


Realtime
publishers

"Leading the Conversation"

The Shortcut Guide[™] To



Securing Automated File Transfers

sponsored by



Ed Tittel

Chapter 3: Securing File Transfer.....	42
TCP/IP and Network Reference Architectures	42
What Is Tunneling?.....	45
What Is IPsec?.....	46
Why Is IPsec Used?	46
How Is IPsec Used?	47
What Is Secure Socket Layer?	50
Why Is SSL Used?	51
How Is SSL Used?	51
What Is TLS?	52
Why Is TLS Used?.....	52
How Is TLS Used?.....	53
What Is SSH?.....	53
How Is SSH Used?.....	53
Using VPNs	54
IPsec VPNs	54
SSL VPNs.....	55
SSH VPNs.....	56
Other VPNs.....	57
Securing FTP	57
SSH	57
SFTP	58
FTPS (aka FTP/SSL)	59
Other Approaches	59
The Push/Pull Concept.....	60
Securing Automated Transfers	63
Preferred Authentication Methods.....	64
Summary	66

Copyright Statement

© 2007 Realtimepublishers.com, Inc. All rights reserved. This site contains materials that have been created, developed, or commissioned by, and published with the permission of, Realtimepublishers.com, Inc. (the “Materials”) and this site and any such Materials are protected by international copyright and trademark laws.

THE MATERIALS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. The Materials are subject to change without notice and do not represent a commitment on the part of Realtimepublishers.com, Inc or its web site sponsors. In no event shall Realtimepublishers.com, Inc. or its web site sponsors be held liable for technical or editorial errors or omissions contained in the Materials, including without limitation, for any direct, indirect, incidental, special, exemplary or consequential damages whatsoever resulting from the use of any information contained in the Materials.

The Materials (including but not limited to the text, images, audio, and/or video) may not be copied, reproduced, republished, uploaded, posted, transmitted, or distributed in any way, in whole or in part, except that one copy may be downloaded for your personal, non-commercial use on a single computer. In connection with such use, you may not modify or obscure any copyright or other proprietary notice.

The Materials may contain trademarks, services marks and logos that are the property of third parties. You are not permitted to use these trademarks, services marks or logos without prior written consent of such third parties.

Realtimepublishers.com and the Realtimepublishers logo are registered in the US Patent & Trademark Office. All other product or service names are the property of their respective owners.

If you have any questions about these terms, or if you would like information about licensing materials from Realtimepublishers.com, please contact us via e-mail at info@realtimepublishers.com.

[**Editor's Note:** This eBook was downloaded from Realtime Nexus—The Digital Library. All leading technology guides from Realtimepublishers can be found at <http://nexus.realtimepublishers.com>.]

Chapter 3: Securing File Transfer

Through earlier chapters, the impetus behind securing any and all forms of at-risk file transfers was clearly elucidated and loosely enumerated. Now that you know the answers to *why* you should secure at-risk file transfers, you will discover *how* such transfers can be secured.

This chapter begins with a topical discussion of network reference models, protocols, and services to describe the security concepts that are introduced immediately thereafter. The chapter takes a ground-up approach to describing the aspects of securing digital communications in the context of public networks. A discussion then follows with several worthy and practical solutions to establishing and maintaining secure communications.

Request for Comment (RFC) documentation is far and away the best jumping-off point for most of the material referenced in this chapter. This series of documents encompasses research, insight, and applicable methodologies related to Internet technology, and serves as the basis for drafting and formalizing that terminology, key concepts, and recommended implementations. Because they are such excellent resources for further exploration of the topics described, RFC references are cited for many of the concepts discussed in this chapter.

TCP/IP and Network Reference Architectures

Before digging into specific TCP/IP security architectures, such as IP Security (IPsec), let's first make a cursory foray into the Open Systems Interconnect (OSI) basic reference model, in which all TCP/IP-based protocols and transactions can be identified and classified according to one of its seven layers of operation. For completeness, Figure 3.1 also shows the original TCP/IP networking model, which consists of only four layers.

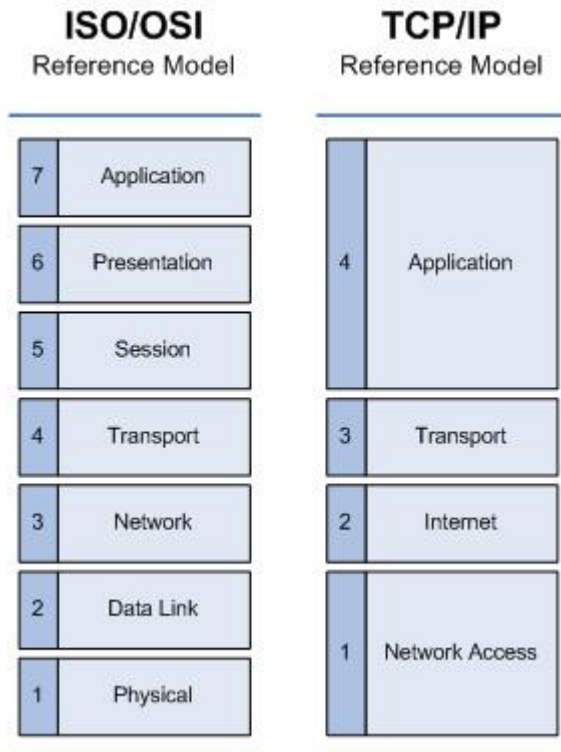


Figure 3.1: The ISO/OSI basic reference model describes network protocol interaction in layers.


The biggest benefit of a layered approach to network communications is that it takes a large, complex, end-to-end problem (network communications) and breaks it up into a series of inter-related but less complex issues that may be solved independently and with less overall effort. Because this has the additional benefit of decoupling hardware signaling and communications development from software development, it's pretty much driven network research and development from its inception, even before this widely used model was spelled out in the early 1980s. Each higher-layer protocol depends on services from the lower layers that encapsulate them as information is passed up and down this network stack.

The ISO/OSI model layers may be summarized as follows:

- Layer 7: Application Layer—A user interface to access information across the network through application protocols. Such uses include SSH, FTP, and HTTP.
- Layer 6: Presentation Layer—Transforms data from lower layers into formats readable by the application layer, and vice versa. Such uses include data compression, encryption, and encoding.
- Layer 5: Session Layer—Controls sessions between network endpoints by establishing, maintaining, and terminating communication among hosts.
- Layer 4: Transport Layer—Provides a transparent transfer of data between endpoints through flow control, segmentation, and error control. TCP and UDP reside at this layer.
- Layer 3: Network Layer—Supplies functional and procedural methods of transmitting variable-length data sequences, performs routing functions, enforces quality of service (QoS) policies, and handles segmentation.
- Layer 2: Data Link Layer—Supplies the functional and procedural means to transfer data and correct errors occurring on the lower level. Ethernet and Media Access Control exist on this level.
- Layer 1: Physical Layer—Consists of electrical specifications and physical provisions in the form of network devices or interfaces. Physical Ethernet standards are in this layer.

The TCP/IP network reference model actually preceded the development of the ISO/OSI network reference model and describes the more common ways in which networking protocol software is implemented inside or alongside most modern operating systems. In the TCP/IP reference model, the capabilities of the OSI/ISO Session and Presentation layers get bundled into the Application layer, and capabilities at the Data Link and Physical layers likewise are subsumed in the Network Access layer.

