

The Security Building Blocks of TLS

Technical White Paper

By Maurice McMullin,
Product Manager

Introduction

System and network administrators regularly have to deploy and configure the TLS (Transport Layer Security) protocol as a method of securing communications on devices such as load balancers and web servers. As they deploy TLS on their application delivery platforms they may find themselves presented with configuration options for the cryptography to be used. This paper takes a look the underlying cryptographic building blocks and security tools and describes how they are used by TLS.

The TLS protocol can use a range of algorithms for authentication, encryption and key exchange known as a cipher suite. During the establishment of a TLS session, the parties will negotiate which of the TLS defined cipher suites to use.

TLS Requirement	Implementation
Securely establishing a secret key between two communicating parties	<p>RSA and Diffe-Hellman (DH) algorithms are the most common method of securely establishing the once off secret for the TLS session.</p> <p>TLS supports the use of a pre-shared key where no key establishment is required.</p>
Authenticating and trusting the other party	Digital certificates are used to establish trust. Certificates can be provided by clients and servers although in most implementations, only the server provides a certificate as the client will be anonymous or authenticate using an alternative method such as a password.
Protecting the confidentiality of data in transit	<p>Encryption algorithms such as AES make sure that data is not visible to third parties. These algorithms are generally known as 'Symmetric' ciphers as the same key is used by both parties for encryption and decryption.</p> <p>This is the key that is established securely between the parties using RSA, DH or pre-shared protocols.</p>

Cipher suites in TLS are named in the format TLS_DHE_RSA_WITH_AES_256_CBC_SHA which includes references to the key exchange mechanism (DHE), the authentication (RSA), the encryption algorithm and options (AES_256_CBC) along with the hashing function (SHA).

About the TLS Protocol

TLS (Transport Layer Security) is a security protocol that is used extensively on the Internet to authenticate and secure communications. TLS is an evolution from SSL and is an IETF (Internet Engineering Task Force) standard. SSL was originally developed by Netscape and has gone through multiple versions and a name change to TLS in 1999. The current TLS version is 1.2 with a draft 1.3 version currently in development. Given the nature and value of communications that we have on the internet, there is an expectation that our communications are private and that other parties in the communications are who they say they are. It is these two basic requirements that TLS addresses by providing a way for parties in a conversation to verify each other and to communicate privately. The most common TLS application is securing client connections to websites where the protocol can verify the identity of the website and secure the communications.

To deliver the goal of authenticated and private communication, the TLS protocol uses a range of security fundamentals. How these fundamental building blocks work is well known and they are continuously under examination by security researchers and other entities to identify weaknesses. Where weaknesses are identified, the security community will issue a fix or a recommendation to deprecate the use. This openness and continuous assessment has identified many vulnerabilities ranging from weaknesses in TLS implementation (e.g. POODLE) through to key size weaknesses due to brute force attacks.

The Security Fundamentals

The RSA Algorithm

The RSA algorithm may be used within the TLS protocol for authentication and key exchange. The RSA asymmetric encryption cipher uses two keys that are mathematically related so that content encrypted by one key can only be decrypted by the other. What this means in practice is that one of the keys can be made public and used by anybody to encrypt messages to the owner of the other key which is kept private. This is known as Public Key or Asymmetric cryptography.

This private key (kept secret) and public key (available to everyone) allow two parties to securely communicate as shown in the following communication between Bob and Alice.

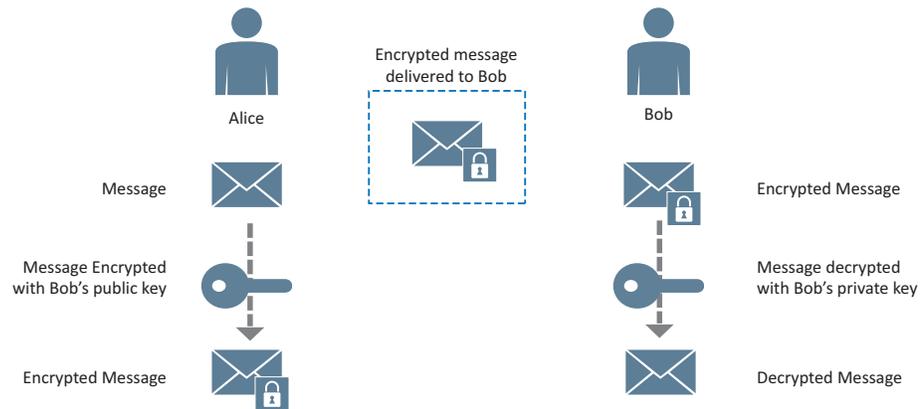


Figure 1 - Message encryption using RSA

Bob's public key is available to everybody so Alice uses this key to encrypt a message and send it to Bob. Bob uses his private key to decrypt the message sent by Alice.

