The **Generic Security Service Application Program Interface** (**GSSAPI**, also **GSS-API**) is an [application programming interface](http://en.wikipedia.org/wiki/Application_programming_interface) for programs to access [security](http://en.wikipedia.org/wiki/Security) services.

The GSSAPI is an [IETF](http://en.wikipedia.org/wiki/IETF) standard that addresses the problem of many similar but incompatible security services in use today.

How it works

The GSSAPI, by itself, does not provide any security. Instead, security service vendors provide GSSAPI *implementations* usually in the form of[libraries](http://en.wikipedia.org/wiki/Library_(computer_science)) installed with their security software. These libraries present a GSSAPI-compatible interface to application writers who can write their application to use only the [vendor-independent](http://en.wikipedia.org/wiki/Standardization) GSSAPI. If the security implementation ever needs replacing, the application need not be rewritten.

The definitive feature of GSSAPI applications is the exchange of opaque messages (*tokens*) that hide the implementation detail from the higher level application. The client and server sides of the application are written to convey the tokens given to them by their respective GSSAPI implementations. GSSAPI tokens can usually be sent over an insecure network as the mechanisms provide inherent message security. After some number of tokens have been exchanged, the GSSAPI implementations at both ends inform their local application that a *security context* has been established.

Once a security context is established, sensitive application messages can be wrapped (encrypted) by the GSSAPI for secure communication between client and server. Typical protections guaranteed by GSSAPI wrapping include [confidentiality](http://en.wikipedia.org/wiki/Confidentiality) (secrecy) and [integrity](http://en.wikipedia.org/wiki/Data_integrity) (authenticity). The GSSAPI can also provide local guarantees about the identity of the remote user or remote host.

The GSSAPI describes about 45 procedure calls. Significant ones include:

* *GSS\_Acquire\_cred* - obtains the user's identity proof, often a secret cryptographic key
* *GSS\_Import\_name* - converts a username or hostname into a form that identifies a security entity
* *GSS\_Init\_sec\_context* - generates a client token to send to the server, usually a challenge
* *GSS\_Accept\_sec\_context* - processes a token from *GSS\_Init\_sec\_context* and can generate a response token to return
* *GSS\_Wrap* - converts application data into a secure message token (typically encrypted)
* *GSS\_Unwrap* - converts a secure message token back into application data

The GSSAPI has been standardized for the [C](http://en.wikipedia.org/wiki/C_(programming_language)) ([RFC 2744](http://tools.ietf.org/html/rfc2744)) and [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) ([JSR-072](http://jcp.org/aboutJava/communityprocess/review/jsr072/index.html)) languages.

Limitations of the GSSAPI include that it standardizes only [authentication](http://en.wikipedia.org/wiki/Authentication), and not [authorization](http://en.wikipedia.org/wiki/Authorization), and that it assumes a [client–server](http://en.wikipedia.org/wiki/Client%E2%80%93server) architecture.

Anticipating new security mechanisms, the GSSAPI includes a negotiating *pseudo mechanism*, [SPNEGO](http://en.wikipedia.org/wiki/SPNEGO), that can discover and use new mechanisms not present when the original application was built.

[[edit](http://en.wikipedia.org/w/index.php?title=Generic_Security_Services_Application_Program_Interface&action=edit&section=2)]Relationship to Kerberos

The dominant GSSAPI mechanism implementation in use is [Kerberos](http://en.wikipedia.org/wiki/Kerberos_(protocol)). Unlike the GSSAPI, the Kerberos API has not been standardized and various existing implementations use incompatible APIs. The GSSAPI allows Kerberos implementations to be API compatible.

