Network Working Group
Request for Comments: 2479
Category: Informational

C. Adams
Entrust Technologies

December 1998

Independent Data Unit Protection Generic Security Service Application Program Interface (IDUP-GSS-API)

Status of this Memo

Copyright Notice

ABSTRACT

The IDUP-GSS-API extends the GSS-API [RFC-2078] for applications requiring protection of a generic data unit (such as a file or message) in a way which is independent of the protection of any other data unit and independent of any concurrent contact with designated "receivers" of the data unit. Thus, it is suitable for applications such as secure electronic mail where data needs to be protected without any on-line connection with the intended recipient(s) of that data. The protection offered by IDUP includes services such as data origin authentication with data integrity, data confidentiality with data integrity, and support for non-repudiation services. Subsequent to being protected, the data unit can be transferred to the recipient(s) - or to an archive - perhaps to be processed ("unprotected") only days or years later.

Throughout the remainder of this document, the "unit" of data described in the above paragraph will be referred to as an IDU (Independent Data Unit). The IDU can be of any size (the application may, if it wishes, split the IDU into pieces and have the protection computed a piece at a time, but the resulting protection token applies to the entire IDU). However, the primary characteristic of an IDU is that it represents a stand-alone unit of data whose protection is entirely independent of any other unit of data. If an application protects several IDUs and sends them all to a single receiver, the IDUs may be unprotected by that receiver in any order over any time span; no logical connection of any kind is implied by the protection process itself.

As with RFC-2078, this IDUP-GSS-API definition provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portability of applications to different environments. This specification defines IDUP-GSS-API services and primitives at a level independent of underlying mechanism and programming language environment, and is to be complemented by other, related specifications:

- documents defining specific parameter bindings for particular language environments;
- documents defining token formats, protocols, and procedures to be implemented in order to realize IDUP-GSS-API services atop particular security mechanisms.

TABLE OF CONTENTS

1. IDUP-GSS-API Characteristics and Concepts

The paradigm within which IDUP-GSS-API operates is as follows. An IDUP-GSS-API caller is any application that works with IDUs, calling on IDUP-GSS-API in order to protect its IDUs with services such as data origin authentication with integrity (DOA), confidentiality with integrity (CONF), and/or support for non-repudiation (e.g., evidence generation, where "evidence" is information that either by itself, or when used in conjunction with other information, is used to establish proof about an event or action (note: the evidence itself does not necessarily prove truth or existence of something, but contributes to establish proof) -- see [ISO/IEC] for fuller discussion regarding evidence and its role in various types of non-repudiation). An IDUP-GSS-API caller passes an IDU to, and accepts a token from, its local IDUP-GSS-API implementation, transferring the resulting protected IDU (P-IDU) to a peer or to any storage medium. When a P-IDU is to be "unprotected", it is passed to an IDUP-GSS-API implementation for processing. The security services available through IDUP-GSS-API in this fashion are implementable over a range of underlying mechanisms based on secret-key and/or public-key cryptographic technologies.

During the protection operation, the input IDU buffers may be modified (for example, the data may be encrypted or encoded in some way) or may remain unchanged. In any case, the result is termed a "M-IDU" (Modified IDU) in order to distinguish it from the original IDU. Depending on the desire of the calling application and the capabilities of the underlying IDUP mechanism, the output produced by the protection processing may or may not encapsulate the M-IDU. Thus, the P-IDU may be the contents of a single output parameter (if encapsulation is done) or may be the logical concatenation of an unencapsulated token parameter and a M-IDU parameter (if encapsulation is not done). In the latter case, the protecting application may choose whatever method it wishes to concatenate or

combine the unencapsulated token and the M-IDU into a P-IDU, provided the unprotecting application knows how to de-couple the P-IDU back into its component parts prior to calling the IDUP unprotection set of functions.

It is expected that any output buffer returned by IDUP (i.e., P-IDU or portion thereof) is ready for immediate transmission to the intended receiver(s) by the calling application, if this is desired. In other words, an application wishing to transmit data buffers as they appear from IDUP should not be unduly restricted from doing so by the underlying mechanism.

The IDUP-GSS-API separates the operation of initializing a security environment (the IDUP_Establish_Env() call) from the operations of providing per-IDU protection, for IDUs subsequently protected in conjunction with that environment. Per-IDU protection and unprotection calls provide DOA, CONF, evidence, and other services, as requested by the calling application and as supported by the underlying mechanism.

The following paragraphs provide an example illustrating the dataflows involved in the use of the IDUP-GSS-API by the sender and receiver of a P-IDU in a mechanism-independent fashion. The example assumes that credential acquisition has already been completed by both sides. Furthermore, the example does not cover all possible options available in the protection/unprotection calls.

The sender first calls IDUP_Establish_Env() to establish a security environment. Then, for the IDU to be protected the sender calls the appropriate protection calls (SE, EV, or GP) to perform the IDU protection. The resulting P-IDU, which may (depending on whether or not encapsulation was chosen/available) be either the token itself or the logical concatenation of the token and the M-IDU, is now ready to be sent to the target. The sender then calls IDUP_Abolish_Env() to flush all environment-specific information.

The receiver first calls IDUP_Establish_Env() to establish a security environment in order to unprotect the P-IDU. Then, for the received P-IDU the receiver calls the appropriate unprotection calls (SE, EV, or GP (known a priori, or possibly determined through the use of the IDUP_Get_token_details call)) to perform the P-IDU unprotection. The receiver then calls IDUP_Abolish_Env() to flush all environment-specific information.

It is important to note that absolutely no synchronization is implied or expected between the data buffer size used by the sender as input to the protection calls, the data buffer size used by the receiver as

input to the unprotection calls, and the block sizes required by the underlying protection algorithms (integrity and confidentiality). All these sizes are meant to be independent; furthermore, the data buffer sizes used for the protection and unprotection calls are purely a function of the local environment where the calls are made.

The IDUP-GSS-API design assumes and addresses several basic goals, including the following.

Mechanism independence: The IDUP-GSS-API defines an interface to cryptographically implemented security services at a generic level which is independent of particular underlying mechanisms. For example, IDUP-GSS-API-provided services can be implemented by secret-key technologies or public-key approaches.

Protocol environment independence: The IDUP-GSS-API is independent of the communications protocol suites which may be used to transfer P-IDUs, permitting use in a broad range of protocol environments.

Protocol association independence: The IDUP-GSS-API's security environment construct has nothing whatever to do with communications protocol association constructs, so that IDUP-GSS-API services can be invoked by applications, wholly independent of protocol associations.

Suitability for a range of implementation placements: IDUP-GSS-API clients are not constrained to reside within any Trusted Computing Base (TCB) perimeter defined on a system where the IDUP-GSS-API is implemented; security services are specified in a manner suitable for both intra-TCB and extra-TCB callers.

1.1. IDUP-GSS-API Constructs

This section describes the basic elements comprising the IDUP-GSS-API.

1.1.1. Credentials

Credentials in IDUP-GSS-API are to be understood and used as described in GSS-API [RFC-2078].

1.1.2. Tokens

Tokens in IDUP-GSS-API are to be understood and used as described in GSS-API [RFC-2078] with the exception that there are no context-level tokens generated by IDUP-GSS-API. The IDUP-GSS-API token may (depending on the underlying mechanism) encapsulate the M-IDU or may

be logically concatenated with the M-IDU prior to transfer to a target; furthermore, for some evidence services the token may be sent independently of any other data transfer.

1.1.3. Security Environment

The "security environment" in IDUP-GSS-API is entirely different from the concept of security contexts used in GSS-API [RFC-2078]. Here, a security environment exists within a calling application (that is, it is purely local to the caller) for the purpose of protecting or unprotecting one or more IDUs using a particular caller credential or set of credentials. In GSS-API, on the other hand, a security context exists between peers (the initiator and the target) for the purpose of protecting, in real time, the data that is exchanged between them. Although they are different concepts, the env_handle in IDUP-GSS-API is similar to the context_handle in GSS-API in that it is a convenient way of tying together the entire process of protecting or unprotecting one or more IDUs using a particular underlying mechanism. As with the GSS-API security contexts, a caller can initiate and maintain multiple environments using the same or different credentials.

1.1.4. Mechanism Types

Mechanism types in IDUP-GSS-API are to be understood and used as described in GSS-API [RFC-2078].

1.1.5. Naming

Naming in IDUP-GSS-API is to be understood and used as described in GSS-API [RFC-2078].

1.1.6. Channel Bindings

The concept of channel bindings discussed in GSS-API [RFC-2078] is not relevant to the IDUP-GSS-API.

1.2. IDUP-GSS-API Features and Issues

This section describes aspects of IDUP-GSS-API operations and of the security services which the IDUP-GSS-API provides. It also provides commentary on design issues.

1.2.1. Status Reporting

Status reporting in IDUP-GSS-API is to be understood and used as described in GSS-API [RFC-2078], with the addition of a number of IDUP-specific status codes. Descriptions of the major_status codes

used in IDUP are provided in Table 1. Codes that are informatory (i.e., that do not cause the requested operation to fail) are indicated with the symbol "(I)".

As with GSS-API, minor_status codes, which provide more detailed status information than major_status codes, and which may include status codes specific to the underlying security mechanism, are not specified in this document.

Table 1: IDUP-GSS-API Major Status Codes

GSS_S_BAD_MECH indicates that a mech_type unsupported by the IDUP_GSS-API implementation was requested, causing the environment establishment operation to fail.

GSS_S_BAD_QOP indicates that the provided qop_alg value is not recognized or supported for the environment.

GSS_S_BAD_MIC indicates that the received P-IDU contains an incorrect integrity field (e.g., signature or MAC) for the data.

GSS_S_COMPLETE indicates that the requested operation was successful.

GSS_S_CREDENTIALS_EXPIRED indicates that the credentials associated with this operation have expired, so that the requested operation cannot be performed.

GSS_S_DEFECTIVE_CREDENTIAL indicates that consistency checks performed on the credential structure referenced by claimant_cred_handle failed, preventing further processing from being performed using that credential structure.

GSS_S_DEFECTIVE_TOKEN indicates that consistency checks performed on the received P-IDU failed, preventing further processing from being performed.

GSS_S_FAILURE indicates that the requested operation could not be accomplished for reasons unspecified at the IDUP-GSS-API level, and that no interface-defined recovery action is available.

GSS_S_NO_CRED indicates that no environment was established, either because the input cred_handle was invalid or because the caller lacks authorization to access the referenced credentials.

IDUP_S_BAD_DOA_KEY indicates that the key used to provide IDU data origin auth. / integ. has either expired or been revoked.

IDUP_S_BAD_ENC_IDU indicates that decryption of the received IDU cannot be completed because the encrypted IDU was invalid/defective (e.g., the final block was short or had incorrect padding).

IDUP_S_BAD_KE_KEY indicates that the key used to establish a key for confidentiality purposes between originator and target has either expired or been revoked.

IDUP_S_BAD_TARG_INFO indicates that the full set of supplied information regarding the target(s) is invalid or is insufficient for the protection of an IDU, so P-IDU cannot be created.

IDUP_S_DEFECTIVE_VERIF indicates that consistency checks performed on Service_Verification_Info failed, preventing further processing from being performed with that parameter.

IDUP_S_ENCAPSULATION_UNAVAIL (I) indicates that the underlying mechanism does not support encapsulation of the M-IDU into the token.

IDUP_S_INAPPROPRIATE_CRED indicates that the credentials supplied do not contain the information necessary for P-IDU unprotection.

IDUP_S_INCOMPLETE (I) indicates that the unprotection of the P-IDU is not yet complete (i.e., a determination cannot yet be made on the validity of the P-IDU). The application should call IDUP_Form_Complete_PIDU and then should call this function again with the complete P-IDU.

IDUP_S_INCONSISTENT_PARAMS indicates that the supplied parameters are inconsistent (e.g., only one or the other of two parameters may be supplied, but both have been input).

IDUP_S_MORE_OUTBUFFER_NEEDED (I) indicates that the output buffer supplied is too small to hold the generated data. The application should continue calling this routine (until GSS_S_COMPLETE is returned) in order to get all remaining output data.

IDUP_S_MORE_PIDU_NEEDED (I) indicates that not enough of the P-IDU has been input yet for the completion of StartUnprotect. The application should call this routine again with another buffer of P-IDU in partial(initial)_pidu_buffer.

IDUP_S_NO_ENV indicates that no valid environment was recognized for the env_handle provided.

IDUP_S_NO_MATCH indicates that Service_Verification_Info (or evidence_check) and the P-IDU to be verified do not match.

IDUP_S_REQ_TIME_SERVICE_UNAVAIL indicates that the time service requested (TTIME or UTIME) is not available in the environment.

IDUP_S_SERVICE_UNAVAIL indicates that the underlying mechanism does not support the service requested.

IDUP_S_SERV_VERIF_INFO_NEEDED (I) indicates that the Service_Verification_Info parameter bundle must be input in order for service verification to proceed. The output parameter service_verification_info_id contains an identifier which may be used by the calling application to locate the necessary information.

IDUP_S_UNKNOWN_OPER_ID indicates that the input prot_oper_id value is not recognized or supported in the underlying mechanism.

1.2.2. Per-IDU Security Service Availability

Per-IDU security service availability in IDUP-GSS-API is to be understood and used as described in GSS-API [RFC-2078], with the exception that combinations of services requested by the calling application and supported by the underlying mechanism may be applied simultaneously to any IDU (true for both the SE and the EV calls, but true in the fullest sense for the GP calls).

GSS-API callers desiring per-message security services should check the relevant service OBJECT IDs at environment establishment time to ensure that what is available in the established environment is suitable for their security needs.

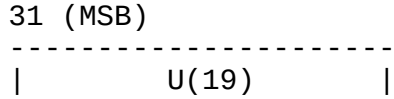
1.2.3. Per-IDU Replay Detection and Sequencing

The concept of per-IDU replay detection and sequencing discussed in GSS-API [RFC-2078] is not relevant to the IDUP-GSS-API.

1.2.4. Quality of Protection

The concept of QOP control in IDUP-GSS-API is to be understood essentially as described in GSS-API [RFC-2078]. However, the actual description and use of the QOP parameter is given as follows.

The qop_algs parameter for IDUP is defined to be a 32-bit unsigned integer with the following bit-field assignments:



where

U is a 19-bit Unspecified field (available for future use/expansion) -- must be set to zero;

TS is a 5-bit Type Specifier (a semantic qualifier whose value specifies the type of algorithm which may be used to protect the corresponding IDU -- see below for details);

IA is a 4-bit field enumerating Implementation-specific Algorithms; and

MA is a 4-bit field enumerating Mechanism-defined Algorithms.