While RSA may be used for both authentication and secure key establishment, the general recommendation is to use Diffie-Hellman for secure key establishment. The reasoning for this is that if a 3rd party gains access to the RSA key, they can decrypt any previously recorded sessions as the RSA key is always the same. The Diffie-Hellman protocol can be used to create different keys (known as ephemeral keys) which are unique to each session. This prevention of playback by using ephemeral keys is known as 'forward secrecy'.

The RSA protocol can also be used for digital signing when a known message encrypted by a private key can be decrypted by a public key. A successful decryption by the public key shows that the private key is held by the encrypting (signing) entity. See the section on Hashing Functions for an example of digital signing.

The private key should be stored securely as a compromised key would allow third parties to masquerade as the owner of the key. See the section on key security to understand how private keys may be protected.

The RSA algorithm was developed by Ron Rivest, Adi Shamir and Leonard Adleman who went on to form RSA Security to commercially exploit the algorithm. The algorithm was patented in the US until September 2000 while it was never patented globally.

Diffie-Hellman Key Exchange

The Diffie-Hellman (DH) key exchange algorithm allows two parties, who do not know each other, to securely exchange a key even if a third party is observing all their communications. The magic of DH is that it does not actually encrypt any data yet it allows each party to create the same key based on the exchange of some unencrypted data.

Subpage heading

Rather than delve into the mathematics, the example below uses color to explain in the abstract how DH key exchange works. The assumption is that it is difficult to unmix the colors that are transmitted by removing the common color agreed in the first step.

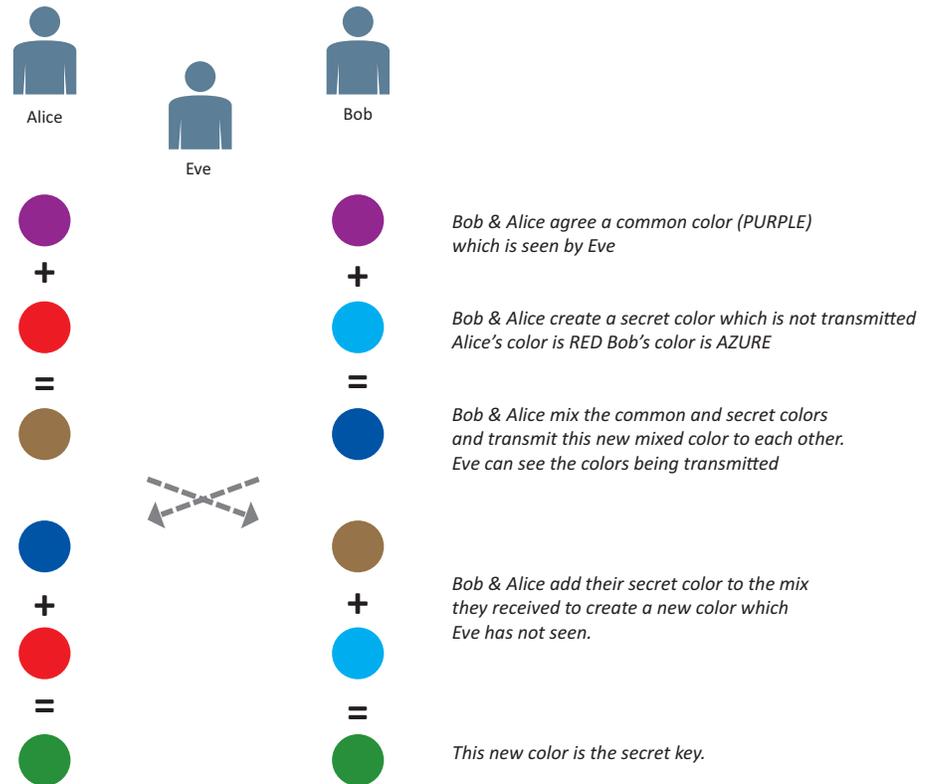


Figure 2 - Diffe-Hellman key exchange

Alice gets to the secret color by adding her private color (RED) to Bob’s mix (PURPLE + AZURE).

