Sonicwall SSL Offloaders Frequently Asked Questions

Troubleshooting/Error Messages:

I can’t establish a connection via the console port.
The Configuration Manager doesn’t see any devices.
The Configuration Manager sees the device, but I can’t connect to it.
After initially setting the SSL-x’s IP address I get a session timeout and disconnect.
I enter a default route on the SSL-x, but it doesn’t appear in the configuration.
“Failed to send set ssl server command! Type "show messages" for more information.”
“Failed to delete ssl server id [x] from device.”

General

What is SSL?
SSL, or Secure Sockets Layer, is a network security mechanism introduced by Netscape in 1995. In March of 1996, Netscape released SSL version 3.0—the standard still in use today—which it designed as a tool "to provide privacy between two communicating applications (a client and a server) … [and also] to authenticate the server, and optionally the client." SSL's most popular application is HTTPS, designated by a URL beginning with *https://* rather than simply *http://*, and it is recognized as the standard method of encrypting web traffic on the Internet. An SSL HTTP transfer uses TCP port 443, whereas a regular HTTP transfer uses TCP port 80. Although HTTPS is what SSL is best known for, SSL is not limited to securing HTTP, but can also be used to secure other TCP protocols such as SMTP, POP3, IMAP, and LDAP.

SSLv3.0 was designed to maintain full backward compatibility with SSLv2.0, while adding the following features:

- Alternate key exchange methods, including Diffie-Hellman.
- Hardware token support for both key exchange and bulk encryption.
- SHA, DSS, and Fortezza support.
- Out-of-Band data transfer.

The Goals of SSLv3.0 were:

1. Cryptographic Security – To be used in establishing a secure connection between two parties.
2. Interoperability – Independently developed applications utilizing SSLv3.0 should be able to seamlessly exchange cryptographic parameters.
3. Extensibility – SSLv3.0 should provide a framework into which new public key and bulk encryption methods can be imported, thereby preventing the need to create new protocols, and avoiding the need for new security libraries.
4. Relative Efficiency – Due to the highly CPU intensive operation of public key cryptography, SSLv3.0 introduces session caching schemes to reduce connection setup overhead.

The SSL protocol itself actually comprises two layers, namely the **Record Layer**, and the **Handshake Layer**. The Record Layer is responsible for the transmission of data during an SSL session. The Record Layer has only a single protocol, the Record Protocol, whereas the Handshake Layer has three protocols:

- **change_cipher_spec** – a single byte message sent to and from both client and server indicating that subsequent records will be encrypted using a the agreed upon (pending) enciphering algorithm.
- **alert** – a two byte message used to convey alerts of varying severity. The first byte can carry either "warning" or "fatal" messages, and the second byte contains

specific failure messages. If a "fatal" message is received, the connection, but not the session, is immediately terminated. This allows for connections to be renegotiated without having to go through the computationally intensive Handshake protocol session setup. This marks the distinction between an SSL session and an SSL connection; the connection is a logical link between peers that is dependent upon a negotiated session.

- **handshake** – the most complex protocol within the suite, and the reason why the first generation of SSL Accelerators came into being. The Handshake protocol defines the SSL session state through a series of communications between the communicating peers. An example of the handshake is as follows:

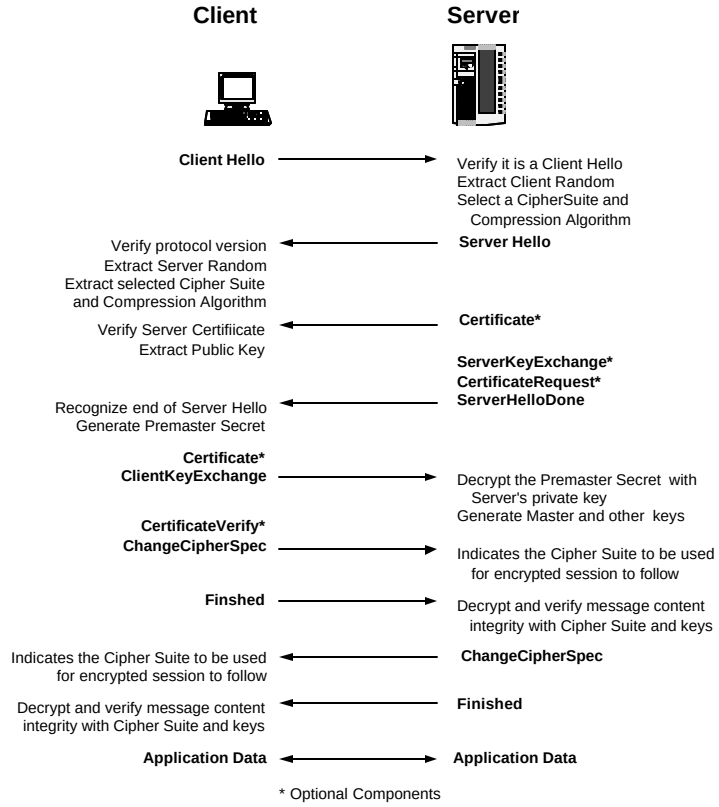

Figure 1 – The SSL Handshake.

As figure 1 illustrates, an SSL session is started following a successful TCP connection, and begins with a **Client Hello**. The client_hello comprises the following information:

- Version – The version of SSL that the client wishes to use in communications. This is usually the most recent version of SSL supported by the client.
- Random – a 32-bit timestamp coupled with a 28 byte random structure.
- Session ID – This can either be empty if no session_id data exists (essentially requesting a new session) or can reference a previously issued session_id.
- Cipher Suites – A list of the cryptographic algorithms, in preferential order, supported by the clients.

- Compression Methods – A list of the compression methods supported by the client (typically null).

The server then processes the information received in the client_hello and issues an alert upon failure, or a **Server Hello** upon success. The server_hello is similar to the client_hello, the biggest difference being in the Session ID portion. If the client_hello contained a session_id, the server will look up the offered session_id in its session cache. If the server finds the ID in its cache and determines it can be resumed, it will respond with the same session_id. This will indicate a resumed session, and the handshake then moves immediately to the finished message. If no match was found in the session cache, the server will issue a new session_id.

Next is the **Server Certificate**. The server sends its certificate, typically X.509v3, to the client in a format that abides by the advertised Cipher Suite's supported key exchange algorithms. Subsequently, if the server requests a client certificate, the same process occurs. Verification of the server certificate involves checking the following:

- Is the certificate date range valid?
- Is the issuing CA trusted? The client checks whether the Distinguished Name (DN) of the issuing CA appears on its list of trusted CA's. If it does, the verification continues. If it does not, the client attempts chained certificated verification up to a level that is trusted. More on certificate chaining later.
- Does the Common Name (CN) of the certificate's subject match the name of the server?
- If any of the above three conditions aren't satisfied, the web-browser will issue a warning to the client about the discrepancy, allowing the user to continue or terminate the connection attempt.