The interpretation of the qop_algs parameter is as follows. The MA field is examined first. If it is non-zero then the algorithm used to protect the IDU is the mechanism-specified algorithm corresponding to that integer value.

If MA is zero then IA is examined. If this field value is non-zero then the algorithm used to protect the IDU is the implementation-specified algorithm corresponding to that integer value. Note that use of this field may hinder portability since a particular value may specify one algorithm in one implementation of the mechanism and may not be supported or may specify a completely different algorithm in another implementation of the mechanism.

Finally, if both MA and IA are zero then TS is examined. A value of zero for TS specifies the default algorithm for the established mechanism. A non-zero value for TS corresponds to a particular algorithm qualifier and selects any algorithm from the mechanism specification which satisfies that qualifier (which actual algorithm is selected is an implementation choice; the calling application need not be aware of the choice made).

The following TS values (i.e., algorithm qualifiers) are specified; other values may be added in the future.

When qop_algs is used to select a confidentiality algorithm:

```
00000  (0) = default confidentiality algorithm
00001  (1) = IDUP_SYM_ALG_STRENGTH_STRONG
00010  (2) = IDUP_SYM_ALG_STRENGTH_MEDIUM
00011  (3) = IDUP_SYM_ALG_STRENGTH_WEAK
11111 (31) = IDUP_NO_CONFIDENTIALITY
```

When qop_algs is used to select a DOA/integrity algorithm:

```
00000  (0) = default integrity algorithm
00001  (1) = IDUP_INT_ALG_DIG_SIGNATURE
             (integrity provided through a digital signature)
00010  (2) = IDUP_INT_ALG_NON_DIG_SIGNATURE
             (integrity without a dig. sig. (e.g., with a MAC))
11111 (31) = IDUP_NO_INTEGRITY
```

Clearly, qualifiers such as strong, medium, and weak are debatable and likely to change with time, but for the purposes of this version of the specification we define these terms as follows. A confidentiality algorithm is "weak" if the effective key length of the cipher is 40 bits or less; it is "medium-strength" if the effective key length is strictly between 40 and 80 bits; and it is "strong" if the effective key length is 80 bits or greater. ("Effective key length" describes the computational effort required to break a cipher using the best-known cryptanalytic attack against that cipher.)

A five-bit TS field allows up to 30 qualifiers for each of confidentiality and integrity (since "0" is reserved for "default" and "31" is reserved for "none", as shown above). This document specifies three for confidentiality and two for integrity, leaving a lot of room for future specification. Suggestions of qualifiers such as "fast", "medium-speed", and "slow" have been made, but such terms are difficult to quantify (and in any case are platform- and processor-dependent), and so have been left out of this initial specification. The intention is that the TS terms be quantitative, environment-independent qualifiers of algorithms, as much as this is possible.

Use of the qop_algs parameter as defined above is ultimately meant to be as follows.

- TS values are specified at the IDUP-GSS-API level and are therefore portable across mechanisms. Applications which know nothing about algorithms are still able to choose "quality" of protection for their message tokens.

- MA values are specified at the mechanism level and are therefore portable across implementations of a mechanism.

- IA values are specified at the implementation level (in user documentation, for example) and are therefore typically non-portable. An application which is aware of its own mechanism implementation and the mechanism implementation of its intended P-IDU recipient, however, is free to use these values since they will be perfectly valid and meaningful for protecting IDUs between those entities.

The receiver of a P-IDU must pass back to its calling application (in IDUP_Start_Unprotect()) a qop_algs parameter with all relevant fields set. For example, if triple-DES has been specified by a mechanism as algorithm 8, then a receiver of a triple-DES-protected P-IDU must pass to its application (TS=1, IA=0, MA=8). In this way, the application is free to read whatever part of the qop_algs parameter it understands (TS or IA/MA).

1.2.5. The Provision of Time

IDUP mechanisms should make provision in their protocols for the carrying of time information from originator to target(s). That is, a target (a legitimate recipient) should get some indication during unprotection regarding the time at which the protection operation took place. This is particularly important if the mechanism offers non-repudiation services because in some cases evidence verification may only be achievable if the time at which the evidence was generated is known.

Depending upon the platform and resources available to the implementation, an IDUP environment may have access to a source of trusted (secure) time, untrusted (local) time, both kinds of time, or no time. OBJECT IDs indicating such availability are returned by the IDUP_Establish_Env() call. When starting a protection operation, an application may specify which time services it wishes to have applied to the IDU. Similarly, for unprotection, an application may specify which kind of time (if any) to consult when the validity of the P-IDU is to be established. Specifying both kinds of time is interpreted to mean that the calling application does not care which kind of time is used.

The IDUP calls which use a time parameter specify the type of that parameter to be INTEGER. This INTEGER is defined in all cases to be the number of seconds which have elapsed since midnight, January 1, 1970, coordinated universal time.

2. Interface Descriptions

This section describes the IDUP-GSS-API's operational interface, dividing the set of calls offered into five groups. Credential management calls are related to the acquisition and release of credentials by API callers. Environment-level calls are related to the management of the security environment by an API caller. Per-IDU calls are related to the protection or unprotection of individual IDUs in established security environments. Special-purpose calls deal with unusual or auxiliary evidence generation/verification requirements. Support calls provide extra functions useful to IDUP-GSS-API callers. Table 2 groups and summarizes the calls in tabular fashion.

Table 2: IDUP-GSS-API Calls

CREDENTIAL MANAGEMENT
(see the calls given in Section 2.1 of GSS-API [RFC-2078])

ENVIRONMENT-LEVEL CALLS
IDUP_Establish_Env
IDUP_Abolish_Env
IDUP_Inquire_Env

PER-IDU CALLS
SE (SIGN,ENCRYPT) CALLS
- IDUP_SE_SingleBuffer_Protect
- IDUP_SE_SingleBuffer_Unprotect
- IDUP_SE_MultiBuffer_StartProtect
- IDUP_SE_MultiBuffer_EndProtect
- IDUP_SE_MultiBuffer_StartUnprotect
- IDUP_SE_MultiBuffer_EndUnprotect
- IDUP_SE_Process_Buffer

EV (EVIDENCE) CALLS
- IDUP_EV_SingleBuffer_Generate
- IDUP_EV_SingleBuffer_Verify
- IDUP_EV_MultiBuffer_StartGenerate
- IDUP_EV_MultiBuffer_EndGenerate
- IDUP_EV_MultiBuffer_StartVerify
- IDUP_EV_MultiBuffer_EndVerify
- IDUP_EV_Process_Buffer

GP (GENERAL PROTECTION) CALLS
- IDUP_Start_Protect
- IDUP_Protect
- IDUP_End_Protect
- IDUP_Start_Unprotect
- IDUP_Unprotect
- IDUP_End_Unprotect

```
SPECIAL-PURPOSE CALLS  (might not be supported by all mechanisms)
IDUP_Form_Complete_PIDU

SUPPORT CALLS
IDUP_Acquire_cred_with_auth
IDUP_Get_Token_Details
IDUP_Get_Policy_Info
IDUP_Cancel_Multibuffer_Op
(see also the calls given in Section 2.4 of GSS-API [RFC-2078])
```

In terms of conformance to this specification, IDUP-GSS-API implementations must support the credential management calls, the environment-level calls, some subset of the per-IDU calls, and the support calls (except where explicitly stated otherwise in Section 2.5). The subset of per-IDU calls supported will depend upon the underlying mechanisms supported and will typically be the SE calls, or the EV calls, or both. As stated in Section 2.3.2.1, implementations are encouraged to support the more powerful GP calls to anticipate the future needs of applications developers, but this is not required for conformance.

2.1. Credential management calls

2.1.1. Relationship to GSS-API

Credential management in IDUP-GSS-API is to be understood and used as described in GSS-API [RFC-2078]. The calls given in Section 2.1 of GSS-API (including all associated parameters) are unchanged, although the interpretation of the cred_usage parameter in the GSS-API calls for IDUP purposes is as follows.

```
ENCRYPT_ONLY    8
DECRYPT_ONLY   16
SIGN_ONLY      32
VERIFY_ONLY    64
```

The values above may be logically OR'ed together in any desired combination to restrict credential usage (where OR'ing all values results in NO_RESTRICTION). Future possible values for this parameter are for further study.

The call IDUP_Acquire_cred_with_auth has been added as a support call in this specification to permit authenticated credential acquirement; see Section 2.5.2 for details.

2.2. Environment-level calls

This group of calls is devoted to the establishment and management of an environment for the purpose of IDU protection and unprotection. Before protecting or unprotecting any IDU, an application must call IDUP_Establish_Env() to initialize environment information and select the underlying IDUP-GSS mechanism to be used. A series of protection or unprotection calls is made to process each IDU, the protection calls resulting in a P-IDU for each. Finally, IDUP_Abolish_Env() is called to flush all environment information.

Semantically, acquiring credentials and establishing an environment is (in many cases) analogous to logging in to a system -- it authenticates a local user to the system and gives that user access to a set of operations which can be performed.

2.2.1. Relationship to GSS-API

The set of calls described in this section is used in place of the calls described in Section 2.2 of GSS-API [RFC-2078], since those calls are specific to a session-oriented environment.

2.2.2. IDUP_Establish_Env call

Inputs: o claimant_cred_handle CREDENTIAL HANDLE,
-- NULL parameter specifies "use default"

- req_mech_type OBJECT IDENTIFIER,
  -- NULL parameter specifies "use default"
- req_environmentPolicies EnvironmentPolicies,
  -- NULL parameter specifies "use default"
- req_services SET OF OBJECT IDENTIFIER,
  -- GSS_C_NO_OID_SET requests full set of services available
  -- for req_mech_type

Outputs:

- major_status INTEGER,
- minor_status INTEGER,
- env_handle ENVIRONMENT HANDLE,
- actual_mech_type OBJECT IDENTIFIER,
  -- actual mechanism always indicated, never NULL
- actual_environmentPolicies EnvironmentPolicies,
  -- actual values always indicated, never NULL
- ret_services SET OF OBJECT IDENTIFIER,

Return major_status codes:

- GSS_S_COMPLETE
  -- environment-level information was successfully initialized,

```
      -- and IDU / P-IDU processing can begin.
   o  GSS_S_DEFECTIVE_CREDENTIAL
   o  GSS_S_NO_CRED
   o  GSS_S_CREDENTIALS_EXPIRED
      -- the credentials provided through claimant_cred_handle are
      -- no longer valid, so environment cannot be established.
   o  GSS_S_BAD_MECH
   o  GSS_S_FAILURE
```

The following structures are defined to facilitate environment policy input and output:

```
EnvironmentPolicies ::= SEQUENCE {
   confPolicy     [0] PolicyAndTime OPTIONAL,
   -- NULL parameter (on input) specifies "use default"
   integPolicy    [1] PolicyAndTime OPTIONAL,
   -- NULL parameter (on input) specifies "use default"
   evidencePolicy [2] PolicyAndTime OPTIONAL }
   -- NULL parameter (on input) specifies "use default"

PolicyAndTime ::= SEQUENCE {
   policy             OBJECT IDENTIFIER,
   -- this environment-level policy identifier is separate from
   -- the policy provisions connected with credentials, if they exist
   time               INTEGER
   -- on input:  the policy rules available at the specified time
   -- on output: the time at which the policy rules came into effect
   -- (defined to be the number of seconds elapsed since midnight,
   -- January 1, 1970, coordinated universal time)
   endTime            INTEGER OPTIONAL }
   -- on input:  unused
   -- on output: the expiration time of the given policy rules
```

This routine is used by an application which protects or unprotects IDUs. Using information in the credentials structure referenced by claimant_cred_handle, IDUP_Establish_Env() initializes the data structures required to protect or unprotect IDUs. The claimant_cred_handle, if non-NULL, must correspond to a valid credentials structure.

This routine returns an env_handle for all future references to this environment; when protection, unprotection, or IDUP_Abolish_Env() calls are made, this handle value will be used as the input env_handle argument. It is the caller’s responsibility to establish a communications path to the intended recipients of the P-IDU, and to transmit the P-IDU to those recipients over that path. This may occur subsequent to the IDUP_Abolish_Env() call.

The req_services parameter may be used by the calling application to request that data origin authentication with integrity, confidentiality with integrity, evidence generation, and/or evidence verification services be available in the established environment. Requests can also be made for "trusted" or "untrusted" time services. Requesting evidence generation or verification indicates that the calling application may wish to generate or verify evidence information for non-repudiation purposes (note: an IDU protector may request that a flag be inserted into a P-IDU asking a recipient to provide an evidence of the type "non-repudiation of delivery"; however, the IDUP-GSS-API cannot by itself guarantee that the evidence will be sent because there is no way to force a target to send an evidence_token back to the IDU protector).

Not all features will be available in all underlying mech_types; the returned value of ret_services indicates, as a function of mech_type processing capabilities and the initiator-provided input OBJECT IDs, the set of features which will be available in the environment. The value of this parameter is undefined unless the routine's major_status indicates COMPLETE. Failure to provide the precise set of services desired by the caller does not cause environment establishment to fail; it is the caller's choice to abolish the environment if the service set provided is unsuitable for the caller's use. The returned mech_type value indicates the specific mechanism employed in the environment and will never indicate the value for "default".