Bob gets to the secret color by adding his private color (AZURE) to Alice’s mix (PURPLE + RED).

In each case the secret is the product of PURPLE + AZURE + RED.

Even though Eve observed (and possibly recorded) all the communications, she cannot deduce the secret.

Within TLS, the DH algorithm is normally used to create ephemeral (one-time) keys while RSA is used for authentication of the parties as in the **TLS_DHE_RSA_WITH_AES_256_CBC_SHA** cipher suite.

Hashing Functions

A hashing function is a one-way transform where a piece of data (known as the message) when passed through the function will produce an output (known as a hash or digest) of a given size. The one-way nature of this mathematical trapdoor

means that it is not possible to derive the original message from the digest. As can be seen from the following table for a 128bit MD5 hash function, even the simple addition of a comma to the message produces a vastly different digest

Message	MD5 – 128bit Digest
Lorem ipsum dolor sit amet	fea80f2db003d4ebc4536023814aa885
Lorem ipsum dolor sit amet,	059e5714948c77a56a5376310e9dc0c9

Hashing functions have many applications including creating file checksums, password storage, indexing and fingerprinting. In cryptography, hashing functions are used to ensure the integrity of communications and in digital signing.

With a digital signature, the sending party will create a hash of the content being signed and encrypt this with their private (RSA) key. The recipient of the message can verify that the message came from the person who possessed the private key by comparing the hash received (decrypted by the public key) with the locally calculated hash. The example below shows how Alice sends a signed message to Bob who validates the identity of the sender.

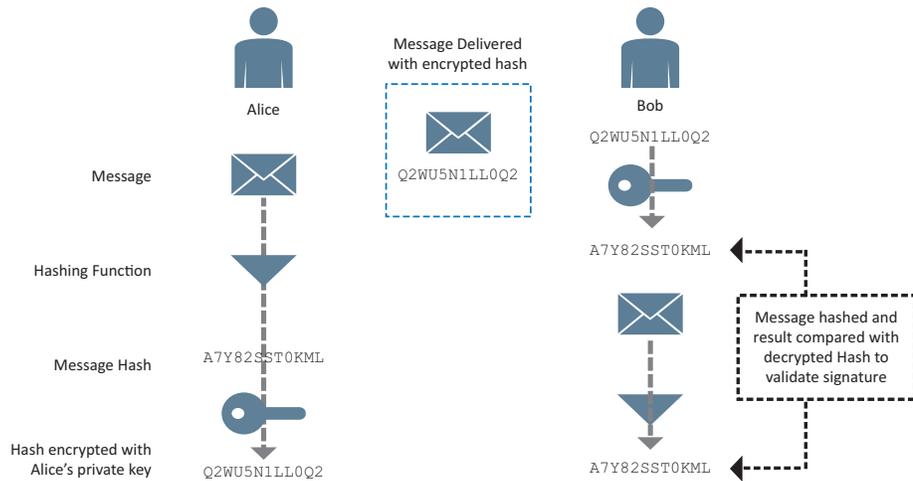


Figure 3 - Signing a message using RSA

Alice passes the message through the hashing function and encrypts it with her private key. The message (unencrypted in this example) and the hash are delivered to Bob who calculates the hash from the message and decrypts the hash provided by Alice using her public key. If the two hashes match then Bob can be assured that:

1. Alice created the message - only her private key could have encrypted the hash so that her public key could decrypt it
2. The message was not altered in transit – Because the decrypted hash matches, the content has not changed since it was hashed by Alice

This is how hashes and public key algorithms can be used for authentication and to provide protection against tampering when in transit.

Some of the earlier implementations of hashing functions such (SHA-1) have had weaknesses identified and are no longer considered safe for use.

Hashing is also used to provide security for storing passwords where rather storing passwords in cleartext, they are hashed before they are stored. Passwords can be validated by hashing the password provided and comparing with the stored hash. Advances in cryptanalysis and computing power make large files of hashed passwords susceptible to cracking by brute force or dictionary attacks. To prevent this type of attack (e.g. where a hacker has stolen a file of hashed passwords) additional random cleartext is added to the password (known as a 'salt').