Although unlikely, in instances where the server has no certificate, or no appropriate certificate, the server can optionally issue a **Server Key Exchange** message. This message contains hashes of the client and server hello random structures, and also key exchange algorithm specific data. Following this, the server can issue a **Certificate Request** to the client, if the server is so configured. The server then indicates the end of the server_hello with a **Server Hello Done** message.

The client may now begin to send data to the server. If requested, the client will provide a **Client Certificate**. If one was requested but none is available, the client can send a no_certificate alert to the server instead. Depending on the server's configuration, the connection may resume as normal, or it could be terminated by the server. The **Client Key Exchange** follows, employing the agreed upon key exchange algorithm. The most common key exchange algorithm is RSA. The RSA key exchange begins with the client generating a 48 byte pre-master-secret which it then encrypts with the server's public key extracted from either the *server certificate*, or from the *server key exchange*. The client then sends the result in an *encrypted pre-master secret* message to the server. The client version is incorporated into this message to prevent version rollback attacks.

It is the generation of the pre-master-secret, the conversion of the pre-master-secret into the master-secret, and the conversion of the master-secret into keys and MAC secrets that are so computationally intensive as to necessitate SSL offloaders.

The client may now optionally send a **Certificate Verify** message, which is used in cases where the client submitted a certificate with signing capabilities. Finally, the client sends a **Change Cipher Spec**, and an encrypted **Finished** message, indicating that the key exchange and authentication processes were successful, and that all subsequent communications will be encrypted with the agreed upon algorithms. The server then verifies the *finished* message received, sends a *change cipher spec* and *finished* message of its own. **Application Data** can now be transmitted by the Record Layer between the two parties.

Summarily, as a result of the handshake, the following session state information is determined:

- Session Identifier – the SSL Session ID is a byte sequence chosen arbitrarily by the server, and used by the server to track sessions. Additionally, the Session ID has come into use by Internet Traffic Management devices (load balancers, and content switches) to help maintain session persistence across multiple devices.
- Certificate Exchange – an X.509v3 certificate can be sent to and from both peers to provide identity authentication. In a typical SSL session, the peer certificate is only requested from the server so that the client may verify that the server is who server claims to be. All versions of the X.509 certificate contain the following information:
    - Version – Identifies the version X.509 (current 1, 2, or 3)
    - Serial Number – A unique serial number assigned during creation.
    - Signature Algorithm Identifier – The algorithm used by the CA to sign the certificate (such as MD5RSA).
    - Issuer Name – The name of the entity that signed the certificate. This is usually a well-known Certificate Authority, such as Verisign, or it can be a self-signed root CA.
    - Validity Period – The valid from and valid to dates of certificate.
    - Subject Name – The X.500 standard Distinguished Name (DN) representation, such as:
      CN=www.sonicwall.com,OU=SonicwallSLC,O=Sonicwall,C=US (CN is Common Name, OU is Organizational Unit, O is Organization, and C is the Country.)
    - Public Key – The subject's public key and an algorithm identifier specifying the public key algorithm in use (such as RSA – 1024 bits).

  X.509v3 added support for extensions that could be used to define the behavior, or intended purpose of a certificate, such as "keyCertSign", which means that the certificate can be used for signing only.
- Compression Method – Specifies the algorithm used to encrypt the data. This is typically null.
- Cipher Spec – Specifies the algorithm used for bulk data encryption (such as RC4, DES, or 3DES) and the hash algorithm (such as MD5 or SHA-1). Also defines cryptographic attributes such as hash size, and initialization vector (IV) size.

- Master Secret – The 48-bytes shared-secret key used for bulk-encryption.
- Is Resumable – Specifies whether this session can be used to create new connections, as defined earlier.

Through the function of the Record Protocol, the Record Layer encapsulates the three different protocols of the Handshake Layer, as well as the high-level protocol data of the application layer, such as HTTP. The Record Protocol operates on high-level data as follows:

1. **Fragmentation**. Higher-layer data to be transmitted is broken into blocks no larger than 2^{14} (16,384) bytes.
2. **Compression**. Although the protocol supports compression, neither SSLv3.0 nor TLS (SSLv3.1) specify a compression algorithm, so none is currently applied.
3. **MAC Computation**. A Message Authentication Code is calculated from the data. The MAC is a message digest, or a one-way hash code that is fairly easy to compute, but which is virtually irreversible. In other words, with the MAC alone, it would be theoretically impossible to determine the message upon which it was based. It is equally impossible to find two different messages that would result in the same MAC. If the receiver's MAC calculation matches the sender's MAC calculation on a given piece of data, the receiver is assured that the data has not been altered in transit.
4. **Symmetric Encryption**. The sender's data and the accompanying MAC are encrypted using the shared-secret key, determined by the Handshake Layer, employing the encryption algorithm also determined by the Handshake Layer.
5. **SSL Record Header**. The record header provides four pieces of information:
    - Content Type – indicates the source high-level protocol providing the data. This field can designate any of the three previously mentioned Handshake Layer protocols, or application_data is the data source was HTTP, for example.
    - Major Version – indicates the major version of SSL in use, e.g. "3" for SSLv3.0.
    - Minor Version – indicates the minor version of SSL, e.g. "0" for SSLv3.0.
    - Length – the number of bytes in the block.

Some of the areas touched-upon above deserve further explanation. In particular, we will look at Public-Key Cryptography, Message Digests, and Symmetric Cryptography as employed by SSLv3.0. Before delving into these areas of modern cryptography, however, the following section offers an abridged history of cryptography. This primer serves as a good point of contrast to modern cryptography, and also helps to illustrate the complexity of even early cryptographic methods. More than anything else, it helps us to appreciate the role of the computer in securing communications.

[return to top]

A Brief History of Cryptography:
Traditional cryptographic systems, or private-key cryptography, have been around in one form or another for centuries. A few of the simpler historical cryptographic systems will be explained here as an introduction to cryptography, and to offer a point of contrast with modern cryptography.

The Caeser Cipher
One of the oldest known crypto systems, the Caeser Cipher, dates back 2000 years, and used a simple substitution, or shifting scheme:

Plain:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Key:	EFGHIJKLMNOPQRSTUVWXYZABCD

So the plain-text message "There is a moose in my backyard" would produce the cipher-text "XLIVI MW E QSSWI MR QC FEGOCEVH." Substitution ciphers appeared in other, slightly more complicated formats, including various types of mono-alphabetic substitution ciphers that used mixed, non-duplicated 26 letter keys. This method offered 26! keys (4×10^{26} keys) and was for a long time thought to be secure. (Note: the above example is not a "classic" Caeser cipher. The classic Caeser cipher used a three to the right modulo 26 character shift, where "A" would become "D", "B" would become "E", etc. Our example uses a four to right modulo 26 character shift, but is still considered a Caeser cipher.)

