Tectia® Client/Server

User Authentication Methods

Technical Note

21 November 2012

Tectia® Client/Server: User Authentication Methods: Technical Note

21 November 2012

SSH Communications Security Corporation
Takomotie 8, FI-00380 Helsinki, Finland

Table of Contents

Chapter 1 Introduction

This document is intended for readers who need information on the various user authentication methods available with the Secure Shell protocol and Tectia products.

The Secure Shell protocol is a specification on how to create a secure channel between computers communicating over a network. The Secure Shell protocol defines authentication, encryption and integrity of transmitted data.

The Tectia products use the Secure Shell technology and once launched, they provide strong authentication and secure communications over unsecured networks. The Tectia SSH Client/Server solution is used for secure system administration, secure file transfer, and secure application connectivity.

We introduce the user authentication methods mainly from the Secure Shell point-of-view, and discuss also the Tectia –specific implementation.

The Secure Shell protocol has separate authentication procedures for authenticating the servers and the users. The authentication is mutual, the client authenticates the server and the server authenticates the client user. So both parties are assured of the identity of the other party. The Secure Shell server configuration defines which user authentication methods are allowed, and the client-side configuration defines the order in which the methods will be tried.

In this document, we concentrate on the user authentication and look at each user authentication method separately. We also list the advantages and disadvantages of each method, to help you in choosing the most suitable method for your environment. The different user authentication methods can be combined or used separately, depending on the level of functionality and security wanted in the system.

Chapter 2 User Authentication Methods

The Tectia SSH Client/Server solution supports several methods that can be used in user authentication. The supported methods depend somewhat on the Tectia product, and on the platform where it is running. The authentication methods can be used separately or combined, depending on the level of functionality and security you want.

Table 2.1. User authentication methods supported by the Tectia SSH Client/Server solution

Authentication method	Tectia Server			Tectia Client, ConnectSecure, and client tools on Server for Linux z and Server for IBM z/OS		
	Unix[2]	Windows	z/OS	Unix[2]	Windows	z/OS
Password	x	x	x	x	x	x
Public-key	x	x	x	x	x	x
Certificate	x	x	x[3]	x	x	x[3]
Host-based	x	x	x	x		x
Keyboard-interactive	x	x	x	x	x	x
PAM[1]	x			x	x	x
RSA SecurID[1]	x	x		x	x	x
RADIUS[1]	x	x		x	x	x
GSSAPI/Kerberos	x	x		x	x	

[1] Through keyboard-interactive.

[2] Including Tectia Server for Linux on IBM System z.

[3] Including certificates in files and SAF certificates.

By default, Tectia Client, ConnectSecure and Server allow the use of these user authentication methods: public keys, certificates, passwords, keyboard-interactive, and GSSAPI authentication. Public-key and certificate authentication can be combined into the public-key authentication method.

Passwords are the easiest method for authenticating users, as then no configuring is needed on the server side. The passwords are protected from eavesdroppers, since all communication is encrypted. Public keys provide

stronger security and make it possible to use non-interactive login securely, for example when scripts are used.

For instructions on configuring the Tectia products to use the selected authentication methods, see the product manuals:

- *Tectia Client User Manual*
- *Tectia ConnectSecure Administrator Manual*
- *Tectia Server Administrator Manual*
- *Tectia Server for IBM z/OS Administrator Manual*

2.1 Password Authentication

Passwords are the most widespread form of authentication. Most computer users are familiar with passwords, as most operating systems either require or provide the option of prompting the user for a username and password before allowing access. Password authentication in general works so that a server prompts the user for the password, and upon entering the correct password the user is granted access. For this method to be reasonably secure for authenticating remote users, there are a number of important security considerations:

- The passwords must not be sent across the network in plaintext form, as a malicious user capable of monitoring the network traffic would then be able to detect the password and use it to pose as the owner of the password.
- The chosen password must have a sufficient amount of entropy, meaning that it is sufficiently random to be very hard to guess. If the password does not have sufficient entropy, dictionary attacks are likely to be effective.
- The user's password must not be revealed to anyone else than the intended user. Also, the password must not be stored in an unsecured location on the client or server machine, since a malicious user who is able to compromise the client machine would also gain access to the server. If the client program maintains a file of usernames and passwords for a number of servers, these should be secured, since the confidentiality of the password would otherwise depend on the security of the machine itself.

2.1.1 Advantages and Disadvantages of Password Authentication

The Secure Shell protocol makes it possible to use numerous features to avoid some of the vulnerabilities with password authentication over a network. Passwords are sent encrypted over the network, thus making it impossible to obtain the password by capturing network traffic. Also, passwords are not stored persistently on the client. Empty passwords are not permitted by default (and they are strongly discouraged).