A packet is the basic atomic unit upon which all TCP/IP-based transmissions are built. Packet construction, usage patterns, and protocol capabilities characterize the type of transmission that occurs for a given connection. There are essentially two direct ways to build, send, or receive and process file transfer-capable packets: streams and datagrams. Although there is the distinct possibility of transferring data within the misused or unused header portions of TCP/IP and ICMP packets, its usage is unorthodox and ill-advised for these purposes, appearing mostly in covert channel communications between attacker and victim. As such, the subject will not be discussed further.

 There are many excellent technical write-ups available on the subject of covert channels embedded by data hiding in the TCP/IP protocol suite. Two such examples include http://www.firstmonday.org/issues/issue2_5/rowland/ and a paper titled “Covert Channel Analysis and Data Hiding in TCP/IP” at <http://gray-world.net/papers/ahsan02.pdf>.

Streams-based protocols are used in a variety of ways, but they all ensure and refer to a succession of elements made reliably available over some period of time through TCP. A streaming communications channel provides a seemingly endless flow of information between sender and recipient. The datagram approach is oriented toward lightweight delivery of individual parcels and not concerned with session management, unlike the TCP streams-based approach. In effect, you get faster transmissions with UDP because it lacks the performance-deteriorating overhead of streaming channels.

What Is Tunneling?

Tunneling is the process of encapsulating a network protocol or session within another presumably lower-layer protocol. Previous chapters introduced the terms Virtual Private Networking (VPN) and Multi-Protocol Switching Labels (MPLS). Some VPN solutions utilize what is called the Layer-2 Tunneling Protocol (L2TP) for secure connection establishment, while MPLS itself is a tunneling protocol that operates in an abstraction between layers 2 and 3 of the OSI reference model. A VPN effectively creates a tunnel for site-to-site communications over public channels.

Tunneling permits unusual and creative new uses and reuses of existing protocols, services, and sessions. Although not all tunneling protocols are security related, many of them do permit some form of connectivity that is otherwise administratively or technologically prohibited or unsecured in its native form. Take, for example, protocols 6to4 and SSH—where 6to4 tunnels IPv6 traffic into IPv4 channels, SSH can tunnel the largely insecure FTP using the SSH protocol as a secure encapsulation mechanism.

The following items are all examples of tunneling protocols:

- Point-to-Point Tunneling Protocol (PPTP)—A dual-session protocol designed to encapsulate protocols in VPN configurations; actually uses GRE.
- Point-to-Point Over Ethernet (PPPoE)—Used to encapsulate Point-to-Point Protocol (PPP) frames within Ethernet frames as with Asynchronous Digital Subscriber Line (ADSL) uplinks.
- IP in IP (RFC 1853)—A method of tunneling IP within IP packets, whereby the first IP header contains protocol data specific to the tunnel with another IP-based header as a payload.
- 6to4 (IPv6 over IPv4)—A means to encapsulate IPv6 packets to traverse IPv4 networks such as the Internet, without the need to configure explicit tunnels.
- Secure Socket Layer (SSL)—An application-layer protocol used for the management and secure transmission of private information via public networks.
- Transport Layer Security (TLS)—The successor to SSL; serves an identical purpose and shares much of the same features and capabilities as SSL.
- Secure Shell (SSH)—A set of standards and an associated network protocol for establishing secure connections between endpoints.



Note that the last three protocols SSL, TLS, and SSH are actually stream-based protocols while the rest are datagram-driven.

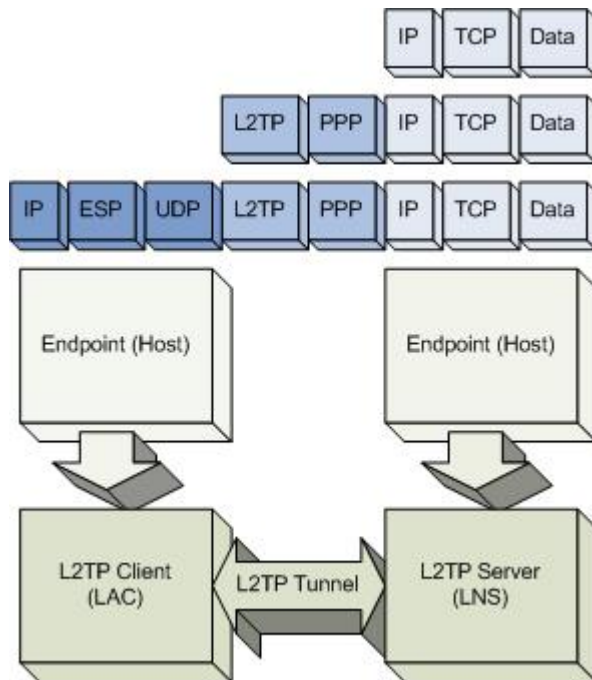


Figure 3.2: How tunneling works using L2TP.

What Is IPsec?

IP security (IPsec) is a suite of protocols for securing Internet Protocol (IP) transmissions between unprotected and protected interfaces for a host or an entire network of hosts. IPsec lays the foundation for authentication, encryption and protocols for cryptographic key establishment for use with connection-oriented data streams (TCP) or unreliable datagram transactions like the User Datagram Protocol (UDP).

As the name suggests, IPsec is a layer 3 protocol in the OSI reference model (in metaphorical terms, it sits between layers 2 and 3 as a kind of gatekeeper and guardian). While this aspect gives strength to IPsec as a mechanism for encapsulating unsecured transmissions, its use also introduces increased implementation complexity and computational and communications overhead because IPsec cannot leverage TCP (layer 4) for management and reliability functions.

IPsec is no turnkey solution in and of itself, but rather a single modular portion in a much larger, more complex security equation. Three other site and user-specific components of that equation include appropriate security protocols, cryptographic algorithms, and encryption keys.

Why Is IPsec Used?

IPsec provides transparent encryption and security services for IP network traffic as an add-on option in IPv4 and as a required element in IPv6 environments. Therefore, IPsec effectively bridges the gap left by an original omission of security provisions in the initial implementation of the standard Internet Protocol (IPv4). Traffic traversing the boundary of an IPsec-controlled perimeter is subject to administrative access controls that specify whether packets should be passed unaltered, discarded, or be wrapped in additional security services. VPNs can be and commonly are established, managed, and maintained using the IPsec protocol suite.

The following security services are made possible through the IPsec framework:

- Traffic encryption to deter eavesdropping
- Integrity validation to thwart tampering
- Peer authentication to secure conversations
- Anti-replay protection against playback attacks

The security design goal for IPsec is to provide cryptographically secure interoperability between devices and to ensure confidentiality, party verification, integrity checking, and rejection of attempts to resubmit previously recorded traffic to gain unauthorized access (playback attacks).



A playback attack occurs when the time-sensitive bits contained in previously recorded network traffic are altered and resubmitted to a target network in an attempt to fool a server and service into thinking this falsified data is part of some existing and/or legitimate traffic. Although such instances are unlikely, they are possible under certain circumstances and merit enough attention to warrant proactive countermeasures for at-risk servers.

As a protocol suite, IPsec comprises the following components:

- Security protocols—Authentication Header (AH) and Encapsulating Security Protocol (ESP)
- Security associations—A simple connection context that provides security services to the traffic it carries.
- Key management framework/standard—For distribution, management, and utilization of public keys for the authorized operation of security services and protocols.
- Algorithms for authentication and encryption services.

The next section describes IPsec usage in basic detail with pointers to more specific documentation.

How Is IPsec Used?