Cryptanalysis, or the deciphering of cryptographic messages, was formally introduced as early as the 9th century. Early cryptanalysts described a method called *frequency analysis* that could be used to decode messages encrypted with substitution routines. Frequency analysis, as applied to the English alphabet, yields the following:

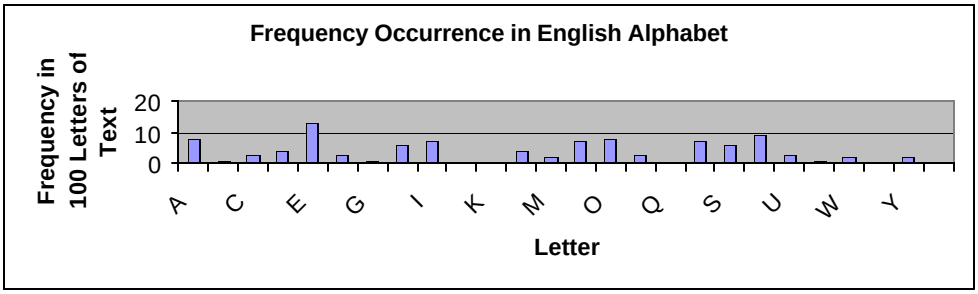

Figure 2 – Frequency Analysis of the English Alphabet.

The vulnerability of mono-alphabetic ciphers to frequency based cryptanalytic attack was addressed with poly-alphabetic ciphers, the best known of which is the Vigenère Cipher. The Vigenère cipher was introduced in the 16th century by French diplomat Blaise de Vigenère, and was basically an implementation of the Caeser cipher combined with a

keyword. For our example, we'll use the keyword "KEY" and repeat it for the length of our plain-text message:

Plain-text: `ThereIsNoSpoon`
Key: `KEYKEYKEYKEYKE`

Translation Alphabet:

```
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
K → CDEFGHIJKLMNOPQRSTUVWXYZAB
E → FGHIJKLMNOPQRSTUVWXYZABCDE
Y → DEFGHIJKLMNOPQRSTUVWXYZABC
```

To apply, if the plain-text letter "L" appeared over the key letter "K", we would use the "K" translation alphabet, and substitute an "N" for the "L". If the "L" appeared over the key letter "Y", we would substitute an "O" for the "L", etc. Applying the key and translation table to the plain-text above would produce the following :

Cipher-text: `VmhtjLuSrUurqs`

Clearly, the Vigenère cipher is not immediately susceptible to frequency analysis because of the multiple shared translation alphabets that are used in connection with a shared key. The longer the cipher-text, however, the more likely it is that identical sequences of characters will occur at a distance that is a multiple of the key length. This could lead to a reasonable guess of the length of the key. Once the key length is known, the system becomes considerably more vulnerable to attack.

Dozens of other ciphers and enciphering mechanisms were introduced throughout history, most notably the Playfair cipher, invented by Lyon Playfair and Charles Wheatstone in the 19th century, and the Hill Cipher from Lester Hill. Although there have been many other well known cryptographic mechanisms, such as the Enigma machine, the Affine Cipher, and Purple, they have been excluded for reasons of complexity and brevity.

The Playfair Cipher

The Playfair cipher requires that the sender and receiver share a key word, which can be any length. If the keyword has duplicated letters, duplicates should be omitted. For our example, let's use the key word "BEANS". Next, a 5x5 cipher-square is created, beginning with the key word, and continuing with the remainder of the alphabet. The letters "I" and "J" are combined into a single square. The result would be the following:

B	E	A	N	S
C	D	F	G	H
I/J	K	L	M	O
P	Q	R	T	U
V	W	X	Y	Z

The plain text message to be encoded is then broken into digraphs, or pair of letters. If a single digraph contains the two of the same letters, it should be broken into two digraphs, and an "X" should be added to both. Similarly, if the plaintext contains an odd number of letters, the final letter should be padded with an "X" to create a digraph. Take the following plaintext message: "The monkey stole my shoes". This would produce the following series of digraphs: "TH-EM-ON-KE-YS-TO-LE-MY-SH-OE-SX". The rules for replacement using the cipher-square are as follows:

- If the digraph letters appear in the same cipher-square row (like "BE" or "AS") they are replaced by letters immediately to their right, wrapping around when necessary. "BE" would become "EA", and "AS" would become "NB".
- If the digraph letters appear in the same cipher-square column (like "IP" or "SO") they are replaced by letters immediately beneath them, wrapping around when necessary. "IP" would become "PV", and "SO" would become "HU".
- If the digraph letter appear neither in the same row nor column in the cipher-square (like "AP" or "FO") you would:
    - Look along the row in which the first letter appears until you reach the column of the second letter. The letter of intersection is used to replace the first letter. ("A" would become "B")
    - Look along the row in which the second letter appears until you reach the column of the first letter. The letter of intersection is used to replace the second letter. ("P" would become "R", making "AP" into "BR")

Our plaintext, enciphered using the above cipher-square would be:

Plaintext:	THEMONKEYSTOLEMYSHOESX
Ciphertext:	UGNKMSQDZNUMKATNHOKSAZ

Decryption would involve the same operation in reverse. Construct the cipher-square using the shared secret keyword, break the message into digraphs and replace according to the same rules, this time shifting to the left rather than right, and up rather than down.

The Hill Cipher

The Hill cipher is a polygraphic system that depends upon matrix multiplication and modulus arithmetic. A matrix is an array of numbers with a certain number of columns and rows. In a Hill cipher application, the matrix will always have an equal row and column count. Modulus arithmetic, in extremely simple terms, concerns itself with remainders of division. Our example of the Hill cipher will use the 26 letters of the English alphabet, so we will perform our multiplication operations modulo 26 or (mod 26). If any of the products of our multiplication exceed 26, we will divide that number by 26, and our remainder will replace the number. For example, if we produce the number 32 as a result, 32 (mod 26) = 6, so 32 will be replaced by 6.

The Hill cipher is not bound to the 26 letters of the alphabet, but can also include numbers and symbols. If the numbers 0 through 9 were added, we would then operate (mod 36), but for our example, we will use the alphabet alone.

We begin by assigning a numerical value to every plaintext letter:

A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
0	1	2	3	4	5	6	7	8	9	10	11	12
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
13	14	15	16	17	18	19	20	21	22	23	24	25

Plaintext letters are then grouped into *n* letters, for our example we will use n=2 or the simplest form of the Hill cipher, the Hill 2-cipher. Our resulting enciphering matrix, or key, would be 2x2. Selecting the members of the matrix cannot be done arbitrarily, but must abide by a series of requirements whose explanations are well beyond the scope of this FAQ. For more information on matrix construction as applied to enciphering and deciphering with the Hill cipher, read on concepts such as determinants and matrix inversion, or refer to a text on Linear Algebra.