Tectia Server also supports limiting the number of password retries and configuring a delay between password prompts, thereby making password cracking with brute-force and dictionary attacks more difficult. On the

server side, the Secure Shell protocol relies on the operating system to provide confidentiality of the user passwords.

However, Secure Shell does not protect against weak passwords. If a malicious user is able to guess or obtain the password of a legitimate user, the malicious user can authenticate and pose as the legitimate user. Weak passwords can also be discovered with dictionary attacks from a remote machine.

Password authentication can also be used as a generic authentication method. This is the case with transparent TCP tunneling of Tectia Client when all users use the same credentials. In this case only data encryption and data integrity services are provided. The responsibility for user authentication is left to the tunneled third-party application.

The following lists sum up the advantages and disadvantages of using password authentication with Tectia.

Advantages

- Simple to use and to deploy – since the operating system provides the user accounts and password, almost no extra configuration is needed.
- Passwords are a familiar method amongst end-users.
- Memorized passwords are not tied to a single workstation (as for example public keys are).

Disadvantages

- Security is entirely based on confidentiality and the strength of the password.
- not provide strong identity check (only based on password).

2.2 Public-Key Authentication

Public-key authentication is often the next step in improving the security of a system that is protected by passwords. It addresses some of the security concerns with password authentication.

Public-key authentication is based on the use of *public-key (asymmetric) cryptography*. In public-key cryptography, messages are encrypted and decrypted with different keys. This means that each entity (person or device) that uses public-key cryptography has a key pair that consists of a public key and a private key.

Private keys are secret and known only to their owners. They are protected by a passphrase and can be stored on separate hardware cryptographic devices such as smart cards. Private keys are used for proving the identity of the entity.

Public keys are, as the name implies, public and should be distributed to all hosts with which the entity wants to communicate securely.

The two keys are mathematically dependent but the private key cannot be derived from the public key. Furthermore, the two keys possess a distinct quality: data encrypted with the public key can only be decrypted with the private key (and vice versa).

Before public-key operations can be made, the public key of the other entity has to be received securely, so that no one can substitute the genuine key with a tampered one. This process of proving the initial identity is called identity establishment.

In basic public-key authentication (without certificates), an entity (for example, a user) can simply send its public key via e-mail to the other entity (for example, an administrator of the remote host). The receiving entity must then verify, by using an out-of-band method (for example, a telephone call), that the public key is correct and the entity really is who it claims to be. Every public key has a unique fingerprint that can be used in verification.

This initial verification of the identity is very important, as it is the basis for all subsequent security policy decisions. If the identity of the entity who sent the public key and the validity of the received public key are not verified, a wrong entity might be trusted.

If the receiving entity is a server host, the administrator can, for example, configure the server to associate the public key with a specific user account. This could be done by copying the public key to a special directory in the user's home directory, or by maintaining a database of usernames and their public keys. The owners of the key pairs will be able to authenticate themselves to the server with the key pair, and then be authorized to access the system based on the server policy.

If the receiving entity is a client host, the user can, for example, add the public key of the (server) entity to a local database and associate it with the server's IP address. The client host will now trust the server in the future when it authenticates itself with the same key pair.

2.2.1 Authentication Procedure

This is what typically happens in a client-server environment, when a user on the client tries to gain access to the server using public-key authentication:

1. The client requests access from the server to a specific user account, and also sends the public key it wishes to use for authentication to the server.
2. The server checks whether the public key is associated with the user account.
3. The client signs a known value with the user's private key. The known value is unique to a session to prevent replay attacks.
4. The server verifies the signature with the user's public key.
5. If the signature is successfully verified, the user is *authenticated*, and the server can move on to *authorizing* the user, or giving access to the relevant parts of the system.

Public-key authentication is often used both ways so that two hosts (a client and a server) authenticate each other mutually via their public and private keys. Server host key pairs typically do not have their private keys protected by a passphrase, as they must be operated non-interactively.

Whereas the initial process of associating an entity with a public key is referred to as identity establishment, Steps 1–4 in the above list are referred to as *identity checking*, and they are performed every time when public-key authentication is attempted. Step 3 is often referred to as *proof of possession*. The entity proves that it is the owner of the public key by showing that it possesses the private key.

The identity checking in public-key authentication is stronger than in password authentication. When using password authentication, knowing the password is enough. Public-key authentication requires both knowing the passphrase and having the private key. This dependency on two separate elements to ensure security in public-key authentication is referred to as *two-factor authentication*. In comparison, password authentication is a *one-factor authentication* scheme, as it depends only on the password.

In both cases, however, security relies on *identity establishment*. If a password is given to the wrong person, or if a wrong public key is associated with a user account, the strength of the *identity checking* will not keep unauthorized users out.

2.2.2 Compatibility with OpenSSH Keys

By default, the Tectia SSH Client/Server solution uses private and public keys stored in the IETF standard Secure Shell v2 format. However, Tectia Client, ConnectSecure, and Server can also use keys and related files in the legacy OpenSSH format.