Two distinct protocols lay the foundation for network-based security services: the optional AH and mandatory ESP, described in RFCs 2402: IP Authentication Header and 2406: IP Encapsulating Security Payload, respectively.



See <http://www.ietf.org/rfc/rfc2402.txt> for a full description of the AH and <http://www.ietf.org/rfc/rfc2402.txt> for details on ESP.

The AH protocol offers both integrity and authentication capabilities with optional anti-replay features at the receiver's discretion.

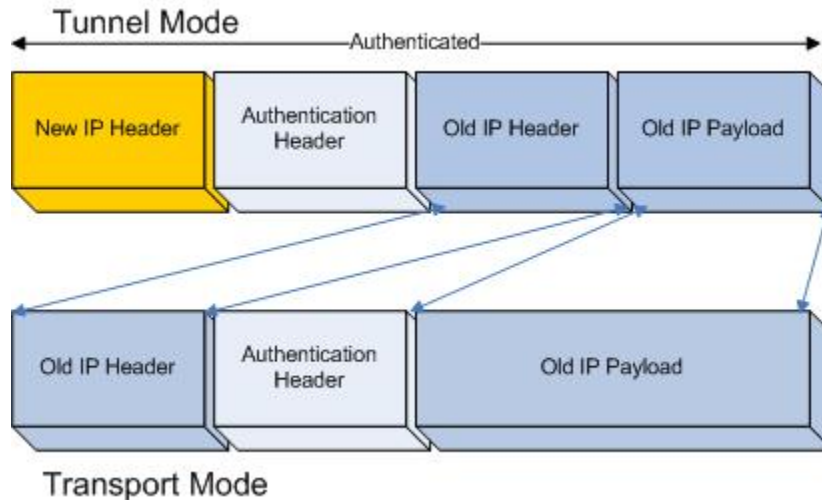


Figure 3.3: The IPsec AH protocol format.

ESP offers all three features plus confidentiality. Though ESP can be used to provide only integrity without confidentiality, ESP use with confidentiality and no integrity is not recommended.

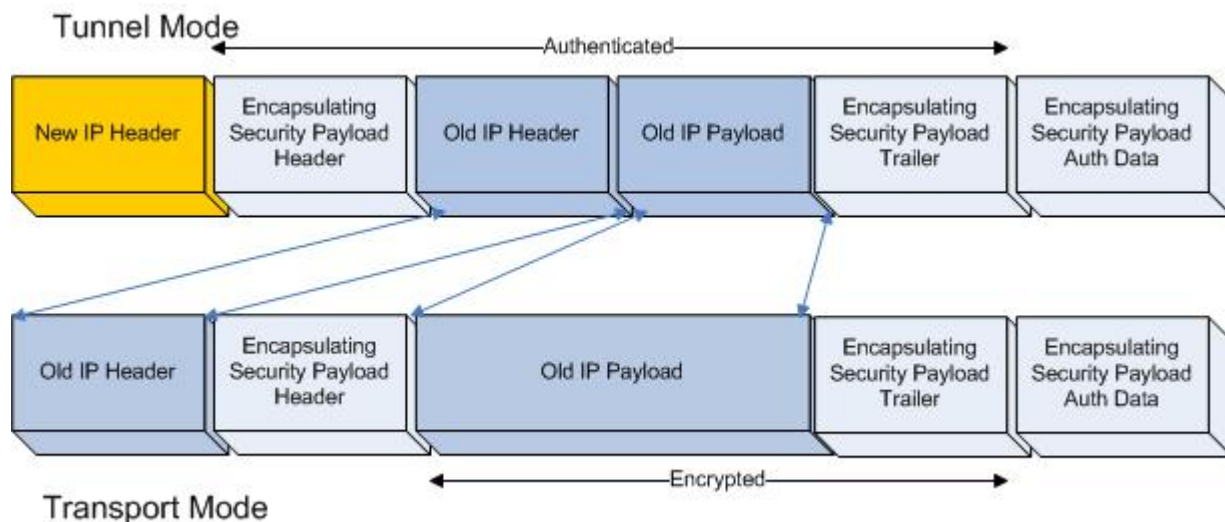


Figure 3.4: The IPsec ESP format.

Both AH and ESP provide access controls enforced through distributed keys and traffic management governed by a Security Policy Database.


Service coverage includes interoperation between endpoints in one of three ways:

- Host-to-host
- Host-to-gateway
- Gateway-to-gateway

Fine-grain control of security services makes it possible to establish a single all-purpose gateway-to-gateway or individualized TCP tunnels for host-to-host connections. IPsec supports two separate encryption modes: transport and tunnel. Transport mode provides end-to-end security for traffic, which requires that endpoint computers handle security processing, but encrypts only the payload portion of each packet leaving the header untouched. Tunnel mode provides point-to-point communications security through which many client computers share a single gateway node and this process encrypts both header and payload content.


Sending and receiving devices are required to share a public key to interoperate under the auspices of IPsec. Most of the security services provided through IPsec require cryptographic keys and therefore rely on separate mechanisms for key exchanges. The RFC 4301: Security Architecture for the Internet Protocol describes a specific public key approach for automated exchanges and states that other key distribution techniques may be used.

Internet Key Exchange (IKE) is the protocol of choice for many IPsec implementations and is the de facto standard for security association establishment. IKE uses a Diffie-Hellman key exchange for public or pre-shared secret keys to mutually authenticate communicating parties. The Internet Security Association and Key Management Protocol (ISAKMP), explained in RFC 2408, forms the basis of IKE. Menezes-Qu-Vanstone (MQV) is another authentication protocol for key agreement based on the Diffie-Hellman scheme. Other asymmetric key algorithms include Rivest, Shamir, and Adleman (RSA), Elliptic Curve Cryptography (ECC), El Gamal, the Digital Signature Algorithm (DSA), and Knapsack.

 IKEv1 is defined in RFC 2407, RFC 2408, and RFC 2409; IKEv2 is defined in RFC 4306.

IPsec may be integrated into endpoint hosts or devices, intermediary devices themselves, or specialized standalone network security appliances placed inline. The concept of IPsec is fully elaborated in the corresponding RFCs, while the subject remains a deep and complex practice for implementers and designers.

Earlier implementations of Network Address Translation (NAT), a means of mapping many non-routable addresses to a single or few mutually shared routable public IP addresses, would fail to pass IPsec traffic. When IPsec traverses one of these preliminary NAT interfaces, a mission-critical hash value is altered, rendering the IPsec provisions useless. A method of safely encapsulating UDP-based IPsec messages is defined in an IETF draft. Modern routing appliances and network devices inherently supporting NAT will harmlessly pass IPsec traffic, but there may be instances where you encounter the resistance of an older implementation.

 UDP encapsulation of IPsec packets is explained at <http://www3.ietf.org/proceedings/04mar/I-D/draft-ietf-IPsec-udp-encaps-08.txt>.

What Is Secure Socket Layer?

SSL is an IETF-approved standard that provides secure communications schemes for sharing private data across the public Internet infrastructure using public key encryption. It operates at the transport layer and up (layers four to seven) in the OSI reference model, so SSL lacks the flexibility, increased processing, and added complexity of IPsec.

SSL is characterized by and comprises the following security services:

- Message confidentiality
- Message integrity
- User authentication
- Key exchange services

Four distinct mechanisms make these services possible. Message confidentiality is made possible through symmetric ciphers or shared-secret key cryptography. Message integrity is provided through message digests or calculated hashes using the encrypted data as input and a fixed-length checksum output. Authentication and key exchange, both being separate components, are established through the use of an RSA handshake, where digitally signed certificates and session keys are exchanged.

