, which is described, as follows:

```
/*
 * A des authentication verifier
 */
struct authdes_verf {
        union {
                struct timeval adv_ctime;       /* clear time */
                des_block adv_xtime;            /* crypt time */
        } adv_time_u;
        u_long adv_int_u;
};
```

This is the structure that the server returns to the client to prove its identity. The first field is a union of a timeval structure and a des_block structure, both of which contain 8 bytes. It is convenient for the server to declare the structure this way, because it must encrypt a time stamp and an integer as part of the proof of identity it sends to the client. The adv_int_u long field is the integer the server encrypts.

The ad_time-stamp field of the ad_private structure is simply the time at which the client created the credential. The server uses this to detect old credentials structures.

The last field of the ad_private structure is the ad_xkey field, which is the encrypted conversation key generated by the client and sent to the server. A pointer to the plain conversation key may be passed as the fourth argument to the authdes_create call. If this pointer is NULL, authdes_create generates and encrypts a pseudo-random conversation key for the client.

Kerberos authentication

E.3

When you use Kerberos authentication, the authentication and validation data exchanged with each RPC call is similar to that used in DES authentication. An RPC client using Kerberos authentication must make a call to authkerb_seccreate to build a Kerberos authentication structure. The Kerberos authentication structure has the following form:

```
typedef struct_auth {
        struct  opaque_auth     ah_cred;
        struct  opaque_auth     ah_verf;
        union   des_block       ah_key;
        struct auth_ops {
                void    (*ah_nextverf)();
                int     (*ah_marshal)();        /* nextverf & serialize */
                int     (*ah_validate)();       /* validate verifier */
                int     (*ah_refresh)();        /* refresh credentials */
              void    (*ah_destroy)();        /* destroy this structure */
        } *ah_ops;
        caddr_t ah_private;
} AUTH;
```

The first field in the AUTH structure is ah_cred. This field points to information the client sends the server to perform authentication. The authentication data is stored in an authkerb_cred structure.

The second field in the AUTH structure, also of struct opaque_auth, is the ah_verf field. This field points to information the client sends to the server for verification. The verification data is stored in an authkerb_verf structure. Both of these fields, ah_cred and ah_verf, are of type opaque_auth.

An opaque_auth structure has the following form:

```
struct opaque_auth {
        enum_t  oa_flavor;              /* flavor of auth */
        caddr_t oa_base;                /* address of more auth stuff */
        u_int   oa_length;              /* not to exceed MAX_AUTH_BYTES */
};
```

The oa_flavor field specifies the type of authentication or validation being done. It can take the value AUTH_NONE, AUTH_UNIX, AUTH_SHORT, AUTH_DES, or AUTH_KERB. For Kerberos RPC, the oa.flavor field is set to AUTH_KERB.

The oa_base field is a pointer to specific data being used for authentication or verification. The ah_cred.oa_base field points to an authkerb_verf structure. The authkerb_cred and authkerb_verf structures are described after the _ak_private structure.

The oa_length field specifies the number of data bytes to which the oa_base field points.

The third field of the AUTH structure is called ah_key, the authentication handle key. This field contains a Kerberos session key used for the duration of the AUTH structure. The ah_key field is of type union des_block, which is specified as follows:

```
union des_block {
        struct {
#ifdef CRAY
                word64 both;
#else
                u_long high;
                u_long low;
#endif
        } key;
        char c[8];
};
typedef union des_block des_block;
```

The des_block is a union of 8 bytes, which constitute the Kerberos session key. This session key exists only for the duration of the AUTH structure, which is no longer than the duration of the CLIENT structure with which this AUTH structure is associated. Typically, a new CLIENT structure is generated each time the client-side application is executed. Thus, a new Kerberos key is generated each time the application is run.

The des_block union contains a conditional compilation statement. This is necessary because, on a Cray Research system, a long consists of 64 bits. On most other machines that run secure RPC, a long value consists of only 32 bits. Thus, to maintain consistency, the key portion of the union is declared a structure that contains one element of type word64 on the Cray Research system. The word64 type is defined in the <rpc/types.h> file, and it is merely a 64-bit entity that allows the user to address the high or low 32 bits of it separately.

The ah_key field of the AUTH structure is used only when performing Kerberos authentication and verification.

The next field in the AUTH structure is the ah_ops field, which is a pointer to a structure that contains function pointers specific to the authentication method. The functions pointed to are enumerated in the AUTH structure and include functions to get the next verifier, to marshal (generate) credentials, to validate credentials, to refresh credentials, and to destroy credentials.

