**Kerberos** is a **network authentication protocol**. It is **designed to provide** **strong authentication for client/server applications** by using **secret-key cryptography**. A free implementation of this protocol is available from the [Massachusetts Institute of Technology](https://web.mit.edu/). Kerberos is available in many commercial products as well.

The Internet is an insecure place. Many of the protocols used on the Internet do not provide any security. Tools to "sniff" passwords off of the network are in common use by malicious hackers. Thus, **applications which send an unencrypted password over the network are extremely vulnerable**. Worse yet, other **client/server applications rely on the client program to be "honest" about the identity of the user** who is using it. Other applications rely on the client to restrict its activities to those which it is allowed to do, with no other enforcement by the server.

**Some sites attempt to use** [**firewalls**](https://web.mit.edu/kerberos/firewalls.html) **to solve their network security problems**. Unfortunately, **firewalls assume that "the bad guys" are on the outside**, which is often a very bad assumption. Most of the really damaging incidents of computer crime are carried out by insiders. Firewalls also have a significant disadvantage in that they restrict how our users can use the Internet. In many places, these restrictions are simply unrealistic and unacceptable.

Kerberos was created by MIT as a solution to these network security problems. The **Kerberos protocol uses strong cryptography so that a client can prove its identity to a server (and vice versa) across an insecure network connection**. After a client and server have used Kerberos to prove their identity, they can also **encrypt all of their communications to assure privacy and data integrity** as they go about their business.

In summary, Kerberos is a solution to our network security problems. It **provides the tools of authentication and strong cryptography over the network** to **help us secure our information** systems across our entire enterprise.

**Kerberos Requirements**

**Kerberos** is **used for** both the **authentication** and **secure communication** aspects of the **client and server applications**.

The JAAS framework, and the Kerberos mechanism required by the Java GSS-API methods, are built into the JREs from all vendors. The Kerberos LoginModule required for the JAAS authentication is provided in the JRE from Sun Microsystems.

As with all Kerberos installations, a Kerberos **Key Distribution Center (KDC)** is required. It needs to **contain the user name and password** we will use to be authenticated to Kerberos.

Note: A KDC implementation is part of a Kerberos installation and not a part of the JRE.

As with most Kerberos installations, a **Kerberos configuration file** **krb5.conf** is **referenced** to **determine** the **default realm and KDC**.

**Setting Properties to Indicate the Default Realm and KDC**

Typically, the **default realm** and the **KDC for that realm** are **indicated in the Kerberos krb5.conf configuration file**.

However, if we like, we can instead specify these values by setting the following system properties to indicate the realm and KDC, respectively:

**java.security.krb5.realm**

**java.security.krb5.kdc**

If we set one of these properties we must set them both. Also note that if we set these properties, then no cross-realm authentication is possible unless a krb5.conf file is also provided from which the additional information required for cross-realm authentication may be obtained.

## **Locating the krb5.conf Configuration File**

The essential Kerberos configuration information is the default realm and the default KDC. If we set properties to indicate these values, they are not obtained from a krb5.conf configuration file.

If these properties do not have values set, or if other Kerberos configuration information is needed, an attempt is made to find the required information in a krb5.conf file.

If the system property **java.security.krb5.conf** is set, its value is assumed to specify the path and file name.



## **Naming Conventions for Realm Names and Hostnames**

By convention, all **Kerberos realm names are uppercase** and all **DNS hostname and domain names are lowercase**. **Hostnames are case insensitive** and by convention they are all lowercase. They must resolve to the same hostname on the client and server by their respective naming services.

However, in the **Kerberos database hostnames are case sensitive**. In all host-based Kerberos service principals in the KDC, hostnames are case-sensitive. The **hostnames used in the Kerberos service principal names** **must exactly match the hostnames returned by the naming service**.

For example, if the naming service returns a fully qualified lowercased DNS hostname, such as "raven.example.com", then the administrator must use the same fully qualified lowercased DNS hostname when creating host-based principal names in the KDC: "host/raven.example.com".

## **Cross-Realm Authentication**

In cross-realm authentication, **a principal in one realm can authenticate to principals in another realm**.

In Kerberos, **cross-realm authentication is implemented by sharing an encryption key between two realms**. The KDCs in two different realms **share a special cross-realm secret**; this secret is used to prove identity when crossing the boundary between realms.

The **key that is shared is the Ticket Granting Service principal's key**. Here's a typical Ticket Granting Service principal for a single realm:

**ktbtgt/EXAMPLE.COM@EXAMPLE.COM**

In cross realm authentication, two principals are created on each participating realm. For two realms, **ENG.EAST.EXAMPLE.COM** and **SALES.WEST.EXAMPLE.COM**, these principals would be:

**krbtgt/ENG.EAST.EXAMPLE.COM@SALES.WEST.EXAMPLE.COM**

**krbtgt/SALES.WEST.EXAMPLE.COM@ENG.EAST.EXAMPLE.COM**

These principals, known as **remote Ticket Granting Server principals, must be created on both realms**.