Let us use the matrix $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ whose determinant is 1, and is easily invertible (can be used to construct a deciphering matrix) and apply it to the following:

Plaintext: `CheeseAndEggs`
Plaintext: `C H E E S  E A N  D E G G S  A`
Translated: `2 7 4 4 18 4 0 13 3 4 6 6 18 0` (0 added for padding).
In *nxn* pairs: `(2,7)(4,4)(18,4)(0,13)(3,4)(6,6)(18,0)`

We now take the *nxn* pairs, and multiply them by the key matrix. Matrix multiplication is performed by multiplying a row's members by a column's members, adding the results, and moving in this fashion through the matrices. For example, our first equation below would be as follows: (2x3)+(7x4) = 6+28 = 34 (our first member) and (2x5)+(7x7) = 10+49 = 59 (our second member). (34,59) mod 26 = 34/26 = 1 remainder **8** and 59/26 = 2 remainder **7** = (8,7).

[2 7] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (34,59) mod 26 = (8,7) = IH

[4 4] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (28,48) mod 26 = (2,22) = CW

[18 4] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (70,118) mod 26 = (18,14) = SO

[0 13] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (52,156) mod 26 = (0,0) = AA

[3 4] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (25,43) mod 26 = (25,17) = ZR

[6 6] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (42,72) mod 26 = (16,20) = QU

[18,0] $\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$ = (54,90) mod 26 = (2,12) = CM

Ciphertext: `IHCWSOAAZRQUCM`

To decrypt, you would apply the inverse of the key matrix: $\begin{bmatrix} 7 & 21 \\ 22 & 3 \end{bmatrix}$ to the Ciphertext.

And the Hill Cipher is trivial when compared to DES.

Conventional cryptographic systems are made stronger by the size of the key, and by the complexity of the operation performed to encipher the clear-text. The larger the key, the less likely it is to be guessed, and the more complex the cryptographic algorithm, the less vulnerable it is to most methods of cryptanalysis. Ultimately, no matter how large the key or complex the calculation, there remains one weakness to private-key cryptography: The private key must be shared between encoding and decoding parties. If someone manages to get hold of the key during the sharing process, it can always be used to decrypt the cipher-text.

[return to top]

Public Key Cryptography

To address the often challenging issues of key sharing and the aforementioned problems inherent in symmetric cryptographic systems, Whitfield Diffie and Martin Hellman in 1976 introduced the first Public Key Cryptography (PKC) System. A year later, Ronald L. Rivest, Adi Shamir, and Leonard Adelman introduced the RSA algorithm, the key exchange algorithm most commonly used in SSLv3.0. Unlike secret-key, or symmetric cryptography, public-key cryptography uses two keys—a public-key and a private-key—for encoding and decoding.

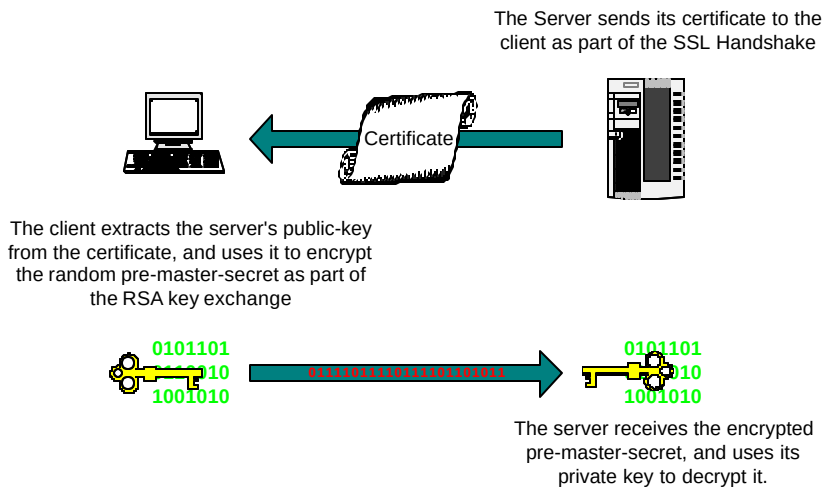

Figure 3 – An Excerpt from the SSL Handshake Illustrating PKC

PKC, as employed by SSL for encryption, uses the public-key for encoding a message, and the private-key for decoding the message. Even though it was encoded with the public-key, the public-key cannot be used to decrypt the message. The public-key can be shared with anyone, and regardless of who has the public-key, the security of the cryptosystem is not compromised. This is achieved through complex math involving calculations performed on large prime numbers. So complex are these calculations that even today's powerful processors can be quickly become overburdened with computations.

What follows is a greatly simplified example of RSA's calculations using very small prime numbers. Keep in mind that a 512-bit key is about 155 digits long, and 512-bit RSA is considered the "weak" version of the cryptosystem. In fact, RSA recommends using no less than a 768-bit key for personal use, 1024-bits for corporate use, and 2048-bit for extremely valuable keys.

> Start with two prime numbers, p and q, and multiply to find their product n (the *modulus*).
>
> $$p=7,\ q=11,\ n=(pq)=77$$
>
> Select the public exponent. This number, e, should be less than n, and should also be relatively prime to $(p-1)(q-1)$. This means that we are looking for a number that shares no common factors with (7-1)(11-1) = 6*10 = 60. The factors of 60 include 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60. Let's use the number 17, which happens to be prime, although public exponent primacy is not a requirement.
>
> Now we have to find a number, d, such that $(ed-1)$ is divisible by $(p-1)(q-1)$. A number that satisfies d is 53: 17*53=901. 901-1=900. 900/60=15.

e and *d* are the public and private exponents. The public-key will consist of the modulus and public exponent (*n* and *e*), and the private-key will consist of the RSA version number, the modulus (*n*), the public exponent (*e*), the private exponent (*d*), the two prime factors (*p* and *q*), the exponent (*d* mod (*p*-1)), the exponent (*d* mod (*q*-1)), and the coefficient (q^{-1} mod *p*).