The last field in the AUTH structure is the ah_private field, which is a generic pointer to data specific to the authentication method. In the case of Kerberos authentication, the ah_private field points to a structure of type _ak_private. Users should not manipulate the data within this structure. The contents of the structure are described as follows; this description is only for informational purposes.

```
/*
 * This struct is pointed to by the ah_private field of an “AUTH *”
 * when doing Kerberos authentication.   */
struct _ak_private {
        char ak_service[ANAME_SZ];     /* service name */
        char ak_srv_inst[INST_SZ];     /* server instance */
        char ak_realm[REALM_SZ];       /* realm */
        u_int ak_window;              /* client specified window */
        bool_t ak_dosync;             /* synchronize? */
        char *ak_timehost;            /* remote host to synch with */
        struct timeval ak_timediff;    /* server’s time – client’s time */
        u_long ak_nickname;           /* server’s nickname for client */
        struct timeval ak_time-stamp;  /* time-stamp sent */
        struct authkerb_cred ak_cred;  /* storage for credential */
        struct authkerb_verf ak_verf;  /* storage for verifier */
        KTEXT_ST ak_ticket;           /* Kerberos ticket */
};
```

This structure contains additional data sent to the RPC server.

The ak_service, ak_srv_inst, ak_realm, and ak_window fields are set by the client side call authkerb_seccreate, and are assigned from the service, instance, realm, and window parameters. The ak_timehost field is always left blank. The ak_nickname field is assigned when a reply is received from the RPC server. The server returns the nickname. The nickname is used to speed up validation after the initial validation has completed.

The ak_window field is an unsigned integer that contains the duration (in seconds) of the credentials. By having a small duration during which the authentication credentials are valid, the client protects itself from malicious users who might intercept these credentials and attempt to retransmit them later. If such a scheme were used, the server would detect that the credentials had expired and would deny the request.

The ak_timediff field is a timeval structure, which is defined in the sys/time.h file. It contains the difference between server time and client time, and it is used as part of the synchronization mechanism.

The ak_nickname field is an unsigned long that the client and server use to speed up validation after initial validation has completed. Essentially, the client specifies in the ad_cred field (described below) whether a "full name" or a "nickname" is being used for the credentials. When a full name is being used, the server must go through the calculations necessary to produce information that allows the client to validate confidently the server's identity. After this is done, the client can specify that, from then on, a nickname credential can be used. This tells the server that a less complex verification and authentication may be used.

The next field in the _ak_private structure is the ak_cred field. This is an element of type struct authkerb_cred, which is described, as follows:

```
struct authkerb_cred {
        enum authkerb_namekind akc_namekind;

        struct authkerb_fullname akc_fullname;
        u_long akc_nickname;
};
```

The authkerb_namekind field takes either the value AKN_FULLNAME or the value AKN_NICKNAME, depending on whether or not full validation is being requested. When an AKN_FULLNAME value is used, an authkerb_fullname structure is sent to the server.

The akn_fullname field, which is an authkerb_fullname structure, looks like this:

```
struct authkerb_fullname {
        KTEXT_ST ticket;
        u_long window;                      /* associated window */
        };
```

The KTEXT_ST structure is a Kerberos ticket structure. The u_long window parameter is the window for the ticket. See the include file <krb/krb.h> for a description of the ticket.

Kerberos RPC places restrictions on client and server clocks. They must be synchronized within five minutes of each other. Cray Research recommends that a site run the Network Time Protocol (NTP) time protocol on the client and server to ensure synchronization.

The AUTH_KERB authentication flavor uses Cipher Block Chaining (CBC) mode encryption when sending a fullname credential that includes the ticket and the window. Electronic Code Book (ECB) encryption is used for nickname credentials. The Kerberos session key is used for the initial input vector for CBC encryption.

The ak_verf field of the _ak_private structure is of type struct authkerb_verf, which is described, as follows:

```
struct authkerb_verf {
        union {
               struct timeval akv_ctime;       /* clear time */
               des_block akv_xtime;           /* crypt time */
        } akv_time_u;
        u_long akv_int_u;
};
```

This is the structure that the server returns to the client to prove its identity. The first field is a union of a timeval structure and a des_block structure, both of which contain 8 bytes. It is convenient for the server to declare the structure this way, because it must encrypt a time stamp and an integer as part of the proof of identity it sends to the client. The akv_int_u long field is the integer the server encrypts.

The ak_time-stamp field of the _ak_private structure is simply the time at which the client created the credential. The server uses this to detect old credentials structures.

The server returns the nickname to the client by reusing the authkerb_verf structure on the return call. The client stores the nickname in its _ak_private structure.