The following OBJECT IDs are defined for protection and unprotection services (the OBJECT ID iso.org.dod.internet.security.services, 1.3.6.1.5.7, has been assigned by IANA, and some of the security services under that node are assigned as shown below). It is recognized that this list may grow over time.

```
PER_CONF = { 1.3.6.1.5.7.1.1 }
   -- perform data confidentiality (i.e., encrypt data)
PER_CONF_FULL = { 1.3.6.1.5.7.1.3 }
   -- perform full confidentiality (i.e., encrypt data and sig)
   -- (may be used only when PER_DOA is requested simultaneously)
PER_DOA  = { 1.3.6.1.5.7.3.1 }
   -- perform data origin authentication with data integrity
PER_DOA_CIPH  = { 1.3.6.1.5.7.3.3 }
   -- perform DOA with DI over ciphertext (rather than plaintext)
   -- (may be used only when PER_CONF is requested simultaneously)
PER_POO  = { 1.3.6.1.5.7.4.1 }
   -- perform (i.e., create) non-repudiable "proof of origin"
PER_POD  = { 1.3.6.1.5.7.4.3 }
   -- perform (i.e., create) non-repudiable "proof of delivery"
```

```
REC_CONF = { 1.3.6.1.5.7.1.2 }
   -- receive data confidentiality (i.e., decrypt data)
REC_CONF_FULL = { 1.3.6.1.5.7.1.4 }
   -- receive full confidentiality (i.e., decrypt data and sig)
   -- (may be used only when REC_DOA is received simultaneously)
REC_DOA  = { 1.3.6.1.5.7.3.2 }
   -- receive / verify DOA with data integrity
REC_DOA_CIPH  = { 1.3.6.1.5.7.3.4 }
   -- verify DOA with DI over ciphertext (rather than plaintext)
   -- (may be used only when PER_CONF is received simultaneously)
REC_POO  = { 1.3.6.1.5.7.4.2 }
   -- receive / verify "proof of origin"
REC_POD  = { 1.3.6.1.5.7.4.4 }
   -- receive / verify "proof of delivery"
TTIME    = { 1.3.6.1.5.7.7.1 }
   -- trusted time availability
UTIME    = { 1.3.6.1.5.7.7.2 }
   -- untrusted time availability
```

The PER_CONF return value (in the ret_services paramater) indicates whether the environment supports confidentiality services, and so informs the caller whether or not a request for encryption can be honored. In similar fashion, the PER_DOA return value indicates whether DOA services are available in the established environment, and the PER_POO and PER_POD return values indicate whether evidence generation services are available. The TTIME and UTIME values indicate whether trusted time and untrusted time are available for protection / unprotection services.

Note that, unlike a GSS "context", an IDUP environment does not have an explicit lifetime associated with it. Instead, it relies on the lifetime of the calling entity’s credential (set by the caller in the GSS_Acquire_cred() call). When the credential expires (or is explicitly deleted in any other way), no new operations are allowed in the IDUP environment (although operations which have begun, such as the Protection set of calls, can be taken to completion).

2.2.3. IDUP_Abolish_Env call

Input:
o env_handle ENVIRONMENT HANDLE

Outputs:
o major_status INTEGER,
o minor_status INTEGER,

Return major_status codes:
o GSS_S_COMPLETE
  -- the relevant environment-specific information was flushed.
o IDUP_S_NO_ENV
o GSS_S_FAILURE

This call is made to flush environment-specific information. (Once an environment is established, cached credential and environment-related info. is expected to be retained until an IDUP_Abolish_Env() call is made or until the cred. lifetime expires.) Attempts to perform IDU processing on a deleted environment will result in error returns.

2.2.4. IDUP_Inquire_Env call

Input:
o env_handle ENVIRONMENT HANDLE,

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o mech_type OBJECT IDENTIFIER,
  -- the mechanism supporting this environment
o environmentPolicies EnvironmentPolicies,
  -- the environment policies in effect
o ret_services SET OF OBJECT IDENTIFIER,

Return major_status codes:
o GSS_S_COMPLETE
  -- referenced environment is valid and mech_type and other return
  -- values describe the characteristics of the environment.
o GSS_S_CREDENTIALS_EXPIRED
o IDUP_S_NO_ENV
o GSS_S_FAILURE

This routine provides environment-related information to the caller.

2.3. Per-IDU calls

This group of calls is used to perform IDU protection and unprotection processing on an established IDUP environment. Some of these calls may block pending network interactions (depending on the underlying mechanism in use). These calls may be invoked by an IDU's protector or by the P-IDU's recipient. Members of this group form pairs; the output from the protection types of calls is typically meant to be input to the unprotection types of calls.

The per-IDU calls can support caller-requested data origin authentication with data integrity, confidentiality with data integrity, evidence, and evidence-requested-from-target services.

The protection operations output a token which encapsulates all the information required to unprotect the IDU. The token is passed to the target (possibly separate from the M-IDU) and is processed by the unprotection calls at that system. Unprotection performs decipherment, DOA verification, evidence verification, or notification of evidence requested, as required.

Each of the two main operations (protection and unprotection) may be separated into three parts: "Start_Operation"; "Operation" (which may be called once for each buffer of input data); and "End_Operation". This separation is available for the case where the IDU or P-IDU is to be processed one buffer at a time. "Start_Operation" allows the caller to specify or retrieve the appropriate "Quality" used during the processing. "Operation" is concerned with the processing itself, receiving a buffer of input data and potentially returning a buffer of output data. "End_Operation" performs any required clean-up and creates the appropriate token or states whether the input token was verified.

If the IDU or P-IDU is wholly contained in a single buffer, the three-part protection/unprotection processing need not be done. Instead, protection or unprotection can be accomplished using only a single call, simplifying application code.

2.3.1. Relationship to GSS-API

The set of calls described in this section is used in place of the calls GSS_GetMIC(), GSS_VerifyMIC, GSS_Wrap(), and GSS_Unwrap() which are specified in [RFC-2078], since those calls are specific to a session-oriented environment.

2.3.2. The "SE" Calls

2.3.2.1. IDUP_SE Purpose

The "SE" group of calls provides a very simple, high-level interface to underlying IDUP mechanisms when application developers need access only to signature and encryption protection/unprotection services. It includes both the single-buffer and multiple-buffer IDU cases and can be used for signing only, encrypting only, signing and encrypting (in either order, and with or without visibility of the resulting signature), and "clear signing" (where the data is not modified in any way and the signature itself is returned as a separate item). [Note that encapsulation occurs in all cases except for clear signing, so that these calls provide functionality similar to the GSS_Wrap call.]

Note that the term "signing" is used in its most generic sense, not necessarily implying the use of public-key techniques. This concept has also been called "sealing" in other contexts (e.g., in other standardization efforts).

The SE calls may be viewed by mechanism implementors as an "API" to the more powerful GP calls defined later and so may be implemented as simple mapping functions to those calls (when those optional calls are supported). Application callers, on the other hand, may find that the SE calls are all they currently need for many environments. At some time in the future when they have need of non-repudiation or "directed receipts" types of services, they may consider using the EV calls (or the GP calls -- when these are supported -- if complex and sophisticated combinations of services are required). To assist in this migration path, mechanism implementors are encouraged to support the full set of IDUP calls (i.e., the SE, EV, and GP calls) even though some calling applications will only use the SE calls in the short term.

2.3.2.2. IDUP_SE Parameter Bundles

The concept of "parameter bundles" is used in the calls presented in the following subsections in order to simplify their presentation and clarify their intended purpose and use. See Section 2.3.4.1 for a more complete description of parameter bundles.

The following parameter bundles are used in the "SE" protection and unprotection sets of calls.

```
   o  Protect_Options PARAMETER BUNDLE
      o  protect_operation  INTEGER {
            sign_only              (0),
            encrypt_only           (1),
            sign_and_encrypt       (2),
            -- let mechanism choose order (and readability of signature)
            sign_then_encrypt_data (3),
            -- sign, then encrypt plaintext (leaving signature in clear)
            sign_then_encrypt_full (4),
            -- sign, then encrypt everything (including signature)
            encrypt_then_sign      (5),
            -- encrypt, then sign the ciphertext
            clear_sign_only        (6)
         } OPTIONAL,
      o  protect_oper_oid   OBJECT IDENTIFIER OPTIONAL,
         -- may be used in place of above parameter if OID is known
      o  sign_qop_alg      UNSIGNED INTEGER,
      o  sign_qop_algID    AlgorithmIdentifier, --overrides sign_qop_alg
      o  enc_qop_alg       UNSIGNED INTEGER,
      o  enc_qop_algID     AlgorithmIdentifier, --overrides enc_qop_alg
      o  idu_type_string    OCTET STRING,
         -- type of the IDU ("data", "e-mail doc", MIME type, etc.)
      o  pidu_type_string   OCTET STRING,
      o  mech_indep_encap_req BOOLEAN -- (see Appendix A)

   o  PIDU_Information PARAMETER BUNDLE
      o  protect_options    Protect_Options,
      o  originator_name    INTERNAL NAME,
      o  originator_role    Originator_Role, -- (see Section 2.3.4.1)
      o  protection_time    INTEGER,
   o  Bad_Target_Name PARAMETER BUNDLE,  -- same as in Section 2.3.3.2
      o  bad_targ_name     INTERNAL NAME,
      o  bad_targ_status   INTEGER,
         -- a status flag giving the reason for rejection of the name
         -- in bad_targ_name.  Specified reasons include:
         --  SYNTAX_INVALID        (0) the syntax of the name is invalid;
         --  NAME_UNRECOGNIZED     (1) the name is not recognized;
         --  NAME_AMBIGUOUS        (2) the name cannot be resolved;
         --  ACCESS_DENIED         (3) access to this target is denied;
         --  CERTIFICATE_NOT_FOUND (4) the encryption certificate of the
                                       target could not be found.

   o  Target_Info PARAMETER BUNDLE,      -- same as in Section 2.3.3.2
      o  targ_names        SET OF INTERNAL NAME,
      o  bad_targ_count    INTEGER,
      o  bad_target_names  SET OF Bad_Target_Name,
```

2.3.2.3. IDUP_SE major_status codes

The following major_status return codes are defined for the "SE" calls in this section:

- GSS_S_COMPLETE
- IDUP_S_MORE_OUTBUFFER_NEEDED
  -- returned (by any SE call) to indicate that there is more output
  -- data than can fit into the supplied buffers. The application
  -- should save the returned data and call again to retrieve the
  -- remaining output.
- IDUP_S_MORE_PIDU_NEEDED
  -- indicates that more PIDU data is needed for the StartUnprotect
  -- operation (e.g., so that PIDU_Information or initial_idu_buffer
  -- may be returned).
- IDUP_S_INCONSISTENT_PARAMS
- GSS_S_CREDENTIALS_EXPIRED
- IDUP_S_NO_ENV
- GSS_S_BAD_QOP
- GSS_S_FAILURE

If Target_Info is used as an input parameter (e.g., if an encryption operation is being performed), the following major_status return code is also defined:

- IDUP_S_BAD_TARG_INFO

Note for this return code that if one or more of the targets in targ_names cannot be used as a valid recipient of the P-IDU, these names will be returned in bad_targ_names (with associated status codes in bad_targ_status). As long as at least one of the targets can be used, however, this does not cause this call to fail (i.e., the failure code IDUP_S_BAD_TARG_INFO is not returned); it is the caller’s choice to discontinue IDU protection if the target set which can be used is unsuitable for the caller’s purposes.

2.3.2.4. IDUP_SE_SingleBuffer_Protect call

Inputs:
- env_handle ENVIRONMENT HANDLE,
- Protect_Options PARAMETER BUNDLE,
- Target_Info PARAMETER BUNDLE,
- idu_buffer OCTET STRING
- additional_protection BOOLEAN
  -- TRUE if idu_buffer is the output of a previous protection
  -- operation (i.e., if this is the second (or higher) in a
  -- series of SE/EV protection calls)

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o pidu_buffer OCTET STRING,
o sig_token OCTET STRING
-- used if Protect_Options is clear_sign_only

Using the security environment referenced by env_handle, encrypt and/or sign the supplied IDU. If "clear signing" is performed, the signature will be returned in sig_token and pidu_buffer may be empty (depends on underlying mechanism).

2.3.2.5. IDUP_SE_SingleBuffer_Unprotect call

Inputs:
o env_handle ENVIRONMENT HANDLE,
o pidu_buffer OCTET STRING,
-- may contain an IDU if sig_token is non-NULL (i.e., if
-- clear_sign_only protection was applied)
o sig_token OCTET STRING

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o idu_buffer OCTET STRING,
-- may be empty if clear_sign_only protection was applied (depends
-- on underlying mechanism)
o PIDU_Information PARAMETER BUNDLE
o additional_unprotection BOOLEAN
-- TRUE if idu_buffer should be input to another unprotection
-- operation (i.e., if this should not be the last in a series
-- of SE/EV unprotection calls)

Using the security environment referenced by env_handle, decrypt and/or verify the supplied PIDU and return the contained IDU along with all available PIDU_Information.

2.3.2.6. IDUP_SE_MultiBuffer_StartProtect call

Inputs:
o env_handle ENVIRONMENT HANDLE,
o Protect_Options PARAMETER BUNDLE,
o Target_Info PARAMETER BUNDLE,
o additional_protection BOOLEAN, -- (see Section 2.3.2.4)
o idu_size INTEGER -- (see Section 2.3.4.2)

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o initial_pidu_buffer OCTET STRING
-- may be empty (depends on underlying mechanism)

Using the security environment referenced by env_handle, initialize the data structures required to begin the process of signing and/or encrypting the IDU (which will be supplied in multiple buffers to the Process_Buffer call).

2.3.2.7. IDUP_SE_MultiBuffer_EndProtect call

Inputs:
o env_handle ENVIRONMENT HANDLE

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o final_pidu_buffer OCTET STRING,
o sig_token OCTET STRING
-- used if Protect_Options was clear_sign_only

Using the security environment referenced by env_handle, complete the protection processing on the data and place the computed output in final_pidu_buffer and/or sig_token. Successful application of IDUP_SE_MultiBuffer_EndProtect() does not guarantee that unprotection can necessarily be performed successfully when the P-IDU arrives at the target (for example, it may be damaged in transit).

2.3.2.8. IDUP_SE_MultiBuffer_StartUnprotect call

Inputs:
o env_handle ENVIRONMENT HANDLE,
o initial_pidu_buffer OCTET STRING,
o sign_qop_alg_in UNSIGNED INTEGER,
-- used if Protect_Options was clear_sign_only (and calling
-- application has prior knowledge of signing alg. applied);
-- if NULL, then sig_token must be supplied
o sig_token OCTET STRING
-- used if Protect_Options was clear_sign_only;
-- if NULL, then sign_qop_alg_in must be supplied

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o PIDU_Information PARAMETER BUNDLE,
-- returns all available information

o initial_idu_buffer OCTET STRING
-- may be empty

Using the security environment referenced by env_handle, initialize the data structures required to begin the process of decrypting and/or verifying the PIDU (which will be supplied in multiple buffers to the Process_Buffer call).

The parameters sign_qop_alg_in and sig_token should not both be supplied (i.e., they should not both be non-NULL). If they are both non-NULL, however, sig_token is taken to be authoritative since this is the token created at protection time and therefore is guaranteed to carry the information needed to unprotect.