### **Types of Realms**

When you **set up multiple realms**, you **must decide if your realm configuration** will be "**hierarchical**" (one realm is a superset of the other) or "**direct**" (the mapping between realms must be defined).

### **How to Set Up Cross-Realm Authentication**

In transitive cross-realm authentication you can **define a path of realms connected via cross-realm secrets** and use this path to traverse realms until you get credentials in the desired realm.

The [**capaths**] section in the Kerberos configuration file **defines a series** of **authentication paths used for transitive cross-realm authentication**. **Clients use [capaths] to determine the correct path for doing transitive cross-realm authentication**. **Application servers check the [capaths] section to determine if a cross-realm authentication path is valid**.

For example, to set-up cross realm authentication between **ENG.EAST.EXAMPLE.COM** and **SALES.WEST.EXAMPLE.COM**, krb5.conf should include the following entry:

**[capaths]**

**ENG.EAST.EXAMPLE.COM = {**

**SALES.WEST.EXAMPLE.COM = .**

**}**

**SALES.WEST.EXAMPLE.COM = {**

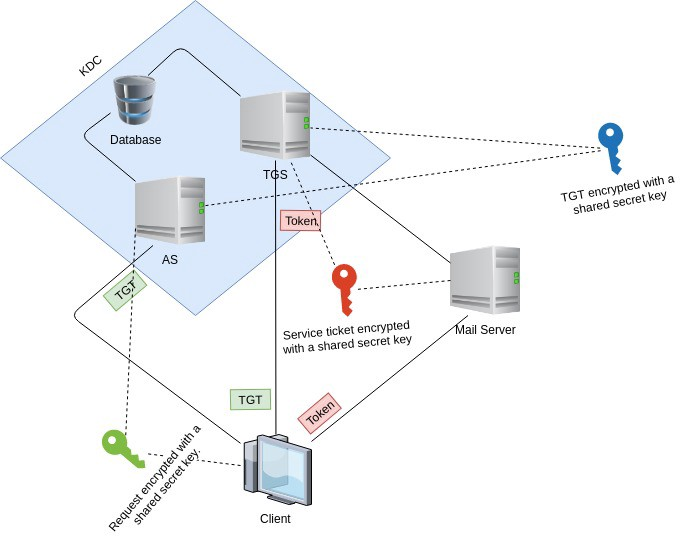
**ENG.EAST.EXAMPLE.COM = .**

**}**

**Kerberos Usage**

**Kerberos** is **used in the client-server model Architecture systems**. This protocol **prevents eavesdropping and replay attacks**. **Kerberos builds on symmetric-key cryptography**.

Kerberos is designed for security and authentication. Suppose a client wants to access a Mail server, but with Kerberos, the **client must be verified through a trusted third party** and this third party **known as** “**Key Distribution Center (KDC)**”. KDC includes an **Authentication Server (AS)** and a **Ticket Granting Server (TGS)**.



If a client wants to access the Mail Server, then these following steps will be executed in Kerberos:

**Step 1**:

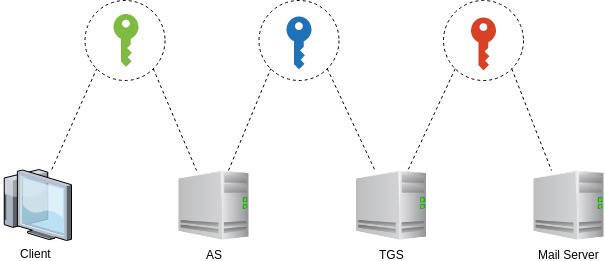
The **client sends a request to the Authentication Server** (AS) such as “my user ID is so and so, and I need a ticket to the mail server”, something like this. The **client request is partially encrypted by a secret key**. That **key is the client password**. The **client never sends his/her password over the unsecured network**. The **client uses his/her password as an encryption key**. When the **AS gets the client’s request**, it will **retrieve their password in the database based on userID**, and **use the client password as a key to decrypt the client’s request**. The **client’s password is a secret shared key between the Authentication Server and the client**. If the password from the database can de-encrypt the request, this means the user has provided a correct combination of password and userID. This is how the user is verified. **After verifying the client**, **AS sends back a ticket** called “**Ticket Granting Ticket (TGT)**”, **encrypted with another secret key**.

**Step 2**:

**After the client gets the encrypted TGT**, the **client sends it to Ticket Granting Server**, **along with the client’s request** such as “I want to access the Mail Server”. When the **TGS gets the TGT**, it **decrypts the ticket** with the **secret key shared with the Authentication Server**. Then **TGS issues the client a token**, which is **encrypted with another secret key**. This **third secret key is shared between TGS and the mail server**. Then the client sends the token to the Mail Server.

**Step 3**:

When the **Mail Server gets the token**, it **decrypts the token with the secret key shared with TGS**. The **Mail Server allows the client to use its services for a certain period according to the token**.



In the above diagram, you can see **all the communications between the different parties involved a secret key**. For example, the client and the Authentication Server shared a secret key, which is the client’s password. Authentication Server and Ticket Granting Server shared a secret key. Ticket Granting Server and the Mail Server share a secret key.