The following OpenSSH-format keys are supported:

- user private keys (used by clients to authenticate to a server)
- authorized user public keys (used by a server to authenticate users), including public-key options

2.2.3 Advantages and Disadvantages of Public-Key Authentication

Public-key authentication with Secure Shell is more secure than password authentication, as it provides much stronger identity checking. An entity must possess both the private key and the correct passphrase to authenticate itself to another entity.

A malicious user would have to obtain the private key of a legitimate user before being able to mount a brute force or dictionary attack to discover the user's passphrase.

When servers authenticate themselves to users, public-key authentication provides a better guarantee for the user that the server is the server the user intended to connect to. A malicious user cannot pose as a legitimate server without obtaining that server's host key, since the user would otherwise be warned that the host identification had changed.

This highlights another crucial security concern of public-key authentication. The private key file must be secret, and no one else except the owner must gain access to it. This is especially true for the private keys of servers since they typically do not have a passphrase, and anyone with the server's host key pair would be able to pose as that server.

To improve security, the private key can often be stored on a hardware cryptographic device such as a smart card or a USB token. This way, malicious users cannot access the private key file, even if they were able to gain remote access to the system, since the private key is not stored on the hard drive. For more information, see Section 3.1.

In Secure Shell, public-key authentication can be used together with an authentication agent for non-interactive logins (see Section 3.2). Alternatively, the private key can be stored with an empty passphrase, but this is not recommended as it removes a layer of security.

The added security of public-key authentication comes at the cost of some added work. The user's public key must be distributed to all of the servers where the user wishes to authenticate. In large environments this can become cumbersome, even though the public key needs to be sent only once to each server and the user can utilize the same public key with several servers.

However, the public keys of server hosts can be managed and distributed by Tectia Manager in a centralized manner.

The following list sums up the advantages and disadvantages of using public-key authentication with Tectia.

Advantages

- More secure than passwords: A malicious user should obtain both the private key and the corresponding passphrase to pose as a legitimate user.
- Provides stronger identity checking through secret private keys.
- Non-interactive login is possible.
- Tectia Manager enables automatic distribution of server host keys.

Disadvantages

- If the private keys cannot be protected, security is no better than with password authentication.
- Not very scalable: Distribution of the public keys can be cumbersome in large environments.

2.3 Certificate Authentication and PKI

Public-key infrastructure (PKI) simplifies the distribution of public keys used in public-key authentication. PKI relies on *digital certificates* as an extension of traditional public keys. Certificate authentication is an extension of public-key authentication because it still uses public keys as the basis but greatly improves

scalability. Instead of trusting several individual entities and maintaining a database of their public keys, it is enough to trust a single trusted party, a *certification authority* (CA).

Because of the improved manageability, security policies can be enforced more easily, and this in turn can result in increased overall security.

Certificates are digital documents that are used for secure authentication of the communicating parties. A certificate binds identity information about an entity to the entity's public key for a certain validity period. A certificate is digitally signed by a *trusted third party* who has verified that the key pair actually belongs to the entity. A certificate can be thought of as analogous to a passport that guarantees the identity of the bearer.

The trusted party who issues certificates to the identified end entities is called a certification authority (CA). Certification authorities can be thought of as analogous to governments issuing passports for their citizens.

2.3.1 Certificate Enrollment

To add a new end entity to a PKI, the end entity has to create a key pair, and a CA has to certify the key pair. This process of an end entity enrolling in a PKI is commonly known as *certificate enrollment*. In many cases, the CA or a separate *registration authority* (RA) gives the end entity the certificate on a smart card or a USB token, in which case the end entity certificate enrollment is not needed at all.