[[edit](http://en.wikipedia.org/w/index.php?title=Generic_Security_Services_Application_Program_Interface&action=edit&section=3)]Related technologies

* [RADIUS](http://en.wikipedia.org/wiki/Remote_Authentication_Dial_In_User_Service)
* [SASL](http://en.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer)
* [TLS](http://en.wikipedia.org/wiki/Secure_Sockets_Layer)
* [SSPI](http://en.wikipedia.org/wiki/Security_Support_Provider_Interface)
* [SPNEGO](http://en.wikipedia.org/wiki/SPNEGO)

[[edit](http://en.wikipedia.org/w/index.php?title=Generic_Security_Services_Application_Program_Interface&action=edit&section=4)]Key concepts

**Name**

A binary string that labels a [security principal](http://en.wikipedia.org/wiki/Security_principal) (i.e., user or service program) - see [access control](http://en.wikipedia.org/wiki/Access_control) and [identity](http://en.wikipedia.org/wiki/Identity_(object-oriented_programming)). For example, [Kerberos](http://en.wikipedia.org/wiki/Kerberos_(protocol)) uses names like *user@REALM* for users and *service/hostname@REALM* for programs.

[**Credentials**](http://en.wikipedia.org/wiki/Credential)

Information that proves an identity; used by an entity to act as the named principal. Credentials typically involve a secret cryptographic key.

**Context**

The state of one end of the authenticating/authenticated [protocol](http://en.wikipedia.org/wiki/Protocol_(computing)). May provide message protection services, which can be used to compose a[secure channel](http://en.wikipedia.org/wiki/Secure_channel).

**Tokens**

Opaque messages exchanged either as part of the initial authentication protocol (context-level tokens), or as part of a protected communication (per-message tokens)

**Mechanism**

An underlying GSSAPI implementation that provides actual names, tokens and credentials. Known mechanisms include [Kerberos](http://en.wikipedia.org/wiki/Kerberos_(protocol)), [NTLM](http://en.wikipedia.org/wiki/NTLM),[Distributed Computing Environment](http://en.wikipedia.org/wiki/Distributed_Computing_Environment) (DCE), SESAME, [SPKM](http://en.wikipedia.org/w/index.php?title=SPKM&action=edit&redlink=1), LIPKEY.

**Initiator/acceptor**

The peer that sends the first token is the initiator; the other is the acceptor. Generally, the client program is the initiator while the server is the acceptor.

RFC 2743:

5.3: X.509 Authentication Framework This example illustrates use of the GSS-API in conjunction with public-key mechanisms, consistent with the X.509 Directory Authentication Framework. The GSS\_Acquire\_cred() call establishes a credentials structure, making the client's private key accessible for use on behalf of the client. The client calls GSS\_Init\_sec\_context(), which interrogates the Directory to acquire (and validate) a chain of public-key certificates, thereby collecting the public key of the service. The certificate validation operation determines that suitable integrity checks were applied by trusted authorities and that those certificates have not expired. GSS\_Init\_sec\_context() generates a secret key for use in per-message protection operations on the context, and enciphers that secret key under the service's public key.

The enciphered secret key, along with an authenticator quantity signed with the client's private key, is included in the output\_token from GSS\_Init\_sec\_context(). The output\_token also carries a certification path, consisting of a certificate chain leading from the service to the client; a variant approach would defer this path resolution to be performed by the service instead of being asserted by the client. The client application sends the output\_token to the service. The service passes the received token as the input\_token argument to GSS\_Accept\_sec\_context(). GSS\_Accept\_sec\_context() validates the certification path, and as a result determines a certified binding between the client's distinguished name and the client's public key. Given that public key, GSS\_Accept\_sec\_context() can process the input\_token's authenticator quantity and verify that the client's private key was used to sign the input\_token. At this point, the client is authenticated to the service. The service uses its private key to decipher the enciphered secret key provided to it for per- message protection operations on the context. The client calls GSS\_GetMIC() or GSS\_Wrap() on a data message, which causes per-message authentication, integrity, and (optional) confidentiality facilities to be applied to that message. The service uses the context's shared secret key to perform corresponding GSS\_VerifyMIC() and GSS\_Unwrap() calls.

**User Authentication with GSSAPI**

GSSAPI (Generic Security Service Application Programming Interface) is a function interface that provides security services for applications in a mechanism independent way. This allows different security mechanisms to be used via one standardized API. GSSAPI is often linked with Kerberos, which is the most common mechanism of GSSAPI.

Link: <http://www.ssh.com/manuals/server-admin/44/User_Authentication_with_GSSAPI.html>

Link: <http://www.sxw.org.uk/computing/patches/openssh.html>

Link: <http://serverfault.com/questions/206054/how-to-integrate-radius-with-kerberos>