The prime numbers chosen as *p* and *q* should be approximately one-half the length of the key size in use. On a 1024-bit key, then, the primes should be about 512 bits each, or about 155 digits long. Ever tried to find a 155-digit prime number? How about multiplying two 155-digit numbers together? Well, that's just the beginning, anyway. Now for the actual encryption and decryption with the keys. Let's use the public and private keys we just generated:

p=7 - The first prime number
q=11 - The second prime number
n=77 - The modulus – this number is roughly equivalent to a 6-bit key.
e=17 - The public exponent (key)
d=53 - The private exponent (key)

And let's use these values to encrypt the plaintext letter "A" which in decimal representation is 65. We do this by taking our plaintext (represented as "*t*") and exponentiating it by our public key (mod) the public modulus. The syntax is: t^e mod *n* or:

65^{17} mod 77.
65^{17} = 6,599,743,590,836,592,050,933,837,890,625
6,599,743,590,836,592,050,933,837,890,625 mod 77 = 32

So our plaintext letter "A", whose decimal value is "65" gets RSA encrypted with a 6-bit key as ciphertext "32". The syntax for decryption is (ciphertext represented as "*c*"): c^d mod *n* or:

32^{53} mod 77.
32^{53} = 5.92855496895058920568683443244448e+79 (best my calculator could do…)
5.92855496895058920568683443244448e+79 mod 77 = 65

And there we have RSA encryption and decryption of the letter "A".

This above helps to illustrate why RSA is considered a trapdoor function: It is simple to construct a function to generate a large number (the modulus) from two prime factors, yet it is infeasible to determine those prime factors from that modulus. This is what makes RSA, and PKC's in general so secure, and so processor intensive. The larger the modulus, the more secure the encryption. Why, then, don't we just use the largest calculable modulus? The modulus sizes chosen for most RSA operations—either 512 or 1,024 bits—was selected because they are balanced between strength and computational

cost. For encrypting extremely sensitive data, modulus sizes of 2,048 bits are sometimes used. Every time the modulus length is doubled, public-key operation time requirements (encryption and signature verification) increases by a factor of four, and private-key operation time requirements (decryption and signing) increases by a factor of eight.

RSA is not used for the entire SSL session because of its enormous CPU costs. Instead, SSL uses RSA's public-key encryption to encrypt and securely transport a shared-secret key for the SSL session to use for the bulk cryptography of the session. The shared-secret, or symmetric key that can be used is usually 40, 56, 128, or 168 bits (Rivest Cipher 2-4, DES, 3DES,etc.). The recommended minimum shared-secret bit length is 70-bits. DES, the Data Encryption Standard, operates at 56-bits, and has recently been replaced by AES, the Advanced Encryption Standard. The chosen AES algorithm is Rijndael (pronounced "Rhine Dahl") and it supports 128, 192, and 256 bit keys. There are no immediate plans to incorporate AES into SSL, but given SSL's extensible architecture, this could happen quickly.

For the sake of comparing symmetric and asymmetic cryptography, symmetric cryptography is generally considered at least 1,000 times less processor intensive than asymmetric cryptography, and, as the key bit-lengths suggest, it does not require nearly as large a key to be considered secure.

PKC can also be used for Digital Signatures by encrypting (or signing) a message with a private-key. Once encrypted with the private-key, only the public key can decrypt the message. Obviously, this implementation of PKC does not offer confidentiality; since anyone can obtain the public key, and anyone can decrypt the message. What it does offer is signer authentication, that is, it irrefutably indicates who encrypted the message.

[return to top]

Message Digests

Message digests are mathematical functions—also known as one-way hashes—that are easy to compute but virtually impossible to reverse. The message digest serves as a fingerprint of sorts for data. As such, it is an element of Digital Signatures, SSL, and most other data security mechanisms. The hashing function takes variable length data as an input, performs a function on it, and outputs a fixed length hash value. In addition to being non-reversible, a requirement of a strong one-way hash function is that it be collision-free. This means that no two inputs should result in the same output.

SSL commonly employs MD5 (Message Digest 5) and SHA-1 (Secure Hash Algorithm 1) for its hash functions. Both are derived of MD4, but SHA at 160-bits is generally considered more secure than MD5 at 128-bits not only because of key length, but also because of algorithmic strength.

[return to top]

Supported Crypto Schemes

The Sonicwall SSL Offloaders supports three predefined Security Policies, and also allows you to define your own. The predefined policies, and their associated algorithm are:

Default
ARC4-MD5,ARC4-SHA,EXP-ARC4-MD5,EXP-ARC4-SHA,
EXP1024-ARC4-MD5,EXP1024-ARC2-CBC-MD5,
EXP1024-ARC4-SHA,NULL-MD5,NULL-SHA

Weak
EXP-ARC4-MD5,EXP-ARC4-SHA,EXP-ARC2-MD5,
EXP1024-ARC4-MD5,EXP1024-ARC2-CBC-MD5,
EXP1024-DES-CBC-SHA,EXP1024-ARC4-SHA,NULL-MD5,
NULL-SHA,EXP-DES-CBC-SHA

Strong
DES-CBC-MD5,DES-CBC-SHA,DES-CBC3-MD5,DES-CBC3-SHA,
ARC4-MD5,ARC4-SHA

All
DES-CBC-MD5,DES-CBC-SHA,DES-CBC3-MD5,DES-CBC3-SHA,
RC4-MD5,ARC4-SHA,EXP-ARC4-MD5,EXP-ARC4-SHA,
EXP-ARC2-MD5,EXP1024-ARC4-MD5,EXP1024-ARC2-CBC-MD5,
EXP1024-DES-CBC-SHA,EXP1024-ARC4-SHA,NULL-MD5,
NULL-SHA,EXP-DES-CBC-SHA

[return to top]

Certificates

What is a Certificate?

Public key certificates are digital stamps of approval for electronic security. There are 3 main characteristics of certificates:

1. Provide identification of the web site and the owner
2. Contain the public key that will be used to encrypt and decrypt messages between parties
3. Provide a digital signature from the trusted organization that issued the certificate and when the certificate expires

This information is critical because it determines whether the certificate can be trusted. Any system that knows the issuer of the certificate's public key can verify the signature and ensure the validity of the certificate.

[return to top]

What is a Certificate Authority?

The issuer of a certificate is traditionally known as a Certificate Authority (CA). The CA is the one who digitally signs a certificate and ensures its validity. There are two types of CA, private and public. Private CA issue certificates to be used in private networks where they can valid the certificate. Public CA issues certificates for servers that belong to the general public. A Public CA must meet certain requirements before they are added as a root authority to a browser. Because this is a controlled process, all public CA must be registered to issue certificates. The most popular public certificate authorities are VeriSign, RSA Data Securities, Thawte, Entrust, and Baltimore Technologies.

[return to top]

How Many Certificates Do I Need?

Each certificate contains information specific to a Common Name, or host name. An example would be: www.sonicwall.com. The web site for this Common Name would only have one certificate no matter how many servers were actually available to serve content. However, some Certificate Authorities have license restrictions that limit the use of a certificate to a single server. Please check with your Certificate Authority prior to using a single certificate for multiple commonly named servers.

[return to top]

How Many Certificates Can The SonicWALL Appliances Manage?