2.3.2.9. IDUP_SE_MultiBuffer_EndUnprotect call

Inputs:
o env_handle ENVIRONMENT HANDLE,
o sig_token OCTET STRING OPTIONAL
-- used if Protect_Options was clear_sign_only and sig_token was
-- not available when StartUnprotect was called

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o PIDU_Information PARAMETER BUNDLE,
-- returns all available information
o final_idu_buffer OCTET STRING -- may be empty
o additional_unprotection BOOLEAN -- (see Section 2.3.2.5)

Using the security environment referenced by env_handle, complete the decryption and/or verification processing on the data and place any residual output in final_idu_buffer.

2.3.2.10. IDUP_SE_Process_Buffer call

Inputs:
o env_handle ENVIRONMENT HANDLE,
o input_buffer OCTET STRING,

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o output_buffer OCTET STRING
-- may be zero length (depends on underlying mechanism and
-- corresponding Start() call and Protect_Options value)

Using the security environment referenced by env_handle, continue the processing on the data in input_buffer and, if it is available, put any resulting output data in output_buffer. The application calls this routine over and over again with new buffers of data until it has processed all the data buffers of the IDU/PIDU. It then calls the appropriate End() call to complete the processing.

2.3.3. The "EV" Calls

2.3.3.1. IDUP_EV Purpose

The "EV" group of calls provides a simple, high-level interface to underlying IDUP mechanisms when application developers need to deal only with evidence but not with encryption or integrity services. It includes both the single-buffer and multiple-buffer IDU cases and can be used for the generation and verification of evidence tokens embodying several different types of evidences.

The following list of evidence types is supported. This list is by no means exhaustive and it is anticipated that it may be extended in future versions of this specification.

"Non-repudiation of Origin" prevents a message creator’s false denial of creating and sending a message.

"Non-repudiation of Creation" prevents a message creator’s false denial of creating a message.

"Non-repudiation of Sender" prevents a message creator’s false denial of sending a message (that was not necessarily created by the sender).

"Non-repudiation of Delivery" prevents a message recipient’s false denial of having received and looked at the content of a message.

"Non-repudiation of Receipt" prevents a message recipient’s false denial of having received a message (whose content was not necessarily looked at by the recipient).

"Non-repudiation of Approval" prevents a message recipient’s false denial of having approved the content of a received message.

An evidence is provided in the form of a evidence token. Two forms of evidence tokens are supported:

- Tokens including the associated data,

- o Tokens without included data (but with a unique reference to the associated data).

Evidence tokens may be freely distributed. Any possessor of an evidence token (and of the associated data, if not included in the token) can verify the evidence if it has the appropriate credentials which include the definition of security policies (i.e., keys alone do not permit the verification of evidence tokens). Any holder of an evidence token may store it (along with the associated data, if not included in the token) for later verification.

Calls that are specific to the support of evidence include:

* Generate_token, which generates a non-repudiation token using the current environment. The generated token may consist of:

  1 - an evidence token
  2 - a token containing a request for an evidence, which carries information describing which evidence type should be generated by the recipient(s) and sent back to some entities (that may or may not include the sender).
  3 - a token containing an evidence token which is an answer to an evidence that has been previously requested.
  4 - a token including both an evidence and a request for another evidence to be provided.

* Verify_evidence, which verifies the evidence token using the current environment. This operation returns a major_status code which can be used to determine whether the evidence contained in a token is complete (i.e., can be successfully verified (perhaps years) later). If a token's evidence is not complete, the token can be passed to form_complete_pidu to complete it.

Additional useful calls for evidence services include:
* IDUP_Get_token_details (see Section 2.5.3);
* IDUP_Form_Complete_PIDU (see Section 2.4.2).

2.3.3.2. IDUP_EV Parameters

The following parameter bundles are used in the "EV" protection and unprotection sets of calls.

- o Nr_Options PARAMETER BUNDLE
  - o evidence_type INTEGER {
    no_evidence (0)
    -- used when request-only token desired
    proof_of_receipt (1),
    proof_of_delivery (2),

```
                proof_of_approval   (3),
                proof_of_creation   (4),
                proof_of_sender     (5),
                proof_of_origin     (6)
        } OPTIONAL,
     o  evidence_type_oid  OBJECT IDENTIFIER OPTIONAL,
        -- may be used in place of above parameter if OID is known
     o  evidence_validity_duration     INTEGER,
        -- duration_in_minutes
        -- DURATION_HOUR  = 60;
        -- DURATION_DAY   = 1440;
        -- DURATION_WEEK  = 10080;
        -- DURATION_MONTH = 43200;// 30 days
        -- DURATION_YEAR  = 525600;//365 days
     o  mech_indep_encap_req BOOLEAN -- (see Appendix A)

  o  Originator_Information PARAMETER BUNDLE
     o  token_generator_name INTERNAL NAME,
        -- obtained from the credentials of the originator
        -- (e.g., from its public key certificate)
     o  token_generator_role Originator_Role OPTIONAL,
        -- (see Section 2.3.4.1)
     o  protection_time      INTEGER OPTIONAL

  o  Bad_Target_Name  PARAMETER BUNDLE  -- (see Section 2.3.2.2)
     o  bad_targ_name          INTERNAL NAME,
     o  bad_targ_status        INTEGER
        -- a status flag giving the reason for rejection of the
        -- name in bad_targ_name

  o  Target_Info PARAMETER BUNDLE       -- same as in Section 2.3.2.2
     o  targ_names           SET OF INTERNAL NAME,
     o  bad_targ_count       INTEGER,
     o  bad_target_names     SET OF Bad_Target_Name

  o  Request_Features PARAMETER BUNDLE
     o  requested_evidence_type  INTEGER {
                no_evidence         (0), - used when no token desired
                proof_of_receipt    (1),
                proof_of_delivery   (2),
                proof_of_approval   (3), },
     o  nr_req_policy                          OBJECT IDENTIFIER,
     o  evidence_from                          Target_Info,
     o  evidence_to                            Target_Info,
     o  include_received_token_in_evidence     BOOLEAN
```

The following data_type is used in the "EV" protection calls.

- Nr_Operation INTEGER {
  evidence_and_or_evidence_request (1),
  returned_evidence (2) }

2.3.3.3. IDUP_EV major_status codes

The following major_status return codes are defined for the "EV" calls in this section:

- GSS_S_COMPLETE
  -- indicates that the evidence is complete
- IDUP_S_INCOMPLETE
- IDUP_S_MORE_OUTBUFFER_NEEDED
  -- returned (by any EV call) to indicate that there is more output
  -- data than can fit into the supplied buffers. The application
  -- should save the returned data and call again to retrieve the
  -- remaining output.
- IDUP_S_INCONSISTENT_PARAMS
- GSS_S_CREDENTIALS_EXPIRED
- IDUP_S_NO_MATCH
- IDUP_S_NO_ENV
- GSS_S_FAILURE

If Target_Info is used as an input parameter (i.e., if an evidence is being requested), the following major_status return code is also defined:

- IDUP_S_BAD_TARG_INFO

Note for this return code that if one or more of the targets in targ_names cannot be used as a valid recipient of the P-IDU, these names will be returned in bad_targ_names (with associated status codes in bad_targ_status). As long as at least one of the targets can be used, however, this does not cause this call to fail (i.e., the failure code IDUP_S_BAD_TARG_INFO is not returned); it is the caller's choice to discontinue IDU protection if the target set which can be used is unsuitable for the caller's purposes.

2.3.3.4. IDUP_EV_SingleBuffer_Generate call

Inputs:
- env_handle ENVIRONMENT HANDLE,
- nr_operation Nr_Operation,
- Nr_Options PARAMETER BUNDLE,
- idu_buffer OCTET STRING,
- form_complete_pidu BOOLEAN,
  -- if TRUE the implementation will attempt to form a complete PIDU
- include_data_in_token BOOLEAN,

```
      -- if TRUE, data provided in idu_buffer will be included in the
      -- generated token; if FALSE, the data will not be included
   o  Request_Features           PARAMETER BUNDLE
      -- the type of the evidence that is requested;
      -- policy under which the returned evidence should be generated;
      -- the recipients that are supposed to send back an evidence;
      -- the recipients that should receive the requested evidence;
      -- an indicator include_received_token_in_evidence:
      --   if TRUE, the evidence token incorporating the request will be
      --   included in the data for which recipients will generate
      --   evidence; if FALSE, evidence will be generated using only
      --   the data (and not the token incorporating the request).
   o  additional_protection BOOLEAN -- (see Section 2.3.2.4)

   Outputs:
   o  major_status               INTEGER,
   o  minor_status               INTEGER,
   o  token                      OCTET STRING,
   o  evidence_check             OCTET STRING,
      -- present only if an evidence is requested.  Consists of data to
      -- be used to verify the requested token(s) (if any) when they are
      -- received.
```

Description:

This operation generates a non-repudiation token associated with the data passed in an input buffer. Two kinds of operations can be performed (using the Nr_Operation parameter):

a) generating a token that includes either an evidence only, or an evidence request only, or both an evidence and an evidence request;

b) generating a response token for some recipients that includes an evidence generated as a response to a request (in this case the idu_buffer is used to enter the request token that was received).

It is possible to request the generation of complete evidence. This may succeed or fail; if it fails, a subsequent call to Form_Complete_PIDU can be made.

2.3.3.5. IDUP_EV_SingleBuffer_Verify call

```
   Inputs:
   o  env_handle                      ENVIRONMENT HANDLE,
   o  token                           OCTET STRING,
   o  external_idu_buffer             OCTET STRING,
      -- if not present within the token
```

```
o  evidence_check                 OCTET STRING,
   -- present only if the input token is a response to a previous
   -- request for evidence (this parameter is used to validate that
   -- evidence).

Outputs:
o  major_status                   INTEGER,
o  minor_status                   INTEGER,
o  Nr_Options                     PARAMETER BUNDLE,
o  Originator_Information         PARAMETER BUNDLE,
o  Request_Features               PARAMETER BUNDLE,
o  trusted_time_stamping_time     INTEGER OPTIONAL,
   -- present for informational purposes only
o  complete_evidence_before       INTEGER OPTIONAL,
   -- if the major status code that is returned is
   -- IDUP_S_INCOMPLETE, IDUP_Form_Complete_PIDU should be called
   -- with the same token before this time.
   --    This may be required, for example, in order to insure that
   --    the time skew between the evidence generation time and
   --    the trusted time service's countersignature on the evidence
   --    falls within the interval allowed by the current NR policy.
o  complete_evidence_after        INTEGER OPTIONAL,
   -- if the major status code that is returned is
   -- IDUP_S_INCOMPLETE, IDUP_Form_Complete_PIDU should be called
   -- with the same token after this time.
   --    This may be required, for example, to insure that all
   --    authorities involved in generating the evidence have passed
   --    the last time at which the current NR policy allows them to
   --    repudiate their keys.
o  encapsulated_idu_buffer         OCTET STRING
   -- if the IDU was present within the token
o  additional_unprotection BOOLEAN -- (see Section 2.3.2.5)
```

Description:

Verifies the validity and discloses the content of a nr_token.

If the token containing the evidence to be verified was provided to the calling application by a partner responding to the calling application's request, then the calling application must pass the evidence check it received when it generated the request as a parameter along with the token it received from the partner.

Output indicators are provided which give guidance about the time or times at which form_complete_pidu should be called; see the parameter descriptions for explanations of these indicators and their use. Note that the time specified by complete_evidence_before may be earlier than that specified by complete_evidence_after; in this case it will

be necessary to call form_complete_pidu twice.

Because keys can be revoked or declared compromised, the return from verify_evidence cannot in all cases be a definitive "valid" or "invalid"; sometimes "conditionally valid" may be returned, depending upon the policy in use. IDUP_S_INCOMPLETE will be returned, for example, if:

- the interval during which the generator of the evidence may permissibly declare his key invalid has not yet expired (and therefore it is possible that the evidence may be declared invalid in the future), or

- trusted time is required for verification, and the time obtained from the token is not trusted.

2.3.3.6. IDUP_EV_MultiBuffer_StartGenerate call

Inputs:
- o env_handle ENVIRONMENT HANDLE,
- o nr_operation Nr_Operation,
- o Nr_Options PARAMETER BUNDLE,
- o form_complete_pidu BOOLEAN,
- o include_data_in_token BOOLEAN,
- o idu_size INTEGER, -- (see Section 2.3.4.2)
- o Request_Features PARAMETER BUNDLE
- o additional_protection BOOLEAN -- (see Section 2.3.2.4)

Outputs:
- o major_status INTEGER,
- o minor_status INTEGER,
- o initial_pidu_buffer OCTET STRING
  -- may be empty (depends on underlying mechanism)

Description:

Using the security environment referenced by env_handle, initialize the data structures required to begin the generation of a token. The IDU will be supplied in multiple buffers to the IDUP_EV_Process_Buffer call). Two kinds of operations can be performed (using the Nr_Operation parameter) :

a) generating a token that includes either an evidence only, or an evidence request only, or both an evidence and an evidence request.

b) generating a return token for some recipients that includes an evidence generated as a response to a request. In that case the received token will be passed into the subsequent IDUP_EV_Process_Buffer calls. The boolean include_data_in_token is ignored as the output will always be contained in a single token. The Request_Features are ignored in that case at this time in order to keep things simple and to avoid the piggy-backing that is theoretically possible.

It is possible to request the generation of complete evidence. This may succeed or fail; if it fails, a subsequent call to Form_Complete_PIDU can be made.

2.3.3.7. IDUP_EV_MultiBuffer_EndGenerate call

Inputs:
- o env_handle ENVIRONMENT HANDLE

Outputs:
- o major_status INTEGER,
- o minor_status INTEGER,
- o final_pidu OCTET STRING,
- o token OCTET STRING,
- o evidence_check OCTET STRING
  -- present only if an evidence is requested.

Description:

Using the security environment referenced by env_handle, provide the requested token or the final P-IDU. A token will be generated if encapsulation was not requested; otherwise, the final P-IDU is provided.

2.3.3.8. IDUP_EV_MultiBuffer_StartVerify call

Inputs:
- o env_handle ENVIRONMENT HANDLE,
- o token OCTET STRING,
- o evidence_check OCTET STRING,
  -- present only if an evidence has been previously requested.

Outputs:
- o major_status INTEGER,
- o minor_status INTEGER

Description:

Using the security environment referenced by env_handle, initialize the data structures required to begin the process of verifying the token. The P-IDU will be supplied in multiple buffers to the IDUP_EV_Process_Buffer call.