Digital certificates are issued by well-established Certificate Authorities (CA), either corporate internal CA or external CA(s) such as VeriSign. These certificates contain the following:

- Digitally signed identifying information for both subject and issuer
- A range of temporal validity
- The subject's public key

It is this final item—the public key—that is the core component of the RSA key exchange. When paired with its counterpart, the private key, two parties may begin sharing information in confidence. For all the relative strengths, security merits, and privacy assurances that an asymmetric key exchange like this offers, it is also highly computationally expensive to conduct in large network environments transacting upon many simultaneous multiple interactions, especially for high-traffic areas servicing thousands within a small window of time.

Why Is SSL Used?

The SSL protocol permits client and server applications to communicate securely by hindering or deterring the following tactics:

- Eavesdropping—The receipt of a message by unauthorized and unintended parties
- Tampering—Unauthorized access and undesirable modification of a message
- Forgery—A certified message sent with deliberately falsified information

Typically these actions will occur one after another against one or more elements of data, perhaps in a man-in-the-middle (MitM) attack pattern. If an unauthorized party or unintended subject has tampered with a message, it is implied that this individual or group also eavesdropped and thereby intercepted the message. Likewise, to commit forgery implies both eavesdropping and tampering, for without both intercepting and deliberately modifying the message, there is little—if anything—malicious that can happen to the message. Prevent eavesdropping and you stand a good chance that data will pass without tampering. Prevent tampering and you ensure a great deal of resistance to forgery.

SSL is used in many places and is often transparent to the end user. Web browsing, email, and Internet-based faxes are three common instances in which SSL is frequently used.

How Is SSL Used?

During the transfer phase, SSL sends an optionally compressed record that is encrypted and packed with a message authentication code (MAC) and specifies what upper-layer protocol is encapsulated. Public key cryptography operations closely associated with the SSL handshake process occur at the start of every SSL-based connection. It is this crucial moment that impacts performance for a multi-function, general purpose Web or FTP server.

Account and password pairs often appear where secure terminal access or centralized account sign-on services are used. There may be other means, perhaps through keys or smartcards, but the point is that a user is authenticated in some fashion to gain access to a certain network application.

Through SSL, the server will negotiate a secure channel by sending a message to the client indicating a desire to establish cryptographically secure communications. The client application will then declare its security parameters, which the server compares against a set of its own in search of a match. This process is called the SSL handshake.

The server authenticates to the client by sending a digital certificate, and at the client's discretion to trust the server's identity and certificate, will continue or discontinue communication. The server can in turn require that the client issue a digital certificate for mutual authentication purposes during this phase, but this is not always the case. The client generates a session key, encrypts it with the server's public key, and issues it back to the server. Both parties then use this symmetric key to encrypt data passed between, establishing a secure channel for the duration of that session.

What Is TLS?

TLS is a proposed Internet standard intended to succeed SSL; it is also a cryptographic protocol that provides secure network transmissions across potentially hostile media, such as the Internet. Though slight and subtle differences exist between SSL version 3.0 and TLS version 1.0, both protocols and their usage constraints remain much the same. In practice, the terms are used interchangeably; however, a common misconception is that TLS is the *same* as SSL, which it is not. Though derived from SSL v3.0, TLS is a different protocol altogether and their differences are substantial enough that the two are not interoperable.

 For more information regarding TLS, see <http://www.ietf.org/rfc/rfc2246.txt>.

Why Is TLS Used?

The primary goal for TLS deployment is to provide privacy and data integrity between applications, just like its predecessor SSL. TLS is a separate protocol from SSL, but they share the same design goals. It, too, is a two-layered composition consisting of a TLS Record and TLS Handshake protocol layered atop a reliable transport protocol.

The TLS Record protocol exists at the lowest layer, stacked above a reliable transport protocol, such as TCP, and is used to securely encapsulate various upper-layer protocols. It provides two essential connection security needs:

- The connection is private
- The connection is reliable

The TLS Handshake protocol establishes a secure communications channel between endpoints, negotiates encryption algorithms, and exchanges symmetric keys. This is necessary before the application-layer protocol can transmit or receive any amount of data. It provides the following security elements:

- The peer's identity can be optionally authenticated
- The negotiation of shared secrets is secure
- The negotiation is reliable


How Is TLS Used?

TLS operates beneath application protocols such as HTTP and FTP and above TCP or UDP. Like SSL, TLS appears primarily in Web-based merchant or electronic commerce sites, online retailers, and banking or financial institutions—wherever security is particularly necessary. TLS can be utilized as a tunnel through application wrapper means such as Stunnel or an entire network stack in the form of a VPN, much like OpenVPN does. The free Stunnel SSL/TLS service provides tunneling for those clients and servers that may not otherwise support such capabilities on their own, while OpenVPN is a free, full-featured VPN built on the OpenSSL library.

As an interesting side note, there are many hardware-assisted solutions, often referred to as *cryptographic accelerators*, for offloading resource-intensive computing tasks associated with cryptography. Instead of burdening the general purpose CPU with the heavy-lifting tasks of cryptographic processing, the task is delegated to a more specialized processing component.

What Is SSH?

SSH is a protocol originally developed in 1995 by Tatu Ylönen from the Helsinki University of Finland in response to a password-sniffing attack of that institution's network. The original impetus behind SSH was to replace the rlogin, TELNET, and rsh protocols, which neither used strong authentication nor provided confidentiality for data transported. This first version of the technology became known as SSH-1 and is served by free and open implementations like those available at <http://www.openssh.com> as well as commercial (and more powerful) implementations like those offered by SSH Communications Security. SSH-1 ultimately evolved into SSH-2, an Internet standard covered in the secsh group of RFCs (documented in RFC 4251, which describes the SSH-2 architecture).

 You can access RFC 4251 through its sponsoring body, the IETF, at <http://tools.ietf.org/html/rfc4251>. Or check out the RFC references from O'Reilly's *SSH: The Secure Shell (The Definitive Guide)* on its "SSH Protocol" page at <http://www.snailbook.com/protocols.html>.

SSH consists of a set of standards and an associated network protocol that permits a secure channel to be established between a local and a remote host that uses public-key cryptography to authenticate the remote host (it also includes optional provisions to enable the remote host to authenticate the user as well). SSH protects message content and ensures message integrity through a combination of encryption (to ensure content confidentiality) and message authentication codes (MACs, to ensure content integrity).

How Is SSH Used?

The most typical use for SSH is to enable users to log into a remote host to execute commands, in much the same spirit as the original command-line utilities (rlogin, TELNET, and rsh) it was in part designed to replace. But SSH also supports tunneling, can forward X11 connections or TCP ports on a user's behalf, and can transfer files using either Secure FTP (SFTP) or the Secure Copy Protocol (SCP). SSH is associated with standard TCP port 22 (ironically, adjacent to ports 20 and 21, which are associated with FTP). SSH offers nearly the same capabilities and functions as both IPsec and SSL/TLS but is considerably less vexing to deploy and less challenging to maintain.

Using VPNs

Ostensibly, a VPN is used to secure private communications over non-public network media. VPN traffic is shuttled via the public networking infrastructure atop standard protocols or through privatized leased-lines between customer and service provider. A VPN can establish internal, site-to-site, inter-organizational, or inter-territorial connections for confidential communication for one or several companies.