The SonicWALL SSL devices will store up to 255 certificates. This value could decrease if a certificate exceeds 5k bytes in size (the average certificate is 1k) or if multiple certificates are chained together.

[return to top]

How Do I Get A Certificate?

An issuer of a certificate is known as a certificate authority (CA). The form used to obtain a certificate from a CA is known as a certificate signing request (CSR) which generates a PKCS10-formatted certification. This request is located on the web site of all certificate authorities. Another way to generate a CSR is to use utilities like OpenSSL or from Microsoft IIS. This gives one the opportunity to generate a self-signed certificate, one which is not validated by a CA but is available for internal testing purposes, or can be forwarded to a CA to be signed.

[return to top]

What Certificate Formats Are Supported?

The standard format for public key certificates is known by it ITU specification number of X.509. However, the X.509 certificate can be stored in different file formats, depending on how one creates the certificate.

We accept three kinds of Certificate Signing Requests (CSR's). The first kind, and by far the most popular form, is the DER format. In technical terms, this is a BASE64 encoded DER PKCS#10 Certificate Signing Request. Most modern web server software will generate a CSR in this format: ApacheSSL, Stronghold, Netscape's newer servers, Microsoft IIS and Zeus all comply with this specification.

The second form of CSR we accept is based on the Privacy Enhanced Mail (PEM) specification. WebSite Professional 1.1x, 4D WebSTAR Server Suite/SSL, Lotus Domino 4 and some other older servers generate these CSR's.

The third form is specific to Microsoft IIS. This format is called NetIIS. It actually combines the certificate with the public key and the private key into a single file.

[return to top]

What Are The Current Certificate Export Laws?

The new US encryption export regulations took effect on 14 January 2000. In terms of the new regulations, CAs may now export certificates to any non-government entity and to any commercial government-owned entity (except those that produce munitions), in any country except Afghanistan (Taliban-controlled areas), Cuba, Iran, Iraq, Libya, North Korea, Serbia (except Kosovo), Sudan and Syria.

[return to top]

What Are SuperCerts, SGC and Step-Up Certificates?

Step-up or SuperCerts Certificates:
Certain certificate Authorities have obtained a special license from the US government to issue certificates that enable international versions of the browsers to do 128-bit encryption. This license allows them to issue Strong Encryption Certificates to enable strongly (128-bit) encrypted communications for international browsers. US "domestic" versions of all browsers should aways give you 128-bit security, but your server must support 128 bit, and you must have generated a 1024 bit key.

Browser Compatibility
SuperCerts are recognized by IE 5.01, Netscape Communicator 4.7 and later browsers. Older browsers will still create a secure SSL connection at 40, 56 or 128 bits depending on the precise browser version.

Microsoft Server Gated Cryptography (SGC):
Banks and their customers around the world now have a secure way to begin taking advantage of the convenience and efficiency of online banking. The U.S. government's recent decision to allow companies to export products that use strong, 128-bit encryption means that Microsoft and others can take online banking a big step forward. Microsoft is responding to the decision by delivering a full solution, which you can read more about at our Server Gated Cryptography (SGC) site.

Benefits
Here are some of the benefits that Microsoft's Server Gated Cryptography (SGC) solution provides: Banks and financial institutions can securely conduct financial transactions with their retail customers worldwide without requiring customers to change their standard Web browser or financial software. Microsoft's online banking solutions do not require any special client software. Customers can use standard, off-the-shelf, 40-bit exportable versions of Microsoft Internet Explorer to connect to an SGC server and conduct secure transactions with strong, 128-bit encryption. SGC is fully interoperable with Netscape browsers and servers. This means that Internet Explorer users will be able to use 128-bit encryption while communicating with Netscape servers.

[return to top]

Can I Be My Own Certificate Authority?

Yes. This is generally done for testing and evaluation purposes. The certificate is generated via a CSR but signed with your information. If a browser receives a self-signed certificate, it will post a message stating that you are accessing a non-trusted site. Another option is to obtain a license from a CA to generate your own certificates and then chain them with a valid certificate from the CA.

[return to top]

What Is Key Management?

With each certificate there is a private-key and a certificate association. When a cluster of servers deals with multiple certificate/key combinations, the management can become a nightmare. Traditionally the web servers had to deal with managing these combinations. The situation becomes compounded with each key added to the cluster. By using SSL offloading appliances, the key management issue becomes simple since it operates independent of the web server. One no longer needs to administrate multiple applications but can centralize all SSL key management to a single platform.

[return to top]

SSL or VPN

Do I need SSL, a VPN, or both?

SSL offers spontaneous security while VPN's require some pre-configuration to offer security. One is no more or less secure than the other. When deciding which is right for you, look at the environment you are trying to secure. The big question is really "*Am I in control of everyone who will be connecting*?" If the answer is "Yes" and you are prepared to commit to managing these secure users, then a VPN is well suited to your needs. A VPN also offers the ability to easily and securely connect remote subnets, or remote offices, together via the Internet. SSL does not currently offer a simple way of doing this.

If you are running an e-commerce site, a financial services site, or any other sort of publicly accessible site that could potentially be visited by millions of anonymous users, SSL is the proper security solutions.

If you have remote users, remote offices, or partners to whom you wish to provide secure access to your network, a VPN is the proper solution. Most network operations concerned with security employ both VPN and SSL solutions for their respective security offerings.

[return to top]

What is a PKI?

A Public Key Infrastructure (PKI) provides security services, including certificates, key management and policy management. These services are used by a variety of applications both in intranets and within the Internet. These applications include special PKI services that support digital signature, data encryption and other authentication services.

In general, PKI products and services allow organizations to establish security domains, and then issue certificates that vouch for the validity of public keys used within those domains. These functions are becoming increasingly important for a variety of network applications and services like virtual private networking (VPNs), secure socket layer (SSL), secure email and others.

In summary, a PKI consists of the following basic components:

- Certificates that bind public keys to the identities of the owners
- Application that uses public key cryptography in a client / server model
- Certificate Authorities that issue certificates
- The ability to create trust relationships from one CA to another CA
- Creation of policies that can govern the use of all of the components used in PKI

[return to top]

Supported Protocols

What protocols are supported?

SSL is a protocol independent security mechanism. Any TCP protocol can make use of SSL, and SonicWALL's SSL Offloaders support all of these protocols. Just like HTTP, which normally operates on TCP port 80, has an SSL counterpart on TCP port 443, most other well known TCP services have SSL counterparts. The following is a list of some of the better known TCP over SSL services:

Secure Service	Secure Port	Non-Secure Service	Non-Secure Port
SSMTP	465	SMTP	25
SNEWS	563	NNTP	119
SSL-LDAP	636	LDAP	389
SIMAP	993	IMAP	143
SPOP3	995	POP3	110
TELNETS	992	TELNET	23
HTTPS	443	HTTP	80

[return to top]

How do I configure [x] protocol on my SSL-x?