Symmetric Ciphers

While the RSA algorithm addresses the challenge of establishing a secure communications channel, it is a computationally intensive and is not suitable for securing large volumes of data. Symmetric ciphers on the other hand are computationally efficient but use a single key (a shared secret) which must be agreed by the parties before secure communications can begin. In common with many other secure communication protocols, TLS will use RSA (or DH) to establish the initial secure communications channel over which the parties can then safely share the symmetric cipher key.

With symmetric ciphers, the same key is used for encryption and decryption and can encrypt data one byte at a time (a Stream Cipher) or in blocks (Block Cipher).

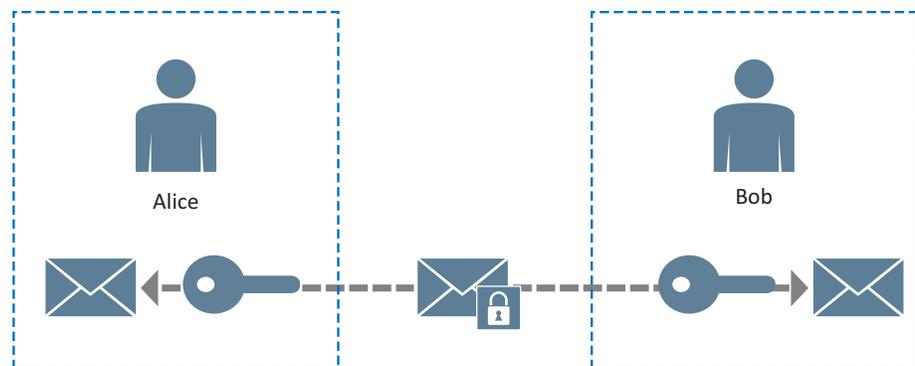


Figure 4 - Symmetric encryption using a shared key

While stream ciphers are generally more efficient than block ciphers, block ciphers can add additional security by implementing Cipher Block Chaining (CBC) where a block of data to be encrypted is altered based on the previous block normally through an operation such as an XOR.

Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) is a public key cryptosystem that provides similar levels of security as RSA but at much smaller key sizes and reduced computational overhead. Like RSA, it is a mathematical trapdoor where getting from A to B is easy, while getting from B to A is hard. Cryptography experts see ECC as being the best long-term public key algorithm as research and computational advances continue to identify weaknesses and brute force exploits in the RSA and DH algorithms.

Digital Certificates

Digital certificates are used by TLS to provide a method of validating the authenticity of a public key being provided by a server or client. As with certificates in the real world, they are of no value unless the certification authority that issues them is trusted. Most web browsers are pre-populated with a range of trusted certification authorities which means that the browser can transparently trust any certificate issued by these authorities.

When a user visits `www.example.com` from their browser over a TLS encrypted connection, the certificate provided by `www.example.com` will contain a blob of information that includes the URL of the website and the public key of that site. The browser will check to see if this blob was signed by a trusted certification authority (CA) and warn the user if there is a problem.

The X.509 notation used to define digital certificates allows for a wide range of content and attributes within a certificate. These would normally include validity dates and valid certificate uses (e.g. Mail, TLS).

The following is a simplified view of how a digital certificate is issued by a CA.

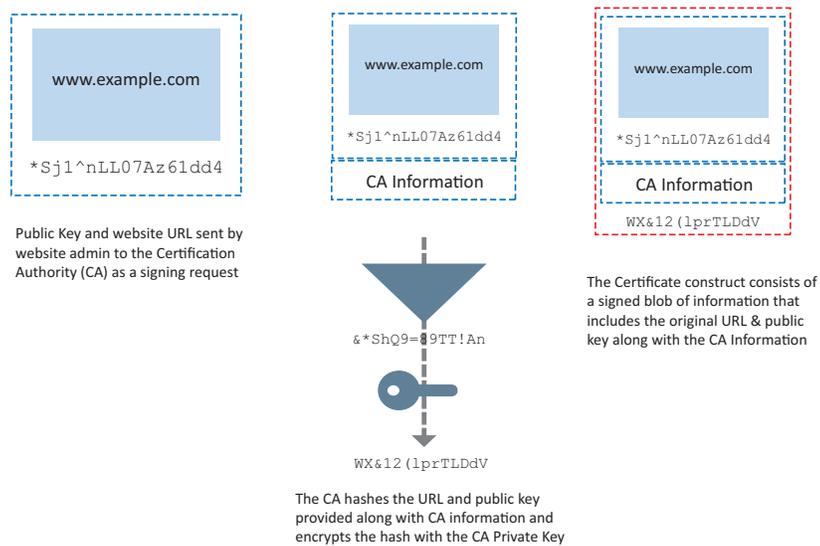


Figure 5 - Certificate signing process

Any cert issued by that CA can be validated by verifying the signature against the trusted CA signature. To an end-user this trust hierarchy is presented as a certificate chain with the self-signed CA certificate at the top of the chain. The CA may issue certificates to intermediate CA's which in turn issue certificates.