Benefits of a well-designed multi-site VPN include:


- Expanded geographic or territorial connectivity
- Improved site-to-site security
- Reduced operational costs versus Wide Area Networks (WANs)
- Reduced transit time/transportation costs for mobile employees
- Simplified network topology in some instances
- Global networking with telecommuter support
- Excellent scalability with Public Key Infrastructure (PKI)

VPNs may utilize public or private telecommunication infrastructures; what remains constant is their use of tunneling protocols and security procedures or processes to achieve confidentiality between endpoints. Encryption plays a full-time role in this capacity, and compression occasionally makes appearances in supportive roles.

IPsec VPNs

IPsec is commonly used in the development and implementation of many VPN solutions. It provides a virtual conduit through which two networks can share resources as one, opening the door to external client callers so that they may access private internal resources. An IPsec implementation is ideal for transparent high-performance passageways between organizations that may be territorially or regionally separated, wherever general purpose coverage applies. In some cases, IPsec is integrated directly into the kernel or inline appliance network stack, and can be facilitated by specialized hardware designs.

Do not be fooled by the misconception that IKE-based VPNs are invisible on the network. Transparency is highly desirable for any type of networking connection; that is why a VPN should operate with little visibility to the end user short of status and link-state indicators. Discovery of IKE systems is made possible through tools such as NTA Monitor's ike-scan tool. This tool exploits characteristics in IKE via specially crafted probe packets that stimulate an IKE response and can, in some instances, identify a VPN solution down to vendor make and model.

 Obtain a full-length description or download ike-scan from <http://www.nta-monitor.com/tools/ike-scan/>.

Implementation Approaches

IPsec VPNs are ideal where high-speed, redundant, site-to-site connectivity is necessary. An IPsec solution is designed to meet the criteria for security provisions within always-on network accessibility contexts for ensured confidentiality and enhanced productivity. A VPN based on IPsec gives geographically dispersed regions a means of mutually secure interconnection and the ability to securely service mobile or off-site clients in an on-demand fashion.

However, with these accommodations come many technical considerations. IPsec is costly to employ both in terms of software and any requisite hardware, most of which is vendor-specific. Also, there is no guarantee that one IPsec solution is interoperable with another, and some networking devices providing NAT services may modify IP addresses in a manner unsuitable for IPsec use. This is particularly true when public IP addresses are required at both ends of an end-to-end connection, and NAT is used at one end or the other, specifically to mask the source or destination IP address (or to permit private IP addresses to access the public Internet).

IPsec is the logical and perhaps most cost-effective choice for establishing remote or branch office connectivity. Due to the specialized components, IPsec ties the end user into using a single computer with individualized site-specific configurations and firewall considerations. However, IPsec VPNs provide neither the easiest nor the most cost-effective method for securing file transfers internally, or even file transfers between partners. Setting up inter-company VPNs can be cumbersome and is seldom trivial, so unless the companies exchange large volumes of data, this may not be advisable for file transfers. Even for internal use, it can be difficult and cost-prohibitive to use IPsec VPN software, so other solutions—most notably, those built around SSH—are often selected to provide true end-to-end security.

SSL VPNs

SSL/TLS VPNs are typically easier to install, support, and maintain than their IPsec counterparts. Because they operate at a higher layer than IPsec, the SSL/TLS variety can provide better granular access control required by remote access and extranet VPNs. An SSL-based VPN delivers user-level authentication to ensure that only authorized parties have access to authorized resources, and the competitive advantage of on-demand access anywhere due to the ubiquitous presence of SSL.

SSL traffic also tends to pass relatively unimpeded across the network, where IPsec sometimes encounters uncooperative NAT devices that can impede IPsec traffic because they hamper the formation of end-to-end connections (but which may be impossible anyway on networks that use private IP addresses). And because SSL is integrated into many common applications, such as popular Web browsers, there is not necessarily the need for vendor-specific client applications on every participating computer.

SSL VPNs also tend to exhibit a lower cost of ownership compared with IPsec solutions with easier scalability and simple access for Web-enabled applications. User-level access controls are necessary as clients may call in from unknown and/or untrusted computers, where sensitive information may be left exposed on public terminals. Still, SSL VPNs are an emergent trend in the market space next to IPsec. Although an SSL VPN solution goes a long way toward reducing administrative and operational overhead, it may not provide everything that an IPsec-based solution does.

Implementation Approaches

The SSL solution can tunnel specific applications rather than giving more general access to the network segment, as is the case for IPsec solutions. This yields application-specific access to a network dictated by administrative controls. In addition, per-user permissions can be established with granular control. It also does not require site-specific configurations and the issuance of company laptops to each authorized mobile end user.

The IPsec approach generally requires third-party software and possibly their specialized hardware. This adds an extra layer of security by requiring each client to have a specific and specifically configured client application installed to use the VPN. Usually this also requires on-site services and financially costly per-seat licenses and per-person laptops for mobile users.

Each implementation expresses its unique advantages and disadvantages in different usage scenarios—never mind the convincing technical details. Where IPsec may create complications between communicating business partnerships, requiring that both agree on a vendor-specific platform and mutual configurations, SSL is much less demanding and ideal for mobile users just checking email. They are not, however, mutually exclusive technologies and can be used together to satisfy a variety of usage scenarios and client conditions.

SSH VPNs

The SSH protocol was not originally designed for use as a VPN, but as mentioned earlier, SSH may be and often is used to tunnel specific protocols end to end between pairs of computers. For example, you can use the SSH TCP port forwarding facility to tunnel the insecure FTP from one computer over an untrusted network to another computer. This provides a fully secure connection that neutralizes FTP's intrinsically insecure features (lack of encryption and authentication, plus lack of protection for user credentials and message content). Likewise, homegrown file transfer solutions or proprietary solutions such as those built around HTTP (usually to make them easily accessible to Internet users, all of whom have Web browsers) are also easy to secure with SSH without requiring any alterations to whatever programs may already be in use.

Implementation Approaches

SSH provides the same levels of security that other solutions offer—most notably, those based on IPSEC or SSL/TLS—but requires considerably less effort to implement, particularly because its use means that existing scripts, programs, and networking infrastructure need not be changed or replaced.

In addition, the SSH protocol may be transmitted across IPsec VPNs without requiring any additional work. This helps to explain the common belief that SSH is probably the most network-friendly security protocol around.

Normal practice is to set up SSH so that end-to-end security is provided. However, IPsec VPNs are most often used in gateway-to-gateway mode to provide a secure link from one network to another, whereas SSH typically provides secure links between a specific client and a particular server, thereby providing a more secure implementation.

Other VPNs

Although IPsec and SSL/TLS make excellent choices for a VPN, and SSH serves admirably for this purpose (particularly when the necessary client software elements are deployed to end users), they are not the only options available. PPTP, L2TP (including versions 2 and 3), MPLS, the Cisco-specific Layer 2 Forwarding (L2F) protocol and Multi-Path VPN are other usable forms of VPN technology. Some large ISPs include managed VPN services for business customers and take away the administrative overhead of operating such technology.

Securing FTP

At this point, you have an idea of both *why* you should use encryption and *what* can be used for this purpose. Securing FTP now becomes a question of *how*. The sections that follow offer many possible and practical solutions to at least partially answer this question. Ultimately the choices will be site-specific, calling upon existing IT knowledge and leveraging current infrastructure trends to adapt the network environment to a more secure workplace.

SSH

SSH provides a set of standards and associated network protocols to establish secure communications channels between endpoints, often across media that cannot be reasonably secured nor is guaranteed to be secure (such as the Internet). SSH uses public key cryptography for authentication, provides confidentiality and integrity for data exchanges, and utilizes message authentication for added tamper-proofing.