Configuring any TCP protocol over SSL on a SonicWALL offloaders is as easy as configuring HTTPS. Let's say you want to configure the SSL-x to Offload a Secure POP3 server. No reconfiguration would be necessary on your POP3 service (assume IP

address 10.8.8.8), it would continue to listen on port 110. On the SSL-x, the service definition would look like this:

```
ssl
server securepop3 create
  ip address 10.8.8.8 netmask 255.255.255.0
  sslport 995
  remoteport 110
  keyassoc default
  secpolicy strong
end
```

Substitute Key Associations and Security Policies as necessary.

[return to top]

Web Servers

What Web Servers Are Supported?

The SonicWALL SSL accelerators work with any web server application. Due to the nature of offloading, these appliances operate completely independent of the application. There are no special software hooks or recompilation required. This solution offers several advantages:

1. This makes SSL certificate & key management easier since they are no longer dependent on the different types of server.
2. Creates to ability for Service Providers to offer SSL as a core technology since SSL is removed from the server.
3. Eliminates the need to limit the certificate from working in only one application platform.

[return to top]

How Do I Get An Existing Certificate and Key From My Web Server?

If you are using:

Apache mod_SSL
The key and certificate locations are listed in the $APACHEROOT/ conf/httpd.conf file.The default key is $APACHEROOT/conf/ ssl.key/*.key .The default certificate is $APACHEROOT/conf/ ssl.crt/*.crt .Note the name and location of these elements.

ApacheSSL
The key and certificate locations are listed in the $APACHESSLROOT/conf/httpd.conf file. The default key is $APACHEROOT/certs/*.key .The default certificate is $APACHEROOT/certs/*.crt . Note the name and location of these elements.

Stronghold
The key and certificate locations are listed in the $STRONGHOLDROOT/conf/httpd.conf file. The default key is $STRONGHOLDROOT/ssl/private/*.key .The default certificate is $STRONGHOLDROOT/ssl/*.cert .Note the name and location of these elements.

Windows NT
The certificate file is in the directory specified when the certificate was downloaded from the CA.

1. Double-click the certificate file to open the viewer.
2. Click the Details tab.
3. Click Copy to file .The Certificate Manager Export Wizard opens.Click Next .
4. Select the DER-encoded binary X.509 radio button.Click Next .
5. Specify a file name and location.Click Next .
6. Click Finish .
7. Click OK when you see the successful completion notice.
8. Exit the Certificate Manager Export Wizard.
9. Close the certificate viewer.

The keys are located within the Key Ring —the key manager program.

1. Follow these instructions to export a key.
2. Click the Start button, point to Programs>Windows NT 4.0 Option Pack>Microsoft Internet Information Server, and click Internet Service Manager .The Microsoft Management Console opens.
3. Navigate to the web site using the object list.
4. Right-click the web site key ring object and click Properties in the shortcut menu.
5. Click the Directory Security tab.
6. Click Edit in the Secure Communication panel.
7. Click Key Manager.
8. Click the key to export.
9. On the Key menu, point to Export Key, and click Backup File.
10. Read the security warning and click OK.
11. Select a file location and enter a file name.
12. Click Save.
13. Exit the Internet Service Manager.

[return to top]

Device Configuration

How Many SonicWALL Offloaders Do I Need?

As with any resource allocation, one needs to plan for the worst-case scenario. For SSL traffic this represents the peak traffic levels plus what is forecasted in the future. Once the worst-case scenario is known, the other factor to take into account is redundancy. To

create a high available cluster, one must add in redundant services and network paths. This is true for SSL offloaders as well. Built into each SonicWALL appliance are the capabilities to fail-over to another unit. When multiple SSL appliances are deployed with a load balancer, the load balancer handles the fail over from one SSL appliance to another.

[return to top]

What Management Options Do The SonicWALL SSL Appliances Offer?

There are 3 modes of management:

1. Command line utility for remote administration. The utility runs on a machine that has network access to the server port of the SSL appliance. The utility does an UDP broadcast to show all devices available for configuration. The utility includes a quick start wizard for added ease of use and the commands can be scripted. Security is handled via passwords / access lists and encrypted communication.
2. Console port via null-modem serial cable. The management features available thru the console are limited. The commands available include network settings, IP settings, one-armed mode of operation, firmware updating, and resetting of configuration.
3. Browser based GUI configuration. From any browser on the secure side of the SSL appliance, managers can configure the SSL parameters. The GUI offers extensive online help and a quick start wizard. Includes password and encryption communication. (Available Q1 2001)

SNMP capabilities will be included in the next release (Q1 2001).

[return to top]

Do the SonicWALL SSL Appliances Need an IP Address?

Yes. An IP address is need for two reasons:

1. Configuration. The configuration utilities use an IP address for configuration purposes. Although the utilities can do an auto-discovery of SonicWALL devices, one can also attach directly by knowing the IP address.
2. Proxy deployment. When the SSL offloading appliance is deployed as a proxy, all traffic is redirected from the load balancer to the IP address of the appliance. All decrypted requests are then forwarded on to the servers with the IP address of the SSL appliance as the source address.

[return to top]

Where Do I Place the SonicWALL SSL Appliance Within My Network?

The easiest method of deploying an SSL accelerator, a method supported by diverse and heterogeneous groupings of equipment, is the In-Line method. The In-Line method places the SSL accelerator at some point on the network between the router and the content switch. This point can be before or after any preliminary switching or firewalling occurs on the network. While the Sonicwall SSL products support this method, it is but one of four deployment methods available to the SSL family. The full list of deployment methods:

- In-Line
- Transparent Sandwich
- Proxy-Mode
- Transparent-Mode

[return to top]

What Is In-Line Operation?

The In-Line configuration is the simplest method to deploy because it requires no specific inter-operation configuration on either the SSL appliance or any load balancing devices.

Because the SSL appliance front-ends the server cluster or a load balancer, passing all traffic in a decrypted fashion, no additional traffic-handling requirements beyond its normal configuration are necessary for the servers or load balancer.

[return to top]

What Is Proxy Operation?

Proxy-Mode is relatively simple to configure and scales well, but has one caveat, namely, it prevents client IP recording by real-server access logs. The reason this occurs is because of the path the traffic flows in a Proxy-Mode configuration and because of the unique role the SSL appliance plays in this design.

Unlike the other methods of deployment, when used in Proxy-Mode the SSL appliance does not act transparently. Instead, it acts—as the name implies—as a proxy between the requesting client and the back-end real-server. Because of this behavior, the real-server sees the access as coming from the IP address of the SSL appliance, rather than from the originating client. While this effect is acceptable in many installations, there will be instances where client IP accounting accuracy will be required. In such cases, an alternate design method is encouraged.