Figure 6 - Example of a Certificate Hierarchy in a browser

For example, the site mail.google.com uses a digital certificate that was issued by and intermediate certificate authority called 'Google Internet Authority G2' which in turn was issued by the 'GeoTrust Global CA' certificate authority.

This hierarchy and trust relationship is called a Public Key Infrastructure (PKI). A PKI includes mechanisms for revocation of certificates (OCSP and CRLs) and processes for requesting the issuance of certificates.

By trusting the top-level GeoTrust CA, all the intermediate CAs and certificates issued by those CAs are trusted.

Random Number Generators

Random number generation (RNG) is used by many security protocols and weaknesses in RNG can lead to exploits against the protocol using the random numbers. Random numbers are used by symmetric ciphers to create encryption keys while public key and key exchange algorithms need to generate random prime numbers. Randomness can be generated from naturally random sources such as background cosmic radiation or by a pseudo-random software algorithm.

Establishing Secure Communications

The TLS Handshake

The TLS protocol defines how the client and server interact to establish a secure authenticated communications channel. Among the tasks to complete during this session establishments are agreeing on the cipher suite to use. TLS includes a range of predefined cipher suites which define the key exchange and encryption protocols to use.

Subpage heading

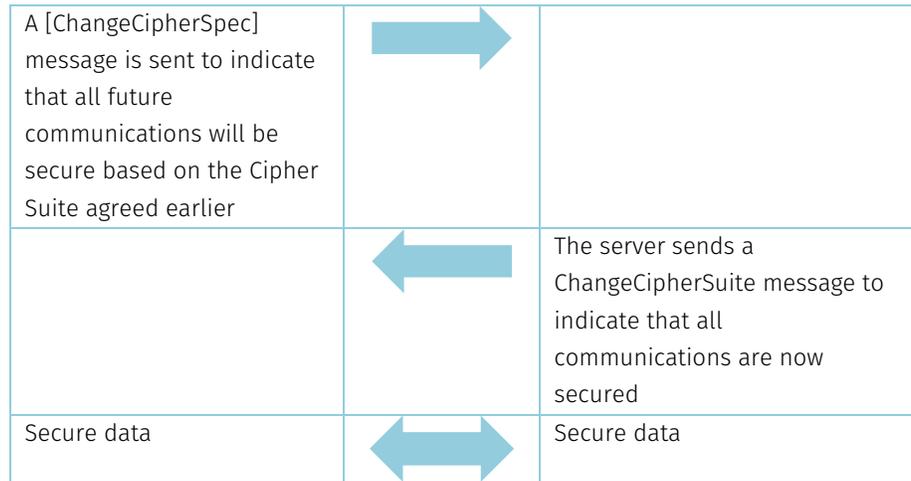
The cipher suite **TLS_RSA_WITH_AES_256_CBC_SHA** indicates that that the suite uses RSA for key exchange and authentication while a 265bit AES algorithm in CBC mode is used as the symmetric encryption protocol with SHA as the hashing algorithm.

TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 indicates the use of Ephemeral Diffie-Hellman for key exchange, RSA for authentication (Certificates) and 256bit AES in CBC mode with hashing performed by 256Bit SHA.

TLS also offers NULL encryption cipher suites (e.g. in **TLS_RSA_WITH_NULL_MD5**).

The following table provides an overview of how a client and server establish a secure communications channel with a TLS handshake.

Client		Server
<p>ClientHello Message</p> <p>This message tells the server which TLS Cipher Suites are supported by the client and provides a client generated random value. The ClientHello message also indicates the TLS versions supported by the client.</p>		
		<p>ServerHello Message</p> <p>Contains a server generated random value and the Cipher Suite and compression method to be used during the communication.</p> <p>The server will also deliver its certificate and key material to the client and optionally request a certificate from the client.</p>
<p>Client Reply</p> <p>Contains the client key material and, if requested, client certificate details along with a certificate verification message</p>		



Protecting Private Keys

Possession of a private key enables assertion of identity and potentially access to encrypted content and as such should be protected from access by non-authorized entities. The protection of private key material needs to address how the key is stored, how it is used and how it is managed. Key management for client devices will normally be implemented via a smartcard where the private key is stored on a cryptographically secured chip. For most internet based services, client authentication is performed via username and password so there are no client keys to manage.