SSH was originally developed for secure systems administration on UNIX servers. But since its inception in 1995, it has increasingly been used to implement secure file transfer solutions as well. Today, many users worldwide employ SSH as a preferred method for secure file transfer on a wide range of computing platforms from Windows PCs to IBM mainframes.


The SSH protocol provides a set of standardized applications that creates an easy migration path to secure file transfer protocols and services. One key SSH-based application is SFTP, an application that is quite similar to FTP but that replaces FTP's insecure features with more secure alternatives. SCP is another key SSH-based application, modeled on the UNIX CP (Copy) command.

Users can choose either of these applications with confidence because SSH ensures strong authentication not only for individual users but also for the computers involved. Also all data transmitted between communication partners (including passwords) is automatically encrypted using advanced algorithms such as the Advanced Encryption Standard (AES) or Triple DES (aka 3DES, a mode of the Data Encryption Standard—DES—that encrypts all data three times over).

SFTP

SFTP is another network protocol and component of the primary SSH-2 design. This protocol facilitates secure network-based file transfer. In addition, the SFTP subsystem of SSH supports a complete set of file and directory operations, which makes it an entirely workable and reliable remote file system protocol.

SFTP shares many properties with FTP and supports all the latter's common capabilities, including directory listings, put/get file, file rename, and file deletion. Because SFTP is a subset (or subsystem) of the SSH protocol, it is platform independent and suitable for just about any computer that can operate an SSH client or server. Today, SFTP is available on virtually any computing platform from a Windows PC to an IBM Mainframe. SFTP is not, however, FTP simply shuttled via SSH; it is, in fact, a new and separate protocol designed from the ground up by Tatu Ylönen and standardized by the IETF SECSH working group (RFC4250-4254).

 Find more information about SFTP solutions based on the SSH protocol at www.ssh.com/solutions/applications/sftp.html and <http://www.ssh.com/products/efl/usage.html>.

Although SFTP has been available for more than 10 years, FTP is still widely used in many organizations despite its many drawbacks. There are many reasons why the use of FTP persists; it is clear that chief among them is that many organizations have built their applications and file transfer systems around FTP. Thus, moving away from this legacy is difficult and often requires considerable development effort. Some organizations have thousands of scripts to control file transfers on a daily basis: switching from insecure to secure implementations can thus involve immense effort to effect such changes.

Some vendors have created special software tools to help organizations make the transition to SFTP systems less difficult and resource consuming. Automated supports like these lower the burden of securing file transfer and make the conversion to secure systems significantly faster and cheaper.

SCP (Secure Copy)


SCP (Secure Copy), another part of the SSH suite, permits only file transfers. This makes it ideal for hands-off automated tasks and scheduled periodic bulk file transfers. SCP makes no provision for authentication or security but relies upon the underlying SSH protocol to handle these tasks.

During upload, a client supplies the server with content and can optionally include basic file attributes such as permissions and timestamps. FTP cannot handle these tasks. During a download, a client requests files or directories to be obtained from a server and the server does the rest, which turns the operation into a server-driven transaction.

FTPS (aka FTP/SSL)

FTP via SSL is yet another method for securing file transfers across the network. Information can be encrypted for the data channel, the control channel, or on both channels. When the data channel remains unencrypted, the protocol is said to be using a clear data channel; likewise, when the control channel is unencrypted, the protocol is said to be using a clear command channel. Interoperating FTP over SSL is defined in three ways:

- **SSL connect**—A connection is made to a separate port, typically 990 according to the Internet Assigned Numbers Authority (IANA), and the SSL negotiation is performed.
- **AUTH SSL**—A connection is made to port 21 in normal fashion, and AUTH SSL or AUTH TLS-P is issued to request SSL negotiation to implicitly protect any further transactions.
- **AUTH TLS**—A connection is made to port 21 in normal fashion, where AUTH TLS or AUTH TLS-C is provided in request for SSL negotiation to explicitly protect the ongoing data transaction.

 RFC 4217 fully describes the concept, operation, and procedures for FTP over SSL. Check out <http://tools.ietf.org/html/rfc4217> for more information.

Of these, the first two are deprecated and not recommended for usage, mainly owing to their lack of standards conformance. Implicit SSL protection of the FTP session (the first item in the previous list) occurs upon connection for control and data connections. The second method, AUTH SSL, permits a connection to negotiate a connection upgrade to SSL security. Once the control connection is secured, the data connection is secured, but the approach so thoroughly clashes with other RFC standards that its use is also not recommended.

The third option, AUTH TLS, is the method of choice according to RFC standards. It makes a client connection in typical fashion to port 21 on the target host and begins an unencrypted dialog with the FTP service. It requests that TLS security be used, performs the TLS Handshake process, and then issues any sensitive data over a secured channel.


Other Approaches

Current examples up to this point are relatively straightforward and intuitive: there exists a simple synergy between client and server applications with regard to previously described secure file transfer technologies. Other implementation approaches may seem unusual, unorthodox, and perhaps unnecessary for most uses. The next sections briefly investigate these remaining items, if only to give a pointer in one of the many tangential directions you may pursue.

Tunneling with SSH

SSH tunneling uses the concept of port-forwarding to achieve its goals. Highly restricted networks normally prohibit the passage of traffic to specific sites, specific ports, specific protocols, or some combination thereof. Port-forwarding helps to bridge the gap between some of these highly restricted destinations by establishing a gateway machine to act as a go-between. This machine, which could in practice be the same computer on which an SSH server is housed, brokers requests from client to server using whatever legal port pairing may be required.

A forwarding agent is established to listen on some presumably unrestricted port; for example, port 21. The forwarding agent is then directed to contact some given host—acme.com port 22, to continue this example—once a connection is established on port 21. This naïve illustration serves to demonstrate the basic nature of SSH tunneling but explains nothing of the more exotic arrangements and exciting scenarios to which such tunneling applies.

 See “Tunneling Explained” on the SSH Web site for more information about how to set up various tunneling, or port forwarding, arrangements using SSH (it’s available at <http://www.ssh.com/products/tunneling/usage.html>).

SSH tunneling is versatile and powerful and has been exploited by numerous software vendors. As described earlier, many organizations have performed file transfers for a long time and have invested heavily in setting up a file transfer infrastructure, often one that is based on FTP at least in part. Securing such legacy file transfers is important but can be resource intensive and costly.

The Push/Pull Concept

Understanding the push-pull concept is perhaps only marginally necessary to utilizing push-pull strategies and technologies in the workplace. After all, not every driver must possess an understanding of the four-cycle combustible engine to operate a motor vehicle. It is helpful knowledge, but not a prerequisite to operate the vehicle.