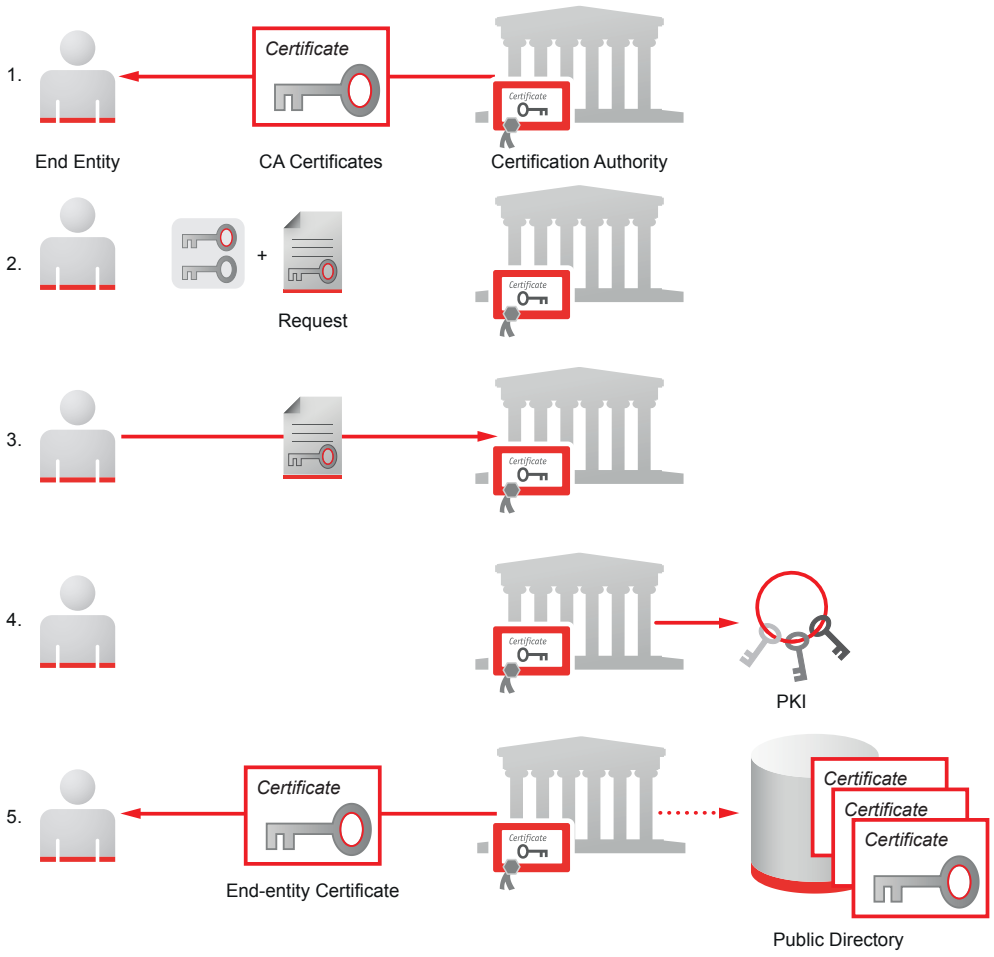

Figure 2.1. Certificate enrollment

The example of a certificate enrollment process, shown in Figure 2.1, consists of the following steps:

1. The end entity obtains the CA certificate (this is the first step of the enrollment process in most cases).

2. The end entity generates a key pair and a certification request for the key pair. The public key is included in the request along with the name and other information identifying the end entity.

3. *Identity establishment:* The end entity sends a certification request to the CA. The CA receives the information, and takes steps to ensure that the identity information is correct. The exact steps depend on the security policy of the CA. For example, the policy could require that the end entity knows a valid pre-shared key (one-time password), or that the CA administrator contacts the end entity offline before manually approving the request.

   The end entity must also prove to the CA that it is the owner of the key pair (*proof of possession*). Usually this is achieved by signing the certification request with the private key.

In the certification request, the end entity may define the values wanted for the certificate. The CA, however, considers the request as a wish list, and under its own policies, it can deny the request completely or modify the requested values.

4. If the identity is valid and can be trusted, the CA creates a public-key certificate and signs it with the CA's own private key.

5. The CA sends the certificate containing the public key, identity information, and the CA's signature back to the end entity. The CA may also optionally publish the certificate in a directory.

Depending on the size of the certification authority, its scalability can be improved by adding one or more *registration authorities* (RAs) to take care of the identity establishment. In that case, the user contacts the RA instead of the CA when requesting to enroll a certificate, and the RA performs the identity establishment

The main difference in the above procedure compared to public-key authentication is that the responsibility of identity establishment is transferred to the trusted third party (CA and its RAs). There is no need to trust individual end entities, just the CA.

2.3.2 Certificate Revocation

Certificates have pre-defined lifetimes, lasting from a couple of weeks to several years. If a private key of an end entity is compromised or the right to authenticate with a certificate is lost before the expiration date, the CA must revoke the certificate and inform all PKI users about this. Certificate revocation lists can be used for this purpose.

A *certificate revocation list* (CRL) is a list identifying the revoked certificates and it is signed by the CA that originally issued the certificates. Each CA publishes CRLs on a regular basis. The publishing interval may vary from a couple of minutes to several hours, depending on the security policy of the CA. Verification of a certificate has to include the retrieval of the latest CRL to check that the certificate has not been revoked.

As the certificate revocation lists are updated on a periodic basis, they do not provide real-time status information. If stricter security is required, online certificate status services can be used. In *Online Certificate Status Protocol* (OCSP), a dedicated OCSP responder entity responds to status requests made by end entities. This kind of function is required for example in a PKI where high-value business transactions are digitally signed.