Key management for internet accessible services provides a much greater challenge as a service may be spread over multiple servers at multiple physical locations and each server needs to access the private key in order to assert the identity of the service.

Storing the private key in a file on a server (or shared storage) raises the issue of how the key material is protected. One approach is to encrypt the file but this would prevent services from auto-starting as they would need the decryption password to access the keys. While having the file unencrypted would potentially expose key material, many administrators would take a balanced view on the cost Vs risks and rely on file system permissions and good security practices for protection.

For some, such as the financial sector, the value of content protected by private keys is such that they need (and often mandated by regulation) to apply the most rigorous security protocols and procedures. This is where the storage of key material in specialist security hardware comes into play. The dedicated hardware can be implemented as a plug in module (PCI or USB) or as a network attached service. These Hardware security modules (HSM) store the private keys and any cryptographic operation that requires access to the key is performed on the module. This means that the private key material never leaves the security environment of

the HSM and so removes the challenge of protecting key material stored on servers. A HSM may also provide enhanced protection including detection of physical attacks and authentication. The level of protection offered by a HSM is often measured against the FIPS-140 standard which outlines the levels of protection as follows

FIPS-140 Level	Protection offered
Level 1	Software only with FIPS approved algorithms
Level 2	The HSM must provide evidence of physical tampering and access must be controlled based on the role of the accessing entity.
Level 3	The HSM must offer tamper resistance and authentication based on identity. This level also provides separation between how key material is used and how it is managed. This allows for separation of security and operational access.
Level 4	At this level the physical protections include active deletion of key material if tampering is detected.

As well as protecting the key material, the HSM will also perform cryptographic operations that use the keys which offloads the RSA key operation during the TLS handshake leading to improved performance. Network attached HSMs provide the most flexibility as they provide centralized key protection that can be access by a range of hosts including load balancers, virtualized machines and cloud services.

Configuring TLS

Most TLS implementations will allow administrative control over which cipher suites to use so that weak or vulnerable cipher suites can be disabled. In an ideal world environment, the recommendations are to disable the use of weak ciphers such as DES and RC4 and to avoid the use of backward compatibility protocols such as SSL 2.0. Older browser and OS combinations such as Explorer 6 on Windows XP do not offer support for the newer and more secure cipher suites and an alternative browser such as Chrome or Firefox should be considered. There is no one 'best source' as to which cipher suites to use so regular trawls of multiple reputable resources is recommended to ensure the security of deployments.

The method of configuring TLS cipher suites varies depending on the platform in use. On the apache web server, the cipher suites to use and their precedence are defined in the Apache configuration files. On Windows server platforms, the cipher suites are defined in the registry with some third party tools available that perform configuration via a GUI.

The KEMP LoadMaster platform allows configuration of cipher suites from the administration interface and also some quick click options to control the use of weak ciphers such as RC4.

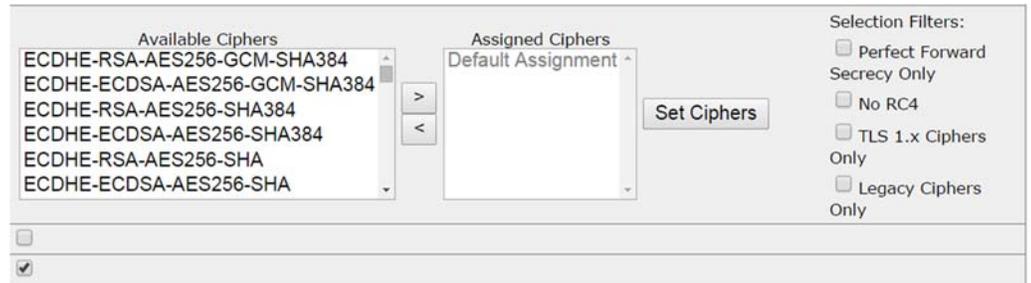


Figure 7 - Cipher Suite Selection with Kemp Loadmaster

The centralized administration of cipher suites on a load balancer such as the KEMP LoadMaster can greatly simplify the administration of TLS across multiple servers and provide a single point of enforcement and audit of TLS policies.