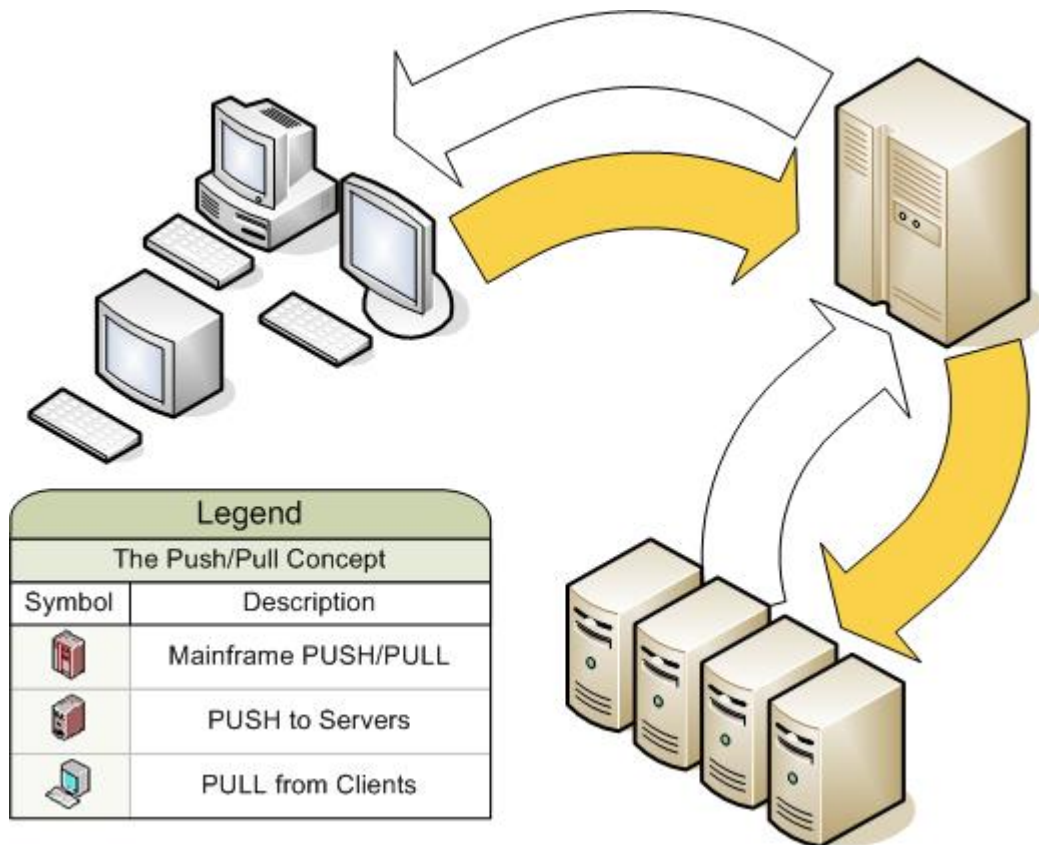


Figure 3.5: The push-pull concept illustrated here shows client computers pulling data from a mainframe computer also pushing data to servers.

In the *push* swing of the pendulum, the recipient need not request data transfer from the sender; the sender supplies what is wanted, perhaps even on a predetermined schedule. This might occur as some part of an automated mandatory update of integral software components, for example.

During the *pull* swing of the pendulum, the client requests a specific file or set of files to be transferred on *its* schedule, and pulls it through the delivery channel. This might occur as the client directly requests to have a software component updated.

Automated File Transfers

File transfer automation is the loyal servant of industrial technology. It is through scheduled automation that all routine and monotonous tasks may be handled consistently, correctly, and in conformance with a well-defined schedule. The best-known and understood examples include such things as regular backups (which may be full system backups or various types of incrementals) and database or directory replication, all of which typically involve large file or data transfers across the network. The key to keeping automated file transfers working properly is to add security envelopes (such as IPsec, SSH tunnels, or other forms of encryption/decryption technology) around those file transfers that cannot simply be replaced with more secure forms of file transfer such as SFTP.

Keyed Logins for Transfer (No Password)

Keyed logins are ideal for several reasons, yet face many of the same security implications as do typical passwords or passphrases. Keys are better than passwords or passphrases because key strength in both symmetric and asymmetric forms is many times stronger than passwords or passphrases. Whereas a password or passphrase seldom exceeds 128-bits or 16-bytes, as with a 16-letter password, secure storage and access mechanisms make it easy to support key lengths from 1024 bits through 4096 bits (if not larger) for short transfers (shared secrets, certificates, authentication data) where security concerns outweigh computational overhead. In turn, this permits regular exchange of message digest hashes and other key mechanisms to encrypt streams of data over the short term between communications partners so that compromising any single portion of the stream (theoretically possible, but unlikely in the timeframe between key changes) will still not compromise overall communications confidentiality and integrity.

Symmetric cryptography uses the same key to encrypt and decrypt data, whereas asymmetric cryptography uses a combination of public keys for encryption and private keys for decryption. Both are equally secure but asymmetric methods require more effort in practice. Keys need to be digitally signed and exchanged through a verified third-party entity or other neutral medium:

- SSH keys
- SSH keys with passphrase
- RSA SecurID cards

Keys can also be in the form of digital key fobs such as RSA Security's SecurID mechanism for two-factor authentication. These user-assigned hardware tokens generate authentication codes in periodic intervals using a factory-encoded random key, known as the seed. Each seed is different among SecurID devices and is typically 128-bits long.

Key-agreement protocols are effective where two parties agree on a key and both are influential to the outcome. The first and perhaps foremost public protocol established for this trend is the Diffie-Hellman exponential key exchange, where two parties jointly agree upon a generator composed using random numbers. In itself, this specifies no prior agreement or subsequent authentication between participants; it is therefore considered an anonymous key agreement protocol and remains susceptible to Man in the Middle (MitM) attacks.


Public key exchanges defeat such attempts using digitally signed keys, the very integrity and identity of which is sealed and delivered by a trusted third party and optionally signed by a CA. This mechanism is widely used on the Internet for SSL/TLS-based HTTP and FTP transactions.

Keyed Logins for Transfer (With Password)

Although it is entirely possible to password-protect a key, it may not be entirely useful for every situation. In this case, the password is usually an actual phrase and is therefore called a passphrase.

Passphrases have their arguable advantages and disadvantages. An ideal passphrase is generally better than a password because it is more than just a single word or word combination. It is an entire phrase of words, preferably of intermixed case and alphanumeric flavor, perhaps including symbols and non-printable characters for good measure. In fact, a passphrase can be much stronger (and often more difficult to recall) than a simple password. However, a passphrase that can be figured out with minimal computational effort (or by simple guessing) is no better than a poorly chosen password. You can have your public/private asymmetric keypair optionally secured with a password or passphrase, but such a step is futile if the passphrase is weak.

If this is so, why would you want to secure a key with a passphrase? One compelling reason is that, should the private key become disclosed, a proper password will keep it from ever being effectively used. Perhaps the key is kept on a USB thumb drive that is occasionally absentmindedly left in an unattended computer. In such a case, password or passphrase protection offers an additional layer of protection.

 Password selection is explained in further detail in the next section.

Securing Automated Transfers

Automation of file transfers is best facilitated by a form of public key exchange either by agreed Diffie-Hellman keys or Diffie-Hellman public key exchanges. Other key distribution systems of suitable strength and capability (RSA, DSA, ECC, El Gamal, and so on) are also worth consideration, but Diffie-Hellman is an established standard and primary example for this purpose.

Use asymmetric keys for protection and distribution of encryption keys and symmetric key algorithms for bulk automated transfers. This marriage of both technologies can better suit a variety of situations. Asymmetric keys can safeguard the symmetric keys from being purloined by unauthorized parties.

Preferred Authentication Methods

An ideal authentication method is one that gives strong server authentication and convenient user authentication. In practice, it may not happen this way; preferred authentication methods may sometimes appear architected to stymie most forms of end-user convenience. For instance, users do not generally consider it convenient to generate strong password choices more than 8 to 12 characters in length that include mixed-case letters along with alphanumeric or symbolic combinations (nevertheless, they are better off using such passwords, and should be well-informed as to the benefits and value that they deliver; it's often a good idea to offer access to a random password generator for those willing to use one who are also willing to observe the obligatory cautions against writing them down on a post-it note attached to their monitors). Realistically, the preferred methods are those that match many criteria, some of which are site- or installation-specific such as Microsoft's Encrypted Authentication or Extensible Authentication Protocol.