[return to top]

What Is One-Armed Transparent Operation?

Offering a scalable, single content switch solution without the IP accounting limitations of the Proxy-Mode, Transparent-Mode is both the most complex, and the most feature-rich of all the configurations. Transparent-mode's intricate packet flow requires a certain set of capabilities from a content switch. Specifically, the content switch must offer two features: flow switching and cache-device ACL support.

For more information, see the document "SonicWALL SSL Deployment with Content Switches".

[return to top]

How Do I Upgrade the Firmware?

There are two main things that must happen when updating the firmware, upload the flash image and install the new configuration manager, in that order. Remember to save the SSL configuration to a file before updating the flash because it will be erased.

Load the new flash image using the following directions.

1. Start the orginal release configuration manager and attach to the device.
2. At the inxcfg> prompt enter "copy file flash"
3. Enter the path to and the file name of the new flash file (Example NT: d:\fw\sslrack_upgrade.phr, Example Red Hat Linux: /mnt/cdrom/fw/sslrack_upgrade.phr). Note: when using Linux make sure the CD is mounted first.
4. Enter "yes" at the warning prompt.
5. Press enter and allow five minutes to ensure that the flash has been successfully loaded.
6. Type "reload" to reboot the device.
7. Type "quit" to exit the configuration manager.

Load the new configuration manager.

Windows NT

1. Run the setup.exe found in the \mswin\NT4 folder on the CD by double clicking the icon.
2. Follow the install shield instructions on the screen choosing the repair option.

Red Hat Linux

1. Run the install script found in the \Linux\i386 directory on the CD using instructions found in the manual.

[return to top]

What High-Availability Options Are There?

To maximize uptime, a high-availability option if offered. High-availability in an In-Line configuration is unique in that it is offered by the SSL appliance. The SSL appliance offers a hot-standby mode wherein two SSL devices—one active, and one standby—are connected via a serial cable for the sake of heartbeat communications. Should the active SSL appliance fail, the standby unit will identify the failure, and will assume the role of active SSL appliance. When the failed unit resumes operation, it will do so in a standby mode.

High-availability is offered with load balancers by utilizing its health-check mechanism; if one of the SSL devices, one of the network ports, or one of the real-server farms fails, the load balancer should recognize its unavailability and remove it from its eligible redirection list.

[return to top]

How Do I Load Certificates Onto the SonicWALL SSL Appliance?

The configuration manager is used for importing certificates and keys into the SSL appliance. For each server IP filter you create a key association. This key association can be used for many filters or just one.

This example is for the filter named 'myDevice'

```
(config-ssl[myDevice])> keyassoc myKey create
(config-ssl-keyassoc[myKey])> pem certFile keyFile
(config-ssl-keyassoc[myKey])> end
(config-ssl[myDevice])>
```

Note: Use the **der** command when using DER-encoded keys and certificates, the **netiis** command when using keys and certificates from IIS, or the **cert** and **key** commands for combinations of encoding formats.

[return to top]

Troubleshooting/Error Messages

I can't establish a connection via the console port.

The console port on the SSL-R or SSL-IA are set to auto-detect the settings of your terminal emulation software. The physical connection requires a 9 pin F-F Null Modem Cable. We recommend setting your terminal emulation software as follows:

- Baud Rate: 115,200
- Data bits: 8
- Parity: None
- Stop bits: 1
- Flow Control: None

Hit [Enter] a few times after making initiating the terminal session, and you should be prompted with a menu. If connection problems persists, verify your communication port settings, and try a different cable.

[return to top]

The Configuration Manager Doesn't See Any Devices.

Upon launching the Configuration Manager software, verifying network connectivity to the device, and issuing the "discover" command, you may receive the response:

> Searching for Phobos devices...
> no new devices found
> inxcfg>

If you receive this message, try issuing the command "show devices". You might now receive something similar to:

```
Key:        XYZ  X=Remote,Local Y=[un]Attached,down,Denied Z=config,readonly
-------------------------------------------------------------------------------
ipXpress    Ru   IN-10-1-7-133 HW 00:60:f5:07:08:04 IP 10.1.7.133
inxcfg>
```

You could attach to the device by issuing the command "attach IN-10-1-7-133". If, however, no devices were listed, check your physical connection to the unit, or try connecting via a cross-over cable directly to the unit. Should that fail, the device might have an IP address assigned to it that places it on a different subnet from your workstation. In this case, the quickest solution is to reset the IP address of the unit via a console connection.

[return to top]

The Configuration Manager sees the device, but I can't connect to it.

The Configuration Manager will maintain a list of previously seen devices, even after they have been disconnected or powered off. If the device shows up in the "show devices" list, but does not allow you to attach, try to ping the unit by the IP addresses listed under the "show devices" output. If you can ping the unit but still cannot attach, close and re-launch the Configuration Manager software and try the connection again. If the unit does not reply to a ping, see the previous category "The Configuration Manager Doesn't See Any Devices." Finally, the configuration manager software utilizes TCP and UDP ports 2932. Make certain that there is nothing that is blocking access to these ports between your workstation and the SSL device.

[return to top]

After Initially Setting the SSL-x's IP Address I Get a Session Timeout and Disconnect.

This is a known issue, and has been resolved in the soon to be released software version 2.0. In the meantime, attach to the unit, and set the IP address. Upon being disconnected, close the Configuration Manager program and then re-launch it. Issue a "show devices". You should now see the device listed by the IP address you set (e.g. IN-10-1-1-1) rather than by MAC address. Issue an "attach IN-10-1-1-1" to connect to the unit, and the session should proceed normally.

[return to top]

I enter a default route on the SSL-x, but it doesn't appear in the configuration.

This occurs if, at any point during the configuration of the unit, you changed the subnet on which it resides. For example, you performed an initial wizard configuration and assigned the unit an IP address of 10.1.1.1 netmask 255.255.0.0 and a default gateway of 10.1.0.1. Later, you changed the IP address to 10.10.1.1 netmask 255.255.0.0. Such a configuration change might cause the "ip route 0.0.0.0 0.0.0.0 10.1.0.1 metric 1" command to disappear from your configuration, and prevent you from entering a new one.

The quickest way to resolve this issue is to change the default gateway of the unit from a console connection.

[return to top]

"Failed to send set ssl server command! Type "show messages" for more information." and "Failed to delete ssl server id [x] from device."

You might receive these error messages if you make changes within the “ssl” configuration section while the unit is in active operation. If a defined SSL service on the SSL Offloader has active connection, and you attempt to change any of its attributes (i.e. IP address, sslport, remoteport, keyassoc, or secpolicy), the change will fail, and you will receive one of the above messages. This is done as a security measure to prevent interruption of active service. To make a change, first make certain there are no active connection through an SSL service definition, and then enter your new values.

[return to top]