2.3.3.9. IDUP_EV_MultiBuffer_EndVerify call

Input:
- env_handle ENVIRONMENT HANDLE

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- Nr_Options PARAMETER BUNDLE,
- Originator_Information PARAMETER BUNDLE,
- Request_Features PARAMETER BUNDLE,
- trusted_time_stamping_time INTEGER OPTIONAL,
- complete_evidence_before INTEGER OPTIONAL,
- complete_evidence_after INTEGER OPTIONAL,
- idu_buffer OCTET STRING
  -- if the IDU was present within the token
- additional_unprotection BOOLEAN -- (see Section 2.3.2.5)

Description:

Using the security environment referenced by env_handle, complete the verification processing on the data and provide verified output parameters to the caller when the major status code is either:

- GSS_S_COMPLETE or
- IDUP_S_INCOMPLETE

2.3.3.10. IDUP_EV_Process_Buffer call

Inputs:
- env_handle ENVIRONMENT HANDLE,
- input_buffer OCTET STRING

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- output_buffer OCTET STRING
  -- may be zero length (depends on underlying mechanism and
  -- corresponding Generate () call and options
  -- (e.g., data_included_in_token)

Description:

Using the security environment referenced by env_handle, continue the processing on the data in input_buffer and, if it is available, put any resulting output data in output_buffer. The application calls this routine over and over again with new buffers of data until it has processed all the data buffers of the IDU/PIDU. It then calls the appropriate End() call to complete the processing.

2.3.4. The "GP" Calls

The "GP" group of calls provides a powerful interface to flexible and sophisticated combinations of protection and unprotection services. This power and flexibility, however, necessitates a more complex interface than either the SE or the EV calls. Furthermore, such combinations of services are not needed in many of the security mechanisms in common use today (although this is likely to change as time goes on). The GP calls are therefore specified to be OPTIONAL and need not be supported by IDUP-conformant implementations. Note, however, that the structure of IDUP tokens should be such that the SE/EV and GP calls may be used interchangably by the receiver.

2.3.4.1. Parameter Bundles

The concept of "parameter bundles" is used in the calls presented in the following subsections in order to simplify their presentation and clarify their intended purpose and use (note that specific language bindings may or may not use parameter bundles for its actual calling conventions). A parameter bundle is simply a set of closely-related parameters of a call which are either all used by / available to the calling application or all not used by / unavailable to the calling application. These parameters may be all input parameters, all output parameters, or any combination of the two.

An example use envisioned for parameter bundles in a language such as C would be as a structure, where individual parameters in the bundle are structure members. The calling application wishing to use a particular bundle would then allocate the appropriate structure variable, assign the desired input values to the appropriate members, and pass the address of the structure as the bundle "parameter". On output, the values of the appropriate output members may be read. An application not wishing to use a particular bundle (or one which is satisfied with default values for all input parameters of the bundle and which doesn’t care about output values), can pass NULL as the bundle "parameter". From the mechanism implementor’s perspective, if a parameter bundle is not supported (for example, if it represents a security service which is not supported by the implementation), then any non-NULL value passed as the bundle parameter will generate an

error status return code.

[Note that the parameter bundles given below, except where explicitly referenced by the SE and EV calls, are specific to the (optional) GP calls. Thus, these bundles need not be supported by IDUP-conformant implementations if the GP calls are not supported.]

The following parameter bundles are used in the subsequent protection and unprotection sets of calls. A parameter preceded by "(I)" is an input parameter; one preceded by "(O)" is an output parameter; one preceded by neither is an input if the bundle itself is an input and is an output if the bundle itself is an output; one preceded by "(X)" is the opposite: an output if the bundle itself is an input and an input if the bundle itself is an output.

- Mech_Specific_Info PARAMETER BUNDLE
  -- actual parameters included in this bundle are defined by (and
  -- specific to) the underlying mechanism

- Sensitivity PARAMETER BUNDLE,
  -- actual parameters included in this bundle are defined by (and
  -- specific to) the underlying mechanism, but may include
  -- codified values for "Unclassified", "Secret", "Top Secret",
  -- and so on

- Service_Creation_Info PARAMETER BUNDLE
  -- actual parameters included in this bundle are defined by (and
  -- specific to) the underlying mechanism, but it is mandatory
  -- that they include at least service_id and Quality

- Service_Verification_Info PARAMETER BUNDLE
  -- actual parameters included in this bundle are defined by (and
  -- specific to) the underlying mechanism, but it is mandatory
  -- that they include at least service_id and Quality

- Quality PARAMETER BUNDLE
  - qop_algs UNSIGNED INTEGER,
  - qop_algID AlgorithmIdentifier, --overrides qop_algs
  - validity UNSIGNED INTEGER,
    -- protection guaranteed to be valid until time specified
  - policy_id OBJECT IDENTIFIER,
    -- security policy under which protection is/was carried out
  - allow_policy_mapping BOOLEAN,
    -- determines whether mapping between policy IDs is allowed
  - actual_policy_time INTEGER
    -- time at which the above policy rules came into effect

```
   o  Idu_Information PARAMETER BUNDLE,
      o  idu_type_oid OBJECT IDENTIFIER,
      o  idu_type_string OCTET STRING,
      o  idu_title OCTET STRING,
      o  idu_sensitivity Sensitivity,
      o  pidu_type_oid OBJECT IDENTIFIER,
      o  pidu_type_string OCTET STRING,
      o  pidu_title OCTET STRING,
      o  pidu_sensitivity Sensitivity,

   o  Prot_Information PARAMETER BUNDLE,
      o  originator_name INTERNAL NAME,
      o  originator_role Originator_Role,
      o  idu_information Idu_Information,
      o  protection_time INTEGER,

   o  Originator_Role PARAMETER BUNDLE, -- role in organization
      o  domain_name                 INTERNAL NAME OPTIONAL,
      o  role                        PRINTABLE STRING,
      o  role_info_is_authenticated  BOOLEAN
         -- TRUE if info. is authenticated (e.g., inside a cert.)

   o  Special_Conditions PARAMETER BUNDLE,
      o  prot_oper_id INTEGER,
      o  form_complete_pidu BOOLEAN,
         -- input to protection operations for evidence generation
      o  pidu_in_solic_service BOOLEAN,
         -- in protection operations, used as input for service
         -- solicitation to request that receiver include the
         -- received PIDU when generating the response.  In unprot.
         -- operations, used as output to inform receiver that PIDU
         -- should be included when generating the response.
      o  use_trusted_time BOOLEAN,
      o  use_untrusted_time BOOLEAN,
      o  mech_indep_encap_req BOOLEAN -- (see Appendix A)

   o  Bad_Target_Name PARAMETER BUNDLE,
      o  (O) bad_targ_name INTERNAL NAME,
      o  (O) bad_targ_status INTEGER,
             -- a status flag giving the reason for rejection of
             -- the name in bad_targ_name. Specified reasons include:
             --    SYNTAX_INVALID        (0)
             --       the syntax of the name is invalid;
             --    NAME_UNRECOGNIZED     (1)
             --       the name is not recognized;
             --    NAME_AMBIGUOUS        (2)
             --       the name cannot be resolved;
             --    ACCESS_DENIED         (3)
```

```
                --       access to this target is denied;
                --    CERTIFICATE_NOT_FOUND (4)
                --       the encryption certificate of the target could
                --       not be found.
```

- Target_Info PARAMETER BUNDLE,
  - targ_names SET OF INTERNAL NAME,
  - (O) bad_targ_count INTEGER,
  - (O) bad_target_names SET OF Bad_Target_Name,

- General_Service_Data PARAMETER BUNDLE,
  - target_info Target_Info,
  - (X) unencapsulated_token OCTET STRING,
    -- zero length if encapsulation_request is TRUE
  - (O) minor_status INTEGER,

Three types of protection services are defined in IDUP. These are:

1. perform unsolicited service (i.e., act on a locally-generated service request),
2. perform solicited service (i.e., act on a remotely-generated service request), and
3. perform service solicitation (i.e., send a service request to the remote end).

As an originator, applying data confidentiality with data integrity, or data origin authentication with data integrity, or proof of origin evidence is an example of service type 1. As a target, creating a proof of delivery (i.e., receipt) evidence token as the result of a request received from the originator is an example of service type 2. Finally, as an originator, submitting a request that one or more targets return a receipt for the data sent is an example of service type 3.

The first four parameters in the Prot_Service parameter bundle pertain to all service types; the fifth parameter is used if and only if service type 2 is desired; parameters 6-8 are used if and only if service type 3 is desired.

- Prot_Service PARAMETER BUNDLE
  - (I) prot_service_type INTEGER,
  - (I) service_id OBJECT IDENTIFIER,
  - (I) quality Quality, -- NULL specifies default Quality
  - (I) general_service_data General_Service_Data,
  - (I) service_creation_info Service_Creation_Info,
  - (I) service_to SET OF INTERNAL NAME,
  - (O) service_verification_info Service_Verification_Info,
  - (O) service_verification_info_id INTEGER,

Also, three types of unprotection services are defined. These are:

1. receive unsolicited service (i.e., process unrequested remotely-generated service),
2. receive solicited service (i.e., process remotely-generated response to locally-generated request), and
3. receive service solicitation (i.e., process req. from rem. end)

As a target, unprotecting an encrypted message, or verifying the originator's proof of origin is an example of service type 1. As an originator, verifying a proof of delivery which you requested from a target is an example of service type 2. Finally, as a target, receiving a request from an originator for a proof of delivery is an example of service type 3.

The first four parameters in the Unprot_Service parameter bundle pertain to all service types; parameters 5-6 are used if and only if service type 2 is required; parameters 7-8 are used only if service type 3 is required.

- Unprot_Service PARAMETER BUNDLE
  - (O) unprot_service_type INTEGER,
  - (O) service_id OBJECT IDENTIFIER,
  - (O) quality Quality,
    -- actual Quality specified (never NULL)
  - (O) general_service_data General_Service_Data,
  - (O) service_verification_info_id INTEGER,
  - (I) service_verification_info Service_Verification_Info,
  - (O) service_to SET OF INTERNAL NAME,
  - (O) service_creation_info Service_Creation_Info,

2.3.4.2. IDUP_Start_Protect call

Inputs:
- env_handle ENVIRONMENT HANDLE,
- Mech_Specific_Info PARAMETER BUNDLE,
  -- NULL selects the mechanism-defined default values
- Idu_Information PARAMETER BUNDLE,
- Special_Conditions PARAMETER BUNDLE,
- encapsulation_request BOOLEAN,
- single_idu_buffer OCTET STRING,
  -- non-zero length for this buffer means that Protect/End_Protect
  -- won't be called (i.e., entire IDU is contained in this buffer)
- idu_size INTEGER,
  -- size (in bytes) of the IDU to be protected;
  -- may be "-1" signifying "UNKNOWN" (note that some mechanisms
  -- may not support encapsulation in such a case)
- Target_Info PARAMETER BUNDLE,

o Services_to_Perform SET OF Prot_Service,

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o midu_buffer OCTET STRING,
-- zero length if encapsulation_request is TRUE;
-- may be zero length otherwise (depends on underlying mechanism)
o pidu_buffer OCTET STRING,
-- zero length if encapsulation_request is FALSE;
-- may be zero length otherwise (depends on underlying mechanism)

Return major_status codes:
o GSS_S_COMPLETE
-- the protection process can begin (or has completed, if
-- single_idu_buffer has non-zero length).
o IDUP_S_MORE_OUTBUFFER_NEEDED
o GSS_S_CREDENTIALS_EXPIRED
o IDUP_S_NO_ENV
o IDUP_S_ENCAPSULATION_UNAVAIL
o IDUP_S_SERVICE_UNAVAIL
o IDUP_S_REQ_TIME_SERVICE_UNAVAIL
o IDUP_S_UNKNOWN_OPER_ID
o GSS_S_BAD_QOP
o IDUP_S_BAD_TARG_INFO
o GSS_S_FAILURE

Using the security environment referenced by env_handle, initialize the data structures required to begin the process of protecting the IDU buffers. The caller requests specific protection services by supplying the appropriate Prot_Service parameter bundles in Services_to_Perform. Each service is able to return a minor status code to the calling application, if necessary.

The calling application, knowing the size of the IDU it wishes to protect and the buffer size which it has available to it, can choose to input the entire IDU in a single buffer and omit the subsequent IDUP_Protect() and IDUP_End_Protect() calls. Furthermore, the application can request that the resulting M-IDU be encapsulated in the token -- so that the token contains the entire P-IDU -- rather than having it be returned separately in midu_buffer. Encapsulation, however, may not be supported by all underlying mechanisms or implementations; if this is the case, the IDUP_S_ENCAPSULATION_UNAVAIL major status code will be returned and M-IDU will be returned in midu_buffer.

For those mechanisms which allow or require multiple stages of processing, each producing a different aspect of protection for the IDU, the operation identifier prot_oper_id is used to specify which stage is currently being requested by the application. An example where this would be useful is a mechanism which implements the signed Message Security Protocol [MSP]. As another example, a mechanism may choose to do a digital signature in two stages: one for the hashing of the message and another for the signature on the hash. The calling application would therefore use the protection set of calls on the IDU in stage 1 and then use the protection set of calls on the token (from stage 1) in stage 2.

Note that prot_oper_id is simply an integer (1, 2, 3, ..., n, where "n" is the number of stages as defined by the mechanism (typically 1 or 2)). The calling application uses this parameter to indicate to the underlying mechanism whether it wishes to do stage 1 of protection / unprotection processing, or stage 2, and so on. Portable applications may pass "0" to let the mechanism choose the stage (note that mechanism implementers may still iterate when prot_oper_id = 0 (e.g., use output as next input, et cetera).

If one or more of the targets in targ_names cannot be used as a valid recipient of the P-IDU, these names will be returned in bad_targ_names (with associated status codes in bad_targ_status). As long as at least one of the targets can be used, this does not cause this call to fail; it is the caller’s choice to discontinue IDU protection if the target set which can be used is unsuitable for the caller’s purposes. Note that each Prot_Service parameter bundle can also input a list of targ_names; this is used if a separate list is to be used for that service only (the general list of targets is to be used for all services unless overridden in this way).

2.3.4.3. IDUP_Protect call

Inputs:
- env_handle ENVIRONMENT HANDLE,
- input_buffer OCTET STRING,

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- output_buffer OCTET STRING
  -- may be zero length if encapsulation_request was set to TRUE in
  -- IDUP_Start_Protect() (depends on underlying mechanism)

Return major_status codes:
- GSS_S_COMPLETE
- IDUP_S_NO_ENV

o GSS_S_FAILURE

Using the security environment referenced by env_handle, continue the protection processing on the data in input_buffer and, if the underlying mechanism defines this, put any resulting P-IDU/M-IDU data in output_buffer. The application calls this routine over and over again with new buffers of data until it has protected all the data buffers of the IDU. It then calls IDUP_End_Protect() to complete the protection processing.