Even subtle differences to compile-time parameters or run-time configurations for simple SSH client and server combinations yield enough variance to warrant preferential treatment for some options and not for others. In one such case, the SFTP subsystem is largely built upon SSH protocol version 2 and is preferred over the many exposures and vulnerabilities in protocol version 1.

Perhaps the most obvious answer is to say, *stick with the standards*. Standardized, formalized, and ratified (in the case of IETF and RFC-related material) protocols, methods, and implementations are by far the best way to go. A preferred method should be mechanism- and implementation-independent and capable of interoperability between any number of platforms and applications. Such is the case for SSH, SSL/TLS, Diffie-Hellman, and other such security protocols. It should also be certifiably strong, validated to meet administrative and/or governmental criteria for secure data transmission or user authentication, ensure the high standards of privacy and confidentiality, and be relatively simple to use and deploy in any normal capacity.

Using Public-Key Mechanisms (With and Without Certificates)

PKI consists of applications, formats, procedures, protocols, policies, and public key cryptography mechanisms jointly working to ensure predictable, reliable, and secure communication channels. PKI establishes a level of trust within a potentially hostile and untrustworthy environment by providing for trusted third-party vetting of and vouching for user identities and binding of public keys to public entities.

This framework is established to provide authentication, confidentiality, non-repudiation, and integrity for messages in an exchange and is a hybrid system consisting of both symmetric and asymmetric key algorithms and methods. Like IPsec, PKI is only a framework and it also lacks specification of protocols and algorithms for secure exchange. As a framework, it consists of many separate useful components that include CAs, registration authorities, certificates, keys, and end users.

Public key cryptography is not the same as PKI; it's simply another name for asymmetric encryption algorithms, where PKI is an entire infrastructure. Public key cryptography can be viewed as a component of PKI, but they remain separate elements. Other components of this infrastructure or framework help to ensure user identities through certificate signing and the Diffie-Hellman exchange protocol for the negotiation of key exchanges. These pieces identify users, create and distribute certificates and encryption keys, maintain and revoke these certificates, and orchestrate the interoperation of all these implementation goals.

Where PKI will provide the means for handling certificates during a public exchange, basic public key cryptography does not make such allowances. Both methods are secure to use on an everyday basis, but the proper application of each method will be specific to the particular situation and any security considerations at hand.

Avoid Embedding Passwords in Scripts or Communications

Occasionally laziness and/or errant thinking will cause an administrator to make mistakes such as providing passwords for accounts in scripts or text files marked as readable by others. End users regularly put such critical information on sticky notes littering their cubicles; whether or not such exposure is the fault of the administration staff, it remains their responsibility.

Please resist the urge to embed any password within any script or leave it publicly readable in any case, even if you can cleverly disguise its meaning, obfuscate its form, or prohibit its accessibility. Although it may briefly alleviate administrative strains, such as reminding a user what their password is for the umpteenth time, it will increase administrative discomfort when someone else discovers this crucial piece of information and takes advantage of it in some unauthorized manner.

Avoid Weak Passwords

There are many instances where you must issue passwords for authentication purposes. Given all other security precautions for set up, management, and tear-down of cryptographically secured communications, strong password choices must still be made to ensure the confidentiality of each end-user transmission.

Suppose a public HTTP or FTP server provides basic login features for your clients, and it is deemed impractical to issue proper public/private keypairs for each one. In this case, you should take several extra proactive precautions against poor password selection.

Typical usage scenarios involve a fixed-length password of mixed-case alphanumeric passwords. For example, it is common to see password fields restricted to a minimum of 8 to 12 mixed characters. The mixed-case component becomes apparent with case-sensitive systems where UNIX and Unix represent two different forms of the same word. This case-sensitivity and mixture with digits and symbols greatly increases the variability of the password by offering extended combinations and permutations. This decreases susceptibility to brute-force dictionary attacks, in which trial-and-error sessions expose easily guessable password choices.

Also, many modern login systems—beyond interactive prompts—provide some means of cross-checking the end user's password choice. Items deemed unsuitable for general-purpose or production use are issued warnings, and the user is commanded to make a stronger choice. Such systems enforce correct behavior for every password whether being newly created or changed from an existing one.

The following list highlights a few pointers for strong password selections:

- Mix numbers and letters; mix case; include punctuation and symbols for added strength
- Merge one or more words side-by-side or interleaved; in the second example, two words UPPER and lower might become UIPoPwEeRr
- Combine a root word with an appendage: the root word need not be a dictionary term, but it should be pronounceable; the appendage is either a prefix or suffix, such as prefort or forttable, where pre is a prefix and able a suffix attached to the same root word, fort
- Choose an arbitrary phrase and select specific letters from each word; for example, mess with the bull, get the horns becomes mWTbGth
- Find something memorable but nothing too personal that someone else could easily discover, such as significant dates, personally identifiable information, pet names, and so on
- Never use the same password for multiple logins; once an unauthorized party discovers this universal password, guessing the login or account name is a simple hit-or-miss affair at popular online retailers, payment systems, and online banks.

The most important thing to remember is that even the strongest fortifications can be infiltrated where the keys are effortlessly guessed or reproduced. For all the ironclad security properties provided by the best cryptographic algorithms, implementations, and frameworks, each one is only as strong as its weakest key—or in this case, password or passphrase.

Summary

This chapter explored the cryptographic protocols and frameworks necessary to construct reliably secure communications channels. IPsec, SSL/TLS, and SSH are just a few of the standardized solutions to meet the need for secure communications. Some of these solutions represent only part of the answer because they require other components to function in any capacity (for example, IPsec and SSL/TLS versus SSH). Some of these solutions may integrate and work with others or present an alternative choice to another (for example, IPsec versus SSL/TLS VPN). The SSH protocol covers secure communications as well as secure file transfer and will work over an IPSEC or SSL/TLS link.

The chapter also discussed that although file transfer itself is closely associated with the file transfer protocol, not every file transfer is facilitated by FTP. Many secure file exchanges occur over HTTP and this is another area where SSL/TLS is commonly applied. Files may be transferred over a number of technologies, some of which may or may not inherently support SSL/TLS, IPsec, or other cryptographic components that make for secure transfers. Tunneling and VPN technologies become an attractive option for an all-encompassing solution to secure legacy applications or those that do not directly work with typical cryptosystems and cryptographic communications protocols.

Finally, the chapter talked about the strengths of public key-based cryptosystems including:

- Proper key exchange is a vital component in trusted party-to-party communications across shared channels
- Diffie-Hellman first defined the public/private keypair concept, but many other worthy key algorithms like RSA, DSA, ECC and El Gamal exist
- The strength of a cryptosystem comes from the algorithm, initialization vectors, and the length and secrecy of the key

The next chapter further explores the subjects of SFTP, FTPS, and IPsec with more attention to comparison and contrast between these technologies.

Download Additional eBooks from Realtime Nexus!

Realtime Nexus—The Digital Library provides world-class expert resources that IT professionals depend on to learn about the newest technologies. If you found this eBook to be informative, we encourage you to download more of our industry-leading technology eBooks and video guides at Realtime Nexus. Please visit <http://nexus.realtimepublishers.com>.