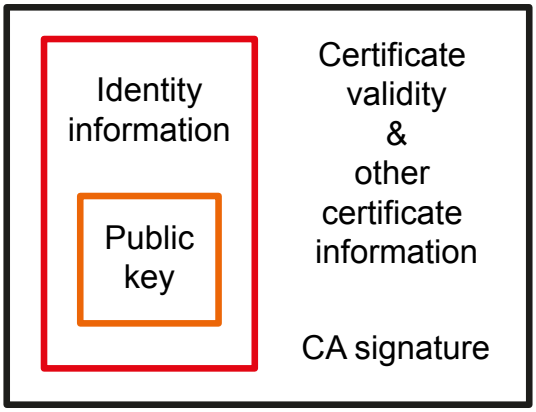

Figure 2.2. Simplified certificate structure

As shown in Figure 2.2, the identity information is stored in the certificate itself. When just public keys are used, the identity of the owning entity must instead be derived from the context that the public key is used in – for example, if it is associated with a specific user account on the server machine, or an IP address of a server in a client program.

2.3.3 Authentication Procedure

This is what typically happens in a client-server environment when a user on the client tries to gain access to a server using certificate authentication:

1. The client requests access from the server to a specific user account, and also sends to the server the user's certificate containing a public key.

2. The server checks the CA signature in the certificate and consults a local database to see if the CA is trusted. If not, the certificate is rejected and the user is not authenticated.

3. The server checks the validity of the certificate, for example, by consulting a certificate revocation list (CRL) published by the CA. If the certificate has been revoked or has expired, the certificate is rejected.

4. The server requests identity proving, and the client user signs a value with the user’s private key pass-phrase.

5. The server verifies the signature with the user's public key.

6. If the signature is successfully verified, the user is authenticated, and the server can move on to authorizing the user, or giving access to the relevant parts of the system.

Steps 1–5 above form the process of *identity checking* in certificate authentication. Step 4 is the *proof of possessio*n.

The identity checking process above is not very different from that used in public-key authentication, but the difference lies in scalability rather than security, and can be summarized as follows:

Public-key authentication

1. Check if the received public key is trusted, for example, by consulting a local database.

2. Require proof that the remote entity has the corresponding private key and knows the passphrase.

Certificate authentication

1. Check if the received certificate is issued by a trusted CA, if it is valid, and whether it has been revoked.

2. Require proof that the remote entity has the corresponding private key and knows the passphrase.

It is important to note that, as is the case with public-key authentication, the security of certificate authentication is no stronger than the security policy employed by the CA in the process of *identity establishment*. It is imperative that the CA employs thorough procedures to establish the identity of the entity wishing to enroll a certificate. If the CA does not live up to this responsibility, users can end up connecting to unsecured servers and servers will risk allowing access to malicious users. Therefore users and administrators alike should choose carefully which CAs to trust, and should take steps to ensure that the CA's security policy is at a sufficiently high level. If "anyone" can get a certificate from the CA, trusting that CA is a security hazard.

2.3.4 Advantages and Disadvantages of Certificate Authentication

Compared to public-key authentication, the principal advantage of using certificate authentication with Secure Shell is that it is much more scalable than authentication using public keys only. Administrators do not have to trust individual public keys but only a small number of CAs (typically only one). Also, the users' access to several servers can be controlled by publishing certificate revocation lists for the CA.

Because certificate authentication is more scalable, it becomes a much more manageable form of authentication for medium to large environments. Provided that the CA implements sufficiently reliable *identity establishment* procedures, this may then in turn lead to increased security because user access is controlled from a central location. If a person leaves a company, it is not necessary to delete his public key from every server to which he has access in order to revoke the rights – instead the person's certificate is simply revoked by the company CA.

From the user's point of view, trusting a certain number of CAs and relying on them for assessing the credibility of other entities can be significantly easier than having to verify the identity of each remote entity manually. Furthermore, when certificates are used to authenticate the user, for example in a company network, there is no need for the user to distribute public keys, and the authentication can therefore become very easy to use. If the certificates and private keys are stored on smart cards, security is increased even further without making authentication more difficult. Certificates also have many more uses than a simple key pair, and the same certificate could be used for gaining access to the operating system, logging in to company file servers, and securing e-mail.

The actual security of certificate authentication depends on the process of *identity establishment* employed by the CA. If the CA is not sufficiently trustworthy in verifying the identity of the entities enrolling certificates,

the basis for trust is not sound. Users must also be careful when obtaining the CA certificate and verifying that the correct CA certificate has been downloaded to their machine.

The following lists sum up the advantages and disadvantages of using certificate authentication with Tectia.