2.3.4.4. IDUP_End_Protect call

Inputs:
o env_handle ENVIRONMENT HANDLE,

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o Services_to_Perform SET OF Prot_Service,
o final_midu_buffer OCTET STRING,
-- zero length if encapsulation_request was set to TRUE in
-- IDUP_Start_Protect(), in which case pidu is used
o final_pidu_buffer OCTET STRING,
-- zero length if encapsulation_request was set to FALSE in
-- IDUP_Start_Protect(), in which case token and midu are used

Return major_status codes:
o GSS_S_COMPLETE
-- protection has successfully completed and the resulting P-IDU
-- is ready for transfer. If defined by the underlying mechanism,
-- final_midu_buffer will contain any residual M-IDU data.
o IDUP_S_MORE_OUTBUFFER_NEEDED
o IDUP_S_NO_ENV
o GSS_S_FAILURE

Using the security environment referenced by env_handle, complete the protection processing on the data and place the computed output in final_pidu_buffer (or final_midu_buffer and the unencapsulated_token parameter for each Prot_Service). If a service was requested from one or more targets in Start_Protect() - and if this is supported by the underlying mechanism - Service_Verification_Info will hold whatever data is necessary for the mechanism to verify a service returned by a target (unprotector) of the P-IDU. Successful application of IDUP_End_Protect() does not guarantee that the corresponding unprotection set of calls can necessarily be performed successfully when the P-IDU arrives at the target (for example, it may be damaged in transit).

2.3.4.5. IDUP_Start_Unprotect call

```
   Inputs:
   o  env_handle ENVIRONMENT HANDLE,
   o  Mech_Specific_Info PARAMETER BUNDLE,
      -- NULL selects the mechanism-defined default values
   o  single_pidu_buffer OCTET STRING,
      -- non-zero length for this buffer means that IDUP_Unprotect() and
      -- IDUP_End_Unprotect() will not be called (i.e., the entire P-IDU
      -- (if encapsulation is used) or M-IDU (if encap. is not used)
      -- is contained in this buffer)
   o  partial_pidu_buffer OCTET STRING,
      -- may be an arbitrary-sized piece of the full pidu (if the
      -- application's buffer isn't large enough to hold entire pidu).
      -- Used if pidu_buffer will be input a buffer at a time (except
      -- that the final buffer must be passed in final_pidu_buffer
      -- rather than partial_pidu_buffer).  Only one of
      -- single_pidu_buffer and partial(final)_pidu_buffer can have
      -- nonzero length.
   o  final_pidu_buffer OCTET STRING,
   o  Special_Conditions PARAMETER BUNDLE,

   Outputs:
   o  major_status INTEGER,
   o  minor_status INTEGER,
   o  Services_to_Receive SET OF Unprot_Service,
   o  Prot_Information PARAMETER BUNDLE,
   o  single_idu_buffer OCTET STRING,
      -- if this buffer has non-zero length, then service processing has
      -- been completed on the data in single_pidu_buffer
   o  initial_idu_buffer OCTET STRING,
      -- holds any data from partial(final)_pidu_buffer which has been
      -- unprotected; remaining data will be returned by Unprotect and
      -- End_Unprotect as they are called with successive buffers of
      -- pidu
   o  Service_Verification_Info PARAMETER BUNDLE,
      -- used only if target is on "service_to" list in Unprot_Service
   o  service_verification_info_id INTEGER,
      -- used only if target is on "service_to" list in Unprot_Service
```

Return major_status codes:
- GSS_S_COMPLETE
  -- unprotection processing can begin (or has completed, if
  -- single_idu_buffer has non-zero length).
- IDUP_S_INCOMPLETE
  -- used only if single_idu_buffer has non-zero length.
- IDUP_S_MORE_OUTBUFFER_NEEDED
- IDUP_S_MORE_PIDU_NEEDED
- GSS_S_DEFECTIVE_TOKEN
- IDUP_S_INAPPROPRIATE_CRED
- IDUP_S_INCONSISTENT_PARAMS
- IDUP_S_DEFECTIVE_VERIF
- IDUP_S_NO_MATCH
- IDUP_S_SERVICE_UNAVAIL
- IDUP_S_REQ_TIME_SERVICE_UNAVAIL
- IDUP_S_SERV_VERIF_INFO_NEEDED
- GSS_S_CREDENTIALS_EXPIRED
- IDUP_S_NO_ENV
- IDUP_S_UNKNOWN_OPER_ID
- GSS_S_BAD_QOP
  -- the qop_algs value specified in P-IDU for at least one of the
  -- services is unavailable in the local mechanism, so processing
  -- cannot continue.
- GSS_S_BAD_MIC
- IDUP_S_BAD_DOA_KEY
- IDUP_S_BAD_KE_KEY
- IDUP_S_BAD_ENC_IDU
- GSS_S_FAILURE

Using the security environment referenced by env_handle, initialize the data structures required to begin the process of unprotecting a P-IDU. The caller will be alerted as to which services were applied to the P-IDU in the returned Services_to_Receive set of parameters.

If encapsulation was not used by the originator, it is the receiving application's responsibility to separate the received P-IDU into a M-IDU and one or more unencapsulated_token buffers (the latter being input in separate Unprot_Service bundles in the Services_to_Receive parameter). These unencapsulated_token buffers should be input before the M-IDU (i.e., in IDUP_Start_Unprotect) or after the M-IDU (i.e., in IDUP_End_Unprotect) as appropriate; this order may be dictated, for example, by their placement in the in-coming message.

If unprotection will be applied more than once to a given P-IDU, it is the responsibility of the calling application to remember if a service solicitation has been responded to previously (i.e., if the requested service has already been generated / sent for that P-IDU) and thus ignore subsequent solicitations on unprotect.

The time flags indicate whether to consult trusted, untrusted, or no time (if both flags are FALSE) during the unprotection operation. If the current time is not to be checked, then unprotection may be successful even if the protector's key has expired since the P-IDU was generated (that is, if the Validity period -- as specified in the Quality parameter bundle -- has expired).

If the underlying mechanism supports it and if this information is contained in the P-IDU, information regarding the originator (that is, the entity which used the protection set of calls to generate this P-IDU) is returned in the Prot_Information parameter bundle.

2.3.4.6. IDUP_Unprotect call

Inputs:
- env_handle ENVIRONMENT HANDLE,
- input_buffer OCTET STRING

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- output_buffer OCTET STRING

Return major_status codes:
- GSS_S_COMPLETE
- IDUP_S_NO_ENV
- GSS_S_FAILURE

Using the security environment referenced by env_handle, continue the unprotection processing on the data in input_buffer, putting any resulting IDU data in output_buffer (if required).

2.3.4.7. IDUP_End_Unprotect call

Inputs:
- env_handle ENVIRONMENT HANDLE,

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- Prot_Information PARAMETER BUNDLE,
- Services_to_Receive SET OF Unprot_Service,
- final_idu_buffer OCTET STRING,
- Service_Verification_Info PARAMETER BUNDLE,
  -- used only if target is on "service_to" list in Unprot_Service
- service_verification_info_id INTEGER,
  -- used only if target is on "service_to" list in Unprot_Service

Return major_status codes:
- GSS_S_COMPLETE
  -- residual IDU data will be returned in final_idu_buffer.
- IDUP_S_INCOMPLETE
- IDUP_S_MORE_OUTBUFFER_NEEDED
- GSS_S_BAD_MIC
- IDUP_S_BAD_DOA_KEY
- IDUP_S_BAD_KE_KEY
- IDUP_S_BAD_ENC_IDU
- IDUP_S_NO_ENV
- GSS_S_FAILURE

Using the security environment referenced by env_handle, complete the unprotection processing on the data and return the appropriate status code. If there is any residual IDU data it will be returned in final_idu_buffer.

If the IDUP_S_INCOMPLETE major status value is returned, all output parameters are conditionally valid; the unprotection set of functions will have to be called again (perhaps with a complete P-IDU, as produced by IDUP_Form_Complete_PIDU) in order to get valid values for all parameters. "Conditional validity" may arise, for example, if all relevant certificates verify correctly, but it is not yet past the time up to which the current policy allows the authorities involved to repudiate their keys.

If the underlying mechanism supports it and if this information is contained in the token, information regarding the originator (that is, the entity which used the protection set of calls to generate this token) is returned in the Prot_Information parameter bundle. This information may or may not be omitted if it was returned by the IDUP_Start_Unprotect() call.

Note that, unlike GSS-API, IDUP-GSS-API does not incorporate the concept of error tokens transferred between sender and recipient since the protection and unprotection of an IDU may be separated by an indefinite amount of time and may or may not be performed by the same entity.

2.4. Special-Purpose Calls

2.4.1. Relationship to GSS-API

The special-purpose call described in this section has no analog in GSS-API [RFC-2078]. This call is used to complete a P-IDU (that is, to generate a P-IDU which can be unprotected successfully with no additional data at any time during its validity period). This call may not be supported by all underlying IDUP mechanisms or

implementations.

2.4.2. IDUP_Form_Complete_PIDU call

Inputs:
- o env_handle ENVIRONMENT HANDLE,
- o single_pidu_buffer OCTET STRING,
- o partial_pidu_buffer OCTET STRING,
  -- an arbitrary-sized piece of the full pidu token. Used if pidu
  -- will be input a buffer at a time (except that the final buffer
  -- must be passed in final_pidu_buffer rather than
  -- partial_pidu_buffer). Only one of single_pidu_buffer and
  -- partial(final)_pidu_buffer can have nonzero length.
- o final_pidu_buffer OCTET STRING,

Outputs:
- o major_status INTEGER,
- o minor_status INTEGER,
- o pidu_token_out OCTET STRING -- the augmented PIDU; may be complete
- o call_again_before INTEGER,
- o call_again_after INTEGER,
- o trusted_time_stamping_time INTEGER -- for information only

Return major_status codes:
- o GSS_S_COMPLETE
- o IDUP_S_MORE_OUTBUFFER_NEEDED
- o IDUP_S_INCOMPLETE
  -- generation of the P-IDU is not yet complete. The application
  -- should call this function again before the time given in
  -- call_again_before (if not NULL), or after the time given in
  -- call_again_after (if not NULL), or both (if neither are NULL).
- o IDUP_S_INCONSISTENT_PARAMS
- o IDUP_S_SERVICE_UNAVAIL
- o GSS_S_DEFECTIVE_TOKEN
- o GSS_S_FAILURE

Form_Complete_PIDU is used primarily by the evidence services; in particular, when the evidence token itself does not contain all the data required for its verification and it is anticipated that some of the data not stored in the token may become unavailable during the interval between generation of the evidence token and verification unless it is stored in the token. The Form_Complete_PIDU operation gathers the missing information and includes it in the token so that verification can be guaranteed to be possible at any future time.

This call generates a PIDU which can be unprotected successfully with no additional data at any time during its validity period. [For background information on the notion of "complete" evidence, see

"CORBA Security Service v1.2 Draft D02", 18 June 1997.]

Using the security environment referenced by env_handle, complete the generation of a P-IDU token and return the appropriate status value along with the completed token (if available). Such a call may be used, for example, for the purpose of batch evidence generation on an "evidence server". A local machine may be able to use the protection set of calls to fill out most of an evidence token and then send a number of these to a batch processor which forms the complete evidence tokens (perhaps by adding a certification path, or a timestamp and signature from a timestamping authority). As another example, on the receiving end an application may make such a call in order to collect all the information necessary to unprotect a P-IDU (such as all relevant certificates and Certificate Revocation Lists); this will ensure that the calls to the unprotection set of operations will be entirely local (i.e., can be performed off-line) and fast.

Note that the complete P-IDU generated will be formed using trusted time if this is available in the environment referenced by env_handle and will use untrusted time or no time otherwise (depending on what is available).

2.5. Support calls

2.5.1. Relationship to GSS-API

Support calls in IDUP-GSS-API are to be understood and used as described in GSS-API [RFC-2078]. The calls described in Section 2.4 of GSS-API (including all associated parameters) are unchanged. The following additional calls are specified for IDUP-GSS-API.

2.5.2: IDUP_Acquire_cred_with_auth call

Inputs:

- desired_name INTERNAL NAME,
  -- NULL requests locally-determined default
- authenticator OCTET STRING
  -- string which authenticates the caller claiming to be
  -- desired_name
- lifetime_req INTEGER,
  -- in seconds; 0 requests default
- desired_mechs SET OF OBJECT IDENTIFIER,
  -- empty set requests system-selected default
- cred_usage BIT STRING
  -- actual values which can be used currently correspond to those
  -- given in Section 2.1.1 (i.e.,
  -- ENCRYPT_ONLY 8
  -- DECRYPT_ONLY 16

```
      --    SIGN_ONLY      32
      --    VERIFY_ONLY    64
      -- with the values logically OR'ed together in any desired
      -- combination to restrict credential usage; OR'ing all values
      -- results in NO_RESTRICTION).
      -- Future possible values for this parameter are for further
      -- study (note that the type of this parameter is BIT STRING
      -- (rather than INTEGER as in GSS_Acquire_cred) to facilitate
      -- such future expansion).
```

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- output_cred_handle CREDENTIAL HANDLE,
- actual_mechs SET OF OBJECT IDENTIFIER,
- actual_cred_usage BIT STRING,
- lifetime_rec INTEGER
  -- in seconds, or reserved value for INDEFINITE

This call (which need not be supported by all underlying mechanisms or implementations) is identical to the GSS_Acquire_cred call, with the exception of the added input parameter "authenticator" and the added output parameter "actual_cred_usage". The authenticator (typically a password, pass-phrase, or PIN) is used to authenticate the caller claiming to be desired_name to the underlying GSS (or mechanism) code. The actual_cred_usage specifies the actual uses available for these credentials; it is up to the caller to determine if this is sufficient for its purposes.

Implementations that are able to authenticate the caller in some other way are encouraged to use the GSS_Acquire_cred call; those having no other means available to them, or wishing to explicitly authenticate the caller at the time of credential acquisition, should use the IDUP_Acquire_cred_with_auth call (if supported).

Note that the return major status codes for this call are identical to those given for the GSS_Acquire_cred call. If the authentication fails (e.g., the wrong authenticator is supplied for the given desired_name), the major status GSS_S_FAILURE is returned (along with an appropriate minor status code).