Advantages

1. No need to distribute public keys or validate fingerprints when creating or updating key pairs.
2. Authentication credentials can be centrally revoked.
3. Highly scalable: No need to trust individual entities, but only a single CA or a limited number of CAs.
4. Same or higher level of security compared to public-key authentication.
5. User access to several servers can be controlled from one location, adding to security in some environments.
6. Users can leave it to the CAs to determine the trustworthiness of remote hosts, thus improving overall security if the CA is in a better position to evaluate their credibility.
7. Provides identity verification through secret private keys.
8. Non-interactive login is possible through an authentication agent.
9. Certificates can be used for many purposes such as login, access to file servers and e-mail security.

Disadvantages

1. Requires a public-key infrastructure (PKI). This can increase the cost of initial deployment in some environments when compared to conventional public-key authentication.

2.4 Keyboard-Interactive Authentication

The keyboard-interactive authentication method is defined in *RFC 4256*. Keyboard-interactive is not an authentication method in itself, but more like a common interface for various other authentication methods that are based on keyboard input. For example, password authentication, RSA SecurID, PAM (Pluggable Authentication Module), and RADIUS authentication methods can be used over keyboard-interactive. Currently, binary messages in PAM are rarely used.

When using keyboard-interactive, the Secure Shell client application (Tectia Client or ConnectSecure) does not have to know which specific authentication method is being used, but only that it is a "keyboard-interactive" authentication method. For users authenticating themselves there is little or no difference in usage, and using keyboard-interactive itself does not add any extra security.

The primary advantage of keyboard-interactive is that it makes adding support for new authentication methods much easier, since the Tectia Client or ConnectSecure software does not have to be modified. This will signi-

ficantly ease upgrading to new and more secure authentication methods when they become available, provided that they rely on keyboard input.

The principle in keyboard-interactive can be seen in the following figure.

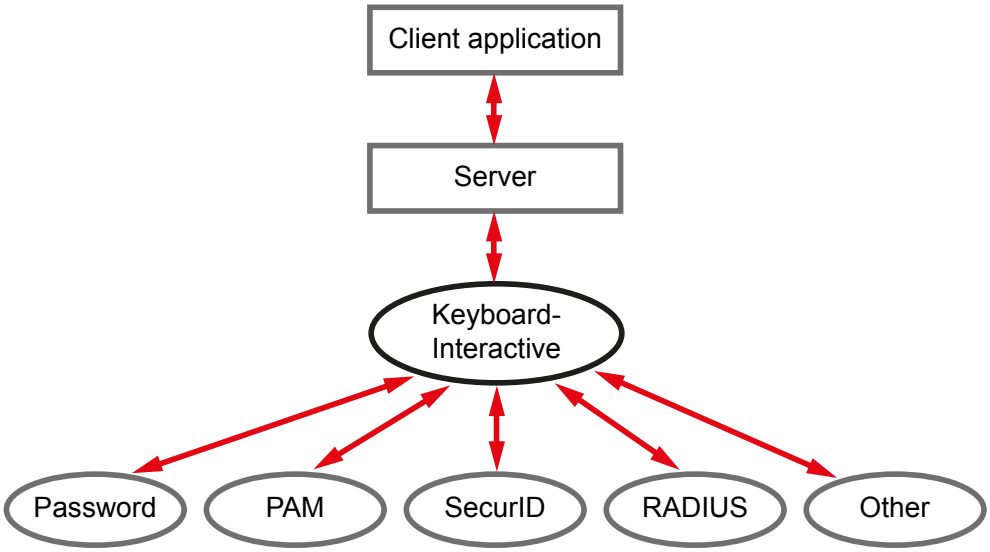

Figure 2.3. Simplified certificate structure

2.4.1 Password Submethod

Password authentication can also be used through keyboard-interactive. For more information see Section 2.1.

2.4.2 PAM Submethod

Pluggable Authentication Module is an authentication framework used in Unix systems. In Tectia, support for PAM is enabled as a submethod of keyboard-interactive authentication.

When PAM is used, Tectia Server transfers the control of authentication to the PAM library, which will then load the modules specified in the PAM configuration file. Finally, the PAM library tells Tectia Server whether the authentication was successful or not. Tectia Server is not aware of the details of the actual authentication method employed by PAM. Only the final result is of interest for the server.

2.4.3 RSA SecurID Submethod

RSA SecurID is a widely used two-factor authentication method based on the use of SecurID Authenticator tokens. In Tectia, support for RSA SecurID is enabled as a submethod of keyboard-interactive authentication.

The prerequisite for enabling SecurID support in Tectia Server is that RSA Authentication Agent software is installed on the server host. When RSA SecurID is used, Tectia Server queries the user for the token's numer-

ical code and passes the code to RSA *Authentication Agent* for verification. RSA Authentication Agent then returns the success or failure of the authentication to Tectia Server.

2.4.4 RADIUS Submethod

RADIUS (Remote Authentication Dial-In User Service) is a protocol for checking a user's authentication and authorization information from a remote server. It is originally intended for authenticating dial-in users, but is also suitable for use with Secure Shell. In Tectia, RADIUS is implemented as a submethod of keyboard-interactive authentication.

When using RADIUS authentication, Tectia Server first asks the user's password and then sends it along with the username to the RADIUS server (PAP authentication). Multiple RADIUS servers can be configured, and these will be queried in turn in case some of them are unreachable.

The supported RADIUS servers are Microsoft IAS (Internet Authentication Service) and FreeRADIUS.

2.4.5 Advantages and Disadvantages of Keyboard-Interactive Authentication

The following lists sum up the advantages and disadvantages of using keyboard-interactive authentication with Tectia.

Advantages

- New authentication methods can be added without upgrading the client software, since the Secure Shell client does not have to be aware of the specifics of the authentication method.

Disadvantages

- Authentication forwarding is not supported.
- The disadvantages depend on the exact submethod.

2.5 Host-Based Authentication

Host-based authentication mimics the old **rhosts** authentication that was used with Unix tools such as **rsh** and **rcp** to control access to systems based on the address of the remote host. The difference with the old way and the current host-based authentication is that in Secure Shell v2 strong cryptography is used to perform the host identity check.

The old way of identifying the host with its name or IP address is very insecure. It was based on the assumption that DNS names and IP addresses could not easily be tampered with. DNS services were typically hosted on servers controlled by system administrators, on ports where ordinary users could not start services. This assumption, however, is no longer valid, since personal computers have become much more common, and many

more people now have the opportunity to set up DNS services or programs that capture and alter network traffic. Since DNS lookups and the portion of an IP packet that contains the IP address are not encrypted, they are vulnerable to interception and alteration.

Host-based authentication in the SecSh v2 protocol does not need to rely solely on the DNS information. In fact, in environments where NAT (Network Address Translation) is used, it is usually necessary to disable the DNS matching in server configuration. Instead, the client host is authenticated with a public key pair, which is typically the host key pair or the host certificate pair of the Tectia Server on the client side.

Host-based authentication is actually similar to public-key authentication. On the server side, the public key is required to identify the client host, or in case certificates are used, the identification is based on the contents of the host certificate as in server authentication with certificates.

On the client side, the difference compared to user public-key authentication is that the user does not have direct access to the private key. Instead, a small **setuid** program is used to sign the challenge. This program is able to access the root-owned private key of the client host. The program also checks that the user is signing the challenge with correct information. The server uses this information, which contains the client-side user name and host name, when it decides whether the user is authorized to log in to the requested account. Each user has one user name per client host, so the client authenticates the user once, and forwards the credentials to the server when that user name is used to log in to a server-side account.

Host-based authentication with certificates also makes it possible to have scalable user authorization in the Tectia Server configuration. It is possible, for example, to allow login to particular user account(s) only from certain client host(s).

The major drawback with host-based authentication is that it effectively puts the server host to the same trust level as the client host. This is usually not a problem if both the client and server machines are in a controlled environment and administered by trusted people. While both user public-key authentication and host-based authentication rely on the integrity of the private key on the client side, host-based authentication relies also heavily on the integrity of the user account management and the initial user authentication to the client-side system. In host-based authentication, a client-side user with the same user name as a server-side user is considered equivalent to the server-side use. Therefore, for example, denying root login with host-based authentication is usually prudent. Otherwise if the client host is breached, the attacker has immediate root privileges in all participating hosts. Requiring strong authentication for root is usually wise in all cases.

2.5.1 Advantages and Disadvantages of Host-Based Authentication

The following lists sum up the advantages and disadvantages of using host-based authentication with Tectia.

Advantages

- Non-interactive login is relatively easy to set up in large networks. Generally no end-user action is needed.
- Users do not have direct access to the private key, so they do not need to create or manage any keys.
- The server security policy defines and controls how users can access remote hosts.

- In host-based authentication with certificates, it is possible to further limit user authorization based on host certificate contents.
- Can be used together with other forms of authentication.

Disadvantages

- Trust is placed on the administration of the client host. Server relies on the integrity of the user account management on the client host.
- Host-based authentication should only be used if the client host is administered by the same organization as the server host.

2.6 GSSAPI Authentication

Generic Security Service Application Programming Interface (GSSAPI) 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.

The most common mechanism of GSSAPI is Kerberos. In the current Tectia version, Kerberos is the only supported GSSAPI method. For the Kerberos authentication to work through GSSAPI, the client and server must already be configured to use Kerberos (i.e. be able to gain tickets).

GSSAPI is about transferring existing credentials (also called tokens) from the client to the server. Kerberos/GSSAPI is not used to initially log on to a network. The user must have logged on to the network, and the user must possess existing logon credentials that can be accessed through GSSAPI.

Kerberos/GSSAPI itself does not transfer anything over the network – that is the responsibility of the application. GSSAPI provides opaque credential data for the application to be sent to a peer. In Secure Shell, the credential data is passed securely over the Secure Shell transport layer, just like in any Secure Shell authentication method.