2.5.3. IDUP_Get_token_details call

Inputs:

```
o  token                  OCTET STRING,
-- all the data to be returned shall be within the first 4 KB of
-- the token; hence, a single call is needed. It is not necessary
-- to provide the entire token when the token includes the IDU.
```

```
   o  mech_type              SET OF OBJECT IDENTIFIER
   -- input if known (typically SET will contain a single member)

   Outputs:
   o  major_status                        INTEGER,
   o  minor_status                        INTEGER,
   o  actual_mech_type                    OBJECT IDENTIFIER,
   o  data_included_in_token              BOOLEAN,
      -- true if the data is encapsulated
   o  idu_size                            INTEGER,
   o  has_SE_protection                   BOOLEAN,
   o  has_EV_protection                   BOOLEAN,
   o  PIDU_Information                    PARAMETER BUNDLE,
   o  nr_policy                           OBJECT IDENTIFIER,
      -- this and subsequent parameters pertain only to evidence tokens
   o  Nr_Options                          PARAMETER BUNDLE,
   o  Originator_Information              PARAMETER BUNDLE,
   o  time_stamping_time                  INTEGER  OPTIONAL
   o  Request_Features                    PARAMETER BUNDLE,
      -- describes the included request, if any.
   o  requested_evidence_back             BOOLEAN,
      -- true if this is an evidence generated in response to a
      -- previously-sent request
   o  evidence_check                      OCTET STRING,
      -- meaningful if the boolean above is true

   Return major_status codes:
   o  GSS_S_COMPLETE
      -- input_token could be parsed for all relevant fields.
   o  GSS_S_CREDENTIALS_EXPIRED
   o  GSS_S_DEFECTIVE_TOKEN
      -- the mechanism type could be parsed, but either the other fields
      -- could not be determined from the input_token, or their values
      -- did not correspond to valid values for that mechanism.
   o  GSS_S_FAILURE
      -- the mechanism type was missing or corrupted.
```

IDUP_Get_token_details() is used to return to an application the attributes that correspond to a given input token. Since IDUP-GSS-API tokens are meant to be opaque to the calling application, this function allows the application to determine information about the token without having to violate the opaqueness intention of IDUP. Of primary importance is the mechanism type, which the application can then use as input to the IDUP_Establish_Env() call in order to establish the correct environment in which to have the token processed.

If all tokens are framed as suggested in Section 3.1 of [RFC-2078] (mandated in the Kerberos V5 GSS mechanism [RFC 1964] and in the SPKM GSS Mechanism [RFC 2025]), then any mechanism implementation should be able to return the mech_type parameter for any uncorrupted input token. If the mechanism implementation whose IDUP_Get_token_details() function is being called does recognize the token, it can return any further relevant information in the other token attributes, as specified. In particular, this function can set has_SE_protection if the SE calls may be used to unprotect it, or has_EV_protection if the EV calls may be used to unprotect it, or both if both kinds of protection have been applied (so that SE or EV calls may be used in any order for unprotection) [note that GP calls, when supported, should be usable for unprotection of any IDUP token].

IDUP_Get_token_details (which need not be supported by all underlying mechanisms or implementations) gives only a hint about the content of the token, there is no integrity check of any kind performed. Regardless of the token type, it is possible to check that this information is correct only by doing a proper unprotection of the token. It is recommended that IDUP callers supply a token buffer at least 4 KB in length in order to ensure that the desired data can easily flow across this interface.

The OID of the mechanism and whether the token contains the associated data is returned. In addition the size of the associated data, whether inside or outside the token, is included if known. [Note: data size will typically be unknown if the data was protected using multibuffer calls. A value of "-1" may be used to indicate "UNKNOWN".]

When the input token contains only an evidence generated spontaneously, the following is returned:
- the evidence type;
- the Non-Repudiation policy under which the evidence was generated;
- the name of the generator of the evidence;
- the date and time when the evidence was generated (if available);
- the date and time when it was time stamped (if available).

When the input token contains only an evidence generated in response to a request from another entity, the following additional information is returned:
- an indicator to state that this evidence relates to a request;
- a string significant for the requester that will allow him to check whether the answer corresponds to the requested evidence.

When the input token only contains a request, the following is returned:
- the name of the requestor of the evidence,

- the date and time when the request was made,
- the evidence type to send back,
- the non-repudiation policy under which the evidence to send back should be generated,
- the names of the recipients which should generate and distribute the requested evidence,
- the names of the recipients to whom the requested evidence should be sent after it has been generated.

When the input token contains both evidence and a request, an indicator is returned describing whether the new evidence should be generated using only the data in the input token, or using both the data and the evidence in the input token.

When the input token contains only CONF and DOA services, the PIDU_Information bundle is returned. Other relevant parameters (such as idu_size and time_stamping_time) may also be returned if this data is available.

2.5.4. IDUP_Get_policy_info call

Inputs:
o policy_id OBJECT IDENTIFIER

Outputs:
o major_status INTEGER,
o minor_status INTEGER,
o policy_version INTEGER,
o policy_effective_time INTEGER,
o policy_expiry_time INTEGER,
o supported_services SET OF Service_Descriptor,
  -- services supported by this particular policy_id (equal to the
  -- intersection of the services supported by the mechanisms
  -- listed in supported_mechanisms)
o supported_mechanisms SET OF Mechanism_Descriptor
  -- mechanisms supported by this particular policy_id

Return major_status codes:
o GSS_S_COMPLETE
  -- policy_id recognized; all relevant fields have been returned.
o GSS_S_FAILURE
  -- the policy_id was not recognized.

This call (which need not be supported by all underlying mechanisms or implementations) allows the application to retrieve information pertaining to a given policy_id. Policies define the following:
- rules for the protection of IDUs, such as trusted third parties which may be involved in P-IDU generation, the roles in

which they may be involved, and the duration for which the generated P-IDU is valid;

- rules for the unprotection of P-IDUs, such as the interval during which a trusted third party may legitimately declare its key to have been compromised or revoked; and

- rules for adjudication, such as which authorities may be used to adjudicate disputes.

The policy itself may be used by an adjudicator when resolving a dispute. For example, the adjudicator might refer to the policy to determine whether the rules for generation of the P-IDU have been followed.

The following parameter bundles are associated with this call.

```
o  Service_Descriptor PARAMETER BUNDLE,
   o  service_type OBJECT IDENTIFIER,
   o  service_validity_duration INTEGER,
   o  must_use_trusted_time BOOLEAN

o  Mechanism_Descriptor PARAMETER BUNDLE,
   o  mechanism_type OBJECT IDENTIFIER,
   o  Authority_List PARAMETER BUNDLE,
   o  maximum_time_skew INTEGER
      -- maximum permissible difference between P-IDU generation
      -- time and the time of countersignature from a time
      -- service (if required).  This parameter is unused if
      -- trusted time is not required.

o  Authority_List PARAMETER BUNDLE,
   o  authority_name INTERNAL NAME,
   o  authority_role OCTET STRING,
   o  last_revocation_check_offset INTEGER
      -- may be 0, greater than 0, or less than 0.  The value of
      -- this parameter is added to P-IDU generation time to
      -- get latest time at which the mechanism will check to
      -- see if this authority's key has been revoked.
```

An example of the use of the last parameter in Authority_List is as follows. If an authority has a defined last_revocation_check_offset of negative one hour, then all revocations taking effect earlier than one hour before the generation of a P-IDU will render that P-IDU invalid; no revocation taking place later than one hour before the generation of the P-IDU will affect the P-IDU's validity.

Note that both the maximum_time_skew and the last_revocation_check_offset values are given in minutes.

2.5.5. IDUP_Cancel_multibuffer_op call

Inputs:
- env_handle ENVIRONMENT HANDLE,

Outputs:
- major_status INTEGER,
- minor_status INTEGER,

Return major_status codes:
- GSS_S_COMPLETE
  -- operation cancelled; state purged.
- GSS_S_FAILURE
  -- unable to cancel operation; state retained.

This call (which need not be supported by all underlying mechanisms or implementations) allows the application to cancel a multibuffer operation prior to normal completion (e.g., subsequent to calling Start_operation and zero or more Process_operation, but prior to calling End_operation). When successful, this call purges any internal state information which would have been used to continue processing for the full set of multibuffer calls.

3. Related Activities

In order to implement the IDUP-GSS-API atop existing, emerging, and future security mechanisms, the following is necessary:

- object identifiers must be assigned to candidate IDUP-GSS-API mechanisms and the name types which they support; and

- concrete data element (i.e., token and parameter bundle) formats must be defined for candidate mechanisms.

Calling applications must implement formatting conventions which will enable them to distinguish IDUP-GSS-API P-IDUs from other IDUs in their environment.

Concrete language bindings are required for the programming environments in which the IDUP-GSS-API is to be employed.

4. Acknowledgments

Many thanks are due to Tim Moses and Dhanya Thakkar of Entrust Technologies, Denis Pinkas of Bull, and David Kurn of Tandem Computers for a number of helpful comments and contributions.

5. Security Considerations

Security issues are discussed throughout this memo.

6. REFERENCES

[MSP] U.S. National Security Agency, "Message Security Protocol", Secure Data Network System SDN.701, March 1994.

[RFC-1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

[RFC-2078] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997..

[RFC 1964] Linn, J, "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC 2025] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[ISO/IEC] 2nd ISO/IEC CD 13888-1, "Information technology - Security techniques - Non-repudiation - Part 1: General Model", ISO/IEC JTC 1/SC 27, May 30, 1995

7. Author’s Address

Carlisle Adams
Entrust Technologies
750 Heron Road, Suite E08,
Ottawa, Ontario, CANADA K1V 1A7

Phone: +1 613.247.3180
EMail: cadams@entrust.com

APPENDIX A: MECHANISM-INDEPENDENT TOKEN FORMAT

This appendix specifies the use, for IDUP-GSS-API tokens, of the mechanism-independent level of encapsulating representation for tokens given in Section 3.1 of GSS-API [RFC-2078]. The representation given there incorporates an identifier of the mechanism type to be used when processing the associated tokens. Use of that octet format is recommended to the designers of IDUP-GSS-API implementations based on various mechanisms so that tokens can be interpreted unambiguously at IDUP-GSS-API peers. It is recognized, however, that for interoperability purposes with peers not using IDUP for specific IDU protection/unprotection protocols, the encapsulating representation may need to be omitted. (In such a case it is necessary that the underlying mechanism provides some sort of internal or external identification that allows it to recognize its own tokens.) When the mechanism-independent level of encapsulating representation is not desired, callers SHOULD set mech_indep_encap_req to FALSE (note that some underlying mechanisms may default this parameter to FALSE).

For purely descriptive purposes, the following simple ASN.1 structure is used to illustrate the structural relationships among token and tag objects. For interoperability purposes, token and tag encoding shall be performed using the concrete encoding procedures described in Section 3.1 of GSS-API [RFC-2078].

```
-- top-level token definition to frame different mechanisms

IDUP-GSS-API DEFINITIONS ::=
BEGIN
MechType ::= OBJECT IDENTIFIER

Token ::= [APPLICATION 0] IMPLICIT SEQUENCE {
        thisMech MechType,
        token ANY DEFINED BY thisMech
           -- contents mechanism-specific
        }
END
```

APPENDIX B: EXAMPLES OF IDUP USE

This appendix provides examples of the use of IDUP to do IDU protection and unprotection. It should not be regarded as constrictive to implementations or as defining the only means through which IDUP-GSS-API functions can be realized with particular underlying technology, and does not demonstrate all IDUP-GSS-API features.

Most of the examples below only illustrate the use of CONF/DOA protection services. Note that when both CONF/DOA and Evidence services are required, calling applications may use a series of SE and EV calls, or may use the GP calls (when these are supported). Using the former approach implies multiple calls (e.g., the SE calls are used to protect some data and the resulting token is then input to the EV calls to add evidence information), but some callers may find this to be more attractive than coding to the GP calls because of the simpler SE/EV interface. Depending upon the underlying mechanism, the series of SE/EV calls may result in a single token that can be unprotected using the SE and EV calls in any order (for example, because it is a single ASN.1 SEQUENCE that incorporates all the specified protection services at one level), or the series may result in a token that can only be unprotected in the reverse order of protection (for example, because each SE/EV output token was effectively embedded in the token of the subsequent call). The IDUP_Get_token_details call can assist callers in determining how to unprotect any received token.

B.1. Simple Mechanism, Single Buffer

To illustrate the simplest possible case, consider an underlying IDUP mechanism which does straightforward encryption/decryption and signing/verification only using public-key techniques; none of the other possible services, such as creation of proof-of-origin evidence, requests for proof-of-delivery evidence, or use of trusted time, are supported. PEM[RFC-1421] is one example of a mechanism which fits this description. Furthermore (again for simplicity), assume that encapsulation is chosen by the calling application during IDU protection.

Such a mechanism would likely use the "SE" set of IDUP-GSS-API calls. The following parameter bundle uses and defaults would therefore be specified in the relevant IDUP mechanism document.

```
   SENDER:

   Set
      env_handle                = environment handle in use;
      idu_buffer                = data buffer;
      Target_Info.targ_names    = receiver names;
      Protect_Options           = as necessary;

   Call
      IDUP_SE_SingleBuffer_Protect() with above input parameters

   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         Target_Info.Bad_Targ_Name,
      (as required) for more detailed information.

   Send
      Output parameter pidu_buffer to receiver.

   RECEIVER (any parameters not listed below are given the value NULL):

   Set
      env_handle         = environment handle in use;
      pidu_buffer        = received data buffer;

   Call
      IDUP_SE_SingleBuffer_Unprotect() with above input parameters
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
      (as required) for more detailed information

   Utilize
      PIDU_Information.Protect_Options.Protect_Operation,
         (to determine which services were applied by the originator)
      PIDU_Information.Protect_Options.sign_qop_alg / enc_qop_alg,
         (to determine the corresponding qualities of the services)
      Prot_Information.originator_name,
         (to determine the name of the originator)
      Prot_Information.protection_time,
         (to determine when the IDU was protected)
      idu_buffer
         (to retrieve the unprotected data).
```

B.2. Simple Mechanism, Single Buffer (Again)

To illustrate a slight variation on the simplest possible case, assume that everything is as in the previous scenario except that the "GP" calls are used.

The following parameter bundle uses and defaults would therefore be specified in the relevant IDUP mechanism document.