The authentication method starts with the client sending the server a list of GSSAPI mechanisms that the client supports. The server selects the first mechanism from the list that it supports, meaning Kerberos. Once the mechanism has been negotiated, token exchange begins. Token exchange may involve several packet exchanges between the client and the server, depending on the mechanism. All the exchanged token data comes from GSSAPI; Secure Shell does not understand it but only relays it. In the end, the token has been transferred to the server to be used for user authentication. The server uses the token to request the identity of the user from GSSAPI.

The GSSAPI authentication method does not ask anything from the user. If something fails during GSSAPI exchange, the reason for the failure can be seen in the client debug log.

2.6.1 GSSAPI Interoperability

The GSSAPI user authentication method for Secure Shell is defined in *RFC 4462*.

For Unix, GSSAPI offers an interoperable Kerberos authentication method. GSSAPI interoperates with standard Kerberos implementations that provide a GSSAPI mechanism. Additionally it allows Unix machines to log on to Windows 2000 Active Directory domains.

GSSAPI offers integrated authentication for Windows 2000/2003 networks with Kerberos. This method utilizes domain accounts, since local accounts are not transferable across machine boundaries.

2.6.2 Advantages and Disadvantages of GSSAPI Authentication

The following list sums up the advantages and disadvantages of using GSSAPI authentication with Tectia.

Advantages

- The user logs on to a workstation and can then automatically access all Secure Shell servers in the same domain and in trusted domains.
- Use of Windows domain accounts is possible.
- Unix machines can interoperate with Windows Active Directory.

Disadvantages

- The client and the server must be in the same Kerberos security domain.
- The host clocks need to be synchronized, as the tickets have a validity period defined.
- The Kerberos server must be available at all times, because otherwise users cannot log in. To avoid the server being a single point of failure, several Kerberos servers can be used to provide high availability.

Chapter 3 Making the Most of Public Keys and PKI

This section gives an overview of how the use of public keys and certificates can be made easier with the Tectia SSH Client/Server solution, and how the security of authentication can be further increased.

Section *Certificates and Keys on Smart Cards* outlines the advantages associated with storing certificates and keys on a smart card instead of the local hard drive, and section *Authentication Agents and Key Providers* discusses the use of an authentication agent to make authentication less interactive.

3.1 Certificates and Keys on Smart Cards

Because the security of public-key cryptography (including certificate and public-key authentication) relies heavily on the confidentiality of the private key, it is important to keep the private key secure. If the private key is stored for example on the local hard drive, it is very important that only the intended user has read access to the private key. If someone could obtain the private key, they could potentially mount a brute-force or dictionary attack to discover the passphrase of the private key, and security would be void.

If the security of the machine on which the public-key or certificate authentication is used cannot be guaranteed, or if a higher level of security is desired, the private key (and any public keys or certificates) can be stored on a smart card or another two-factor authentication token.

Storing the private key and public key or certificate on a smart card can also be convenient if a user uses many different machines to connect from. Storing a copy of the key pair on each machine is often not desirable and transporting the key pair on a disk or other easily damaged or copied media may not be convenient or secure. A smart card could be used in this type of scenario to store the private key and certificate or the public key, and none of the secret key material would need to be stored on the client computers.

In Tectia Client and ConnectSecure, the Connection Broker component takes care of opening the Secure Shell connections and it can access different key providers such as the key and certificate files and hardware cryptographic devices. The Connection Broker can also be used as an authentication agent to store passphrases for key pairs.

3.2 Certificates and Keys on Smart Cards

An authentication agent is used for making authentication with Secure Shell less interactive. The keys can be loaded into the authentication agent, which will prompt the user for the passphrase. After that, all Secure Shell programs (such as Tectia Client and ConnectSecure) can request private-key operations from the authentication agent, and since the authentication agent stores the passphrases in memory, this happens without user interaction. The passphrase for the key pair only needs to be entered when the key is used for the first time.

Authentication agents are useful for frequent logins or for running scripts non-interactively.

The Connection Broker component of Tectia Client and ConnectSecure can be used as an authentication agent for storing passphrases for key pairs. It is not only capable of loading keys from the file system but also from a range of cryptographic devices, such as smart cards or USB tokens.

Chapter 4 Conclusion

The Tectia products support a wide selection of user authentication methods. The authentication methods can be used separately or in combinations to provide reliable user authentication services.

In this document we have studied the basic behavior and the strong and weak points of each of the authentication methods supported by the Secure Shell protocol and hence by the Tectia products.

When choosing the authentication methods for an environment, the administrators and security experts need to consider also the architecture and prerequisites of their system. The most suitable authentication method depends on the actual environment and on the requirements set by the company security policy.