Mech_Specific_Info
- NOT USED (the only acceptable input, therefore, is NULL)

Idu_Sensitivity
- NOT USED (the only acceptable input, therefore, is NULL)

Service_Creation_Info
- NOT USED (the only acceptable input, therefore, is NULL)

Service_Verification_Info
- NOT USED (the only acceptable input, therefore, is NULL)

Quality
- the qop_algs parameter must be supported, with a suitable DEFAULT value specified;
- suitable DEFAULT values for validity, policy_id, and allow_policy_mapping must be specified (it may be an implementation option as to whether these parameters are explicitly modifiable by the calling application, or whether NULLs are the only acceptable input)

Idu_Information
- the idu_type parameter must have a value representing a suitable IDU type (for example, in PEM a value representing the string "RFC822" or some other valid "Content-Domain" would be used), with a suitable DEFAULT value specified;
- the idu_title parameter is NOT USED (the only acceptable input, therefore, is NULL)

Prot_Information
- the originator_name and idu_type (in Idu_Information) parameters are read from the encapsulating information and output by IDUP_Start_Unprotect;
- all other parameters are NOT USED (and therefore NULL)

Special_Conditions
- NOT USED (the only acceptable input, therefore, is NULL)

Target_Info
- this bundle is used as described in IDUP; no DEFAULT values are specified

General_Service_Data
- the unencapsulated_token parameter is used if encapsulation_request is FALSE;
- the minor_status parameter is used to return minor status values as specified by the mechanism document

Prot_Service
- the prot_service_type parameter may have a value of "1" ("perform unsolicited service") or NULL (which specifies the DEFAULT value of "1");
- the service_id parameter must have a value representing "PER_CONF" or "PER_DOA";
- the parameters Service_Creation_Info, service_to, Service_Verification_Info, and service_verification_info_id are NOT USED (and therefore NULL)

Unprot_Service
- the unprot_service_type parameter will always have a value of "1" ("receive unsolicited service");
- the service_id parameter will have a value representing "REC_CONF" or "REC_DOA";
- the parameters service_verification_info_id, Service_Verification_Info, service_to, and Service_Creation_Info, are NOT USED (and therefore NULL)

Assuming that the calling application has only a single buffer of data to protect/unprotect, the following sequence of operations must be performed by the sender and receivers (subsequent to environment establishment).

SENDER (any parameters not listed below are given the value NULL):

Set

```
env_handle                          = environment handle in use;
encapsulation_request               = TRUE;
single_idu_buffer                   = data buffer;
Target_Info.targ_names              = receiver names;
P_Services.Prot_Service_1.service_id = PER_CONF;
P_Services.Prot_Service_2.service_id = PER_DOA;
```

Call

IDUP_Start_Protect() with above input parameters

```
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         Target_Info.bad_targ_names / Target_Info.bad_targ_status,
         P_Services.Prot_Service_1.General_Service_Data.minor_status,
         P_Services.Prot_Service_2.General_Service_Data.minor_status
      (as required) for more detailed information.

   Send
      Output parameter pidu_buffer to receiver.


   RECEIVER (any parameters not listed below are given the value NULL):

   Set
      env_handle         = environment handle in use;
      single_pidu_buffer = received data buffer;

   Call
      IDUP_Start_Unprotect() with above input parameters
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         R_Services.Unprot_Service_1.General_Service_Data.minor_status,
         R_Services.Unprot_Service_2.General_Service_Data.minor_status
      (as required) for more detailed information

   Utilize
      R_Services.Unprot_Service_1/2.service_id,
         (to determine which services were applied by the originator)
      R_Services.Unprot_Service_1/2.Quality,
         (to determine the corresponding qualities of the services)
      Prot_Information.originator_name,
         (to determine the name of the originator)
      single_idu_buffer
         (to retrieve the unprotected data).
```

B.3. Simple Mechanism, Multiple Buffers

To illustrate the next step up in complexity, consider the use of the simple IDUP mechanism described in B.2 above with multiple data buffers. In particular, consider the case in which a large data file is to be signed. For this example, assume that the calling application does not wish to use encapsulation.

Note that the parameter bundle uses and defaults are as specified in B.2. above.

```
   SENDER (any parameters not listed below are given the value NULL):

   Set
      env_handle                          = environment handle in use;
      encapsulation_request               = FALSE;
      P_Services.Prot_Service.service_id  = PER_DOA;

   Call
      IDUP_Start_Protect() with above input parameters
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         P_Services.Prot_Service.General_Service_Data.minor_status
      (as required) for more detailed information.

   For each buffer of input data:
      Set
         input_buffer = buffer
      Call
         IDUP_Protect() with above input parameter
      Check
         major_status.  If not GSS_S_COMPLETE, check
            minor_status

   Call
      IDUP_End_Protect()
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         P_Services.Prot_Service.General_Service_Data.minor_status
      (as required) for more detailed information.

   Send
      P_Services.Prot_Service.General_Service_Data.unencapsulated_token,
      and the file for which the signature was calculated (if required),
      to receiver.

   RECEIVER (any parameters not listed below are given the value NULL):

   Set
      env_handle            = environment handle in use;
      R_Services.Unprot_Service_1.General_Service_Data.
      unencapsulated_token  = received unencapsulated token;

   Call
      IDUP_Start_Unprotect() with above input parameters
   Check
      major_status.  If not GSS_S_COMPLETE, check
```

```
         minor_status,
         R_Services.Unprot_Service_1.General_Service_Data.minor_status,
      (as required) for more detailed information

   For each buffer of input data:
      Set
         input_buffer = buffer
      Call
         IDUP_Unprotect() with above input parameter
      Check
         major_status.  If not GSS_S_COMPLETE, check
            minor_status

   Call
      IDUP_End_Unprotect()
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         R_Services.Unprot_Service_1.General_Service_Data.minor_status,
      (as required) for more detailed information.

   Utilize
      R_Services.Unprot_Service_1.service_id,
         (to determine which service was applied by the originator; note
          that Unprot_Service_2 will have NULL in unprot_service_type
          to indicate that it is not used)
      R_Services.Unprot_Service_1.Quality,
         (to determine the corresponding quality of the service)
      Prot_Information.originator_name, (from IDUP_Start_Unprotect)
         (to determine the name of the signer)
      major_status (from IDUP_End_Unprotect)
         (to determine pass/fail status of signature verification).
```

B.4. More Sophisticated Mechanism, Small Application Buffers

To illustrate a higher level of complexity, consider the use of a more sophisticated IDUP mechanism and a calling application with small data buffers. In particular, consider the case in which a very small e-mail message is to be encrypted for a relatively large receiver list (R), some subset of whom (r) will be asked to send proofs of receipt of the message to some other subset (L) (which includes the originator). So that the example is not unnecessarily complicated, assume again that the originating application uses encapsulation.

The uses and defaults for the various parameter bundles for this mechanism would be specified in the relevant IDUP mechanism document as follows.

Mech_Specific_Info
- NOT USED (the only acceptable input, therefore, is NULL)

Idu_Sensitivity
- NOT USED (the only acceptable input, therefore, is NULL)

Service_Creation_Info
- used to create "proof of delivery" evidence (but actual structure is opaque to calling application)

Service_Verification_Info
- used to verify "proof of delivery" evidence (but actual structure is opaque to calling application)

Quality
- the qop_algs parameter must be supported, with a suitable DEFAULT value specified;
- suitable DEFAULT values for validity, policy_id, and allow_policy_mapping must be specified (it may be an implementation option as to whether these parameters are explicitly modifiable by the calling application, or whether NULLs are the only acceptable input)

Idu_Information
- the idu_type parameter must have a value representing a suitable IDU type, with a suitable DEFAULT value specified;
- the idu_title parameter must have a value representing a suitable IDU title, with a suitable DEFAULT value specified

Prot_Information
- the originator_name, protection_time, and idu_type / idu_title (in Idu_Information) parameters are read from the contained header information and output by IDUP_Start_Unprotect;

Special_Conditions
- the parameter prot_oper_id is NOT USED (the only acceptable input, therefore, is NULL);
- trusted or untrusted time may be selected by the calling application, with a suitable DEFAULT value specified

Target_Info
- this bundle is used as described in IDUP; no DEFAULT values are specified

General_Service_Data
- the unencapsulated_token parameter is used if encapsulation_request is FALSE;
- the minor_status parameter is used to return minor status values

as specified by the mechanism document

Prot_Service
- the prot_service_type parameter may have a value of "1" ("perform unsolicited service"), "2" ("perform solicited service"), "3" (perform service solicitation), or NULL (which specifies the DEFAULT value of "1");
- the service_id parameter must have a value representing "PER_CONF", "PER_DOA", "PER_POO", or "PER_POD";
- the parameters Service_Creation_Info, service_to, Service_Verification_Info, and service_verification_info_id are used when required by the IDUP operation

Unprot_Service
- the unprot_service_type parameter may have a value of "1" ("receive unsolicited service"), "2" ("receive solicited service"), or "3" (receive service solicitation);
- the service_id parameter will have a value representing "REC_CONF", "REC_DOA", "REC_POO", or "REC_POD";
- the parameters service_verification_info_id, Service_Verification_Info, service_to, and Service_Creation_Info, are used when required by the IDUP operation

SENDER (any parameters not listed below are given the value NULL):

Set

```
env_handle                          = environment handle in use;
Idu_Information.idu_type            = value for "e-mail document";
Idu_Information.idu_title           = "Contract 1234";
Special_Conditions.use_trusted_time = TRUE;
encapsulation_request               = TRUE;
single_idu_buffer                   = very small e-mail message;
Target_Info.targ_names              = receiver names (R);
Prot_Service_1.prot_service_type    = "1";
Prot_Service_1.service_id           = PER_CONF;
Prot_Service_2.prot_service_type    = "3";
Prot_Service_2.service_id           = PER_POD;
Prot_Service_2.General_Service_Data.Target_Info.targ_names
                                    = "receipts from" list (r);
Prot_Service_2.service_to           = "receipts to" list (L);
P_Services.Prot_Service_1           = Prot_Service_1;
P_Services.Prot_Service_2           = Prot_Service_2;
```

Call

IDUP_Start_Protect() with above input parameters

```
   Check
      major_status.  If not GSS_S_COMPLETE,
         while major_status == IDUP_S_MORE_OUTBUFFER_NEEDED
            Save
               pidu_buffer,
            Call
               IDUP_Start_Protect() (to get next portion of pidu_buffer)
         Check
            major_status,
            minor_status,
            Target_Info.bad_targ_names / Target_Info.bad_targ_status,
            P_Services.Prot_Service_1.General_Service_Data.minor_status,
            P_Services.Prot_Service_2.General_Service_Data.minor_status
         (as required) for more detailed information.

   Save
      Prot_Service_2.Service_Verification_Info,
      Prot_Service_2.service_verification_info_id

   Send
      All saved buffers of pidu_buffer to receiver list (R).


   RECEIVER (ON RECEIVER LIST (R)):
      (any parameters not listed below are given the value NULL)

   Set
      env_handle          = environment handle in use;
      partial_pidu_buffer = initial buffer of received p-idu;

   Call
      IDUP_Start_Unprotect() with above input parameters
   While major_status == IDUP_S_MORE_PIDU_NEEDED,
      Set
         partial_pidu_buffer = next buffer of p-idu
      Call
         IDUP_Start_Unprotect()
   Check
      major_status,
      minor_status,
      R_Services.Unprot_Service_1.General_Service_Data.minor_status,
      R_Services.Unprot_Service_2.General_Service_Data.minor_status,
   (as required) for more detailed information

   Save
      initial_idu_buffer (if non-empty)
```

```
   Set
      input_buffer = remaining p-idu buffer
   Call
      IDUP_Unprotect() with above input parameter
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status
   Save
      output_buffer

   Call
      IDUP_End_Unprotect()
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         R_Services.Unprot_Service_1.General_Service_Data.minor_status,
         R_Services.Unprot_Service_2.General_Service_Data.minor_status,
      (as required) for more detailed information.

   Utilize
      R_Services.Unprot_Service_1/2.service_id,
         (to determine which services were applied by the originator)
      R_Services.Unprot_Service_1/2.Quality,
         (to determine the corresponding qualities of the service)
      Prot_Information.originator_name/protection_time and
         Prot_Information.Idu_Information.idu_type/idu_title,
         (from IDUP_Start_Unprotect) (to determine originator info.)
      R_Services.Unprot_Service_2.General_Service_Data.Target_Info.
         targ.names, (to determine if rec. is in "receipts from" (r))
      Service_Verification_Info/service_verification_info_id
         (to determine if receiver is in "receipts to" list (L))

   If receiver is in "receipts from" list (r)
      Save
         R_Services.Unprot_Service_2.service_to,
         R_Services.Unprot_Service_2.Service_Creation_Info

   If receiver is in "receipts to" list (L)
      Save
         Service_Verification_Info,
         service_verification_info_id

   RECEIVER (ON "RECEIPTS FROM" LIST (r)):
      (procedure to generate receipt)

   Set
      env_handle                          = environment handle in use;
      Target_Info.targ_names              = service_to
```

```
      Prot_Service_1.prot_service_type     = "2";
      Prot_Service_1.service_id            = "PER_POD";
      Prot_Service_1.Service_Creation_Info = Service_Creation_Info;
      P_Services.Prot_Service_1            = Prot_Service_1

   Call
      IDUP_Start_Protect() with above input parameters
   Check
      major_status.  If not GSS_S_COMPLETE, check
         minor_status,
         P_Services.Prot_Service_1.General_Service_Data.minor_status
      (as required) for more detailed information.

   Send
      pidu_buffer to "receipts to" list (L)

   RECEIVER (ON "RECEIPTS TO" LIST (L)):
      (procedure to process received receipt)

   Set
      env_handle         = environment handle in use;
      single_pidu_buffer = received p-idu buffer (if it fits in a single
         buffer; otherwise use partial_pidu_buffer and make multiple
         calls, as above)

   Call
      IDUP_Start_Unprotect() with above input parameters
   If major_status == IDUP_S_SERV_VERIF_INFO_NEEDED
      Utilize
         R_Services.Unprot_Service_1.service_verification_info.id
         (to assist in locating necessary Service_Verification_Info)
      Set
         R_Services.Unprot_Service_1.Service_Verification_Info
            = Service_Verification_Info
      Call
         IDUP_Start_Unprotect() with above input parameters
   Check
      major_status,
      minor_status,
      R_Services.Unprot_Service_1.General_Service_Data.minor_status
   (as required) for more detailed information.

   Utilize
      R_Services.Unprot_Service_1.service_id,
         (to determine that this is a "proof of delivery" evidence)
      R_Services.Unprot_Service_1.Quality,
      Prot_Information.originator_name, (for evidence generator info.)
      major_status (to determine pass/fail status of evi. verif.).
```

Full Copyright Statement