An Oracle White Paper
September 2010

High Performance Security for SOA and XML Web Services Using Oracle Web Services Manager and Oracle SPARC Enterprise T-series Servers

Introduction

This document details the high performance security strategies for Oracle Web Services Manager secured SOA and XML Web services by adopting the on-chip cryptographic acceleration capabilities of Sun SPARC Enterprise T-Series Servers. Using cryptographic solutions for application security are computationally intensive, which adds significant CPU overheads and severely impacts on the latency and throughput of the Web service transactions. This document presents the business value, technical pre-requisites, configuration, deployment, and verification guidelines for integrating Oracle Web Services Manager with Oracle Solaris Cryptographic Framework on Oracle SPARC Enterprise T-Series servers for delivering wire-speed acceleration of transport-layer and message-layer security scenarios.

Customer Business Values

Oracle Web Services Manager running on Oracle SPARC Enterprise T-Series Servers proven to delver high performance security and helps the customer with the following business values:

- Zero cost to deliver hardware assisted security acceleration
- Zero cost to installation, future-proof configuration and no maintenance required
- High-performance security and consistent scalability irrespective of cryptographic algorithm overheads.
- Out-of-Box virtualization environment support
- Lower cost of Ownership

The high performance is achieved through 16 on-chip cryptographic accelerator units available on Sun UltraSPARC T3 series servers that offloads and accelerates most cryptographic operations intended for application security. Oracle Solaris facilitates support for enabling hardware-assisted acceleration and also extends the support to Oracle VM based virtualized environments as well. In addition, offloading of cryptographic operations to the accelerator frees up CPU resources to other co-existing applications leading to server consolidation, and further helps with lowering total cost of ownership (TCO).

Target Audience and Assumed Knowledge

This document is intended for Oracle Web Services Manager security administrators and Solaris administrators who have been tasked to deploy and integrate the on-chip cryptographic capabilities of Oracle SPARC Enterprise T-Series servers with Oracle Web Services Manager and Oracle WebLogic Server. The administrators should be familiar with the installation of Oracle SPARC Enterprise T-Series servers, Oracle Solaris 10, Oracle Fusion Middleware 11gR2, Oracle WebLogic 11g suites and applied techniques for enabling SSL and WS-Security in Oracle Fusion Middleware applications.

Hardware and Software Environments

The Oracle Web Services Manager and Oracle WebLogic server security scenarios using cryptographic acceleration has been tested and verified to run on the following Oracle hardware and software environments (Table 1):

OPERATING SYSTEM	HARDWARE ENVIRONMENT	ORACLE FUSION MIDDLEWARE VERSION
Oracle Solaris 10 Update 8 and Solaris 10 Update 9	Oracle SPARC Enterprise T3 and T2 Plus servers	Oracle Fusion Middleware SOA Suite 11gR1 (11.1.1.4.0) * Oracle WebLogic 11g Suite 10.3.3

Table 1: Supported Hardware And Software Environment

* Testing and verification was performed with pre-release software of Oracle Fusion Middleware 11.1.1.4.0, which is planned for release in CY2010.

Oracle Web Services Manager - Overview

Oracle Web Services Manager (WSM) is Oracle's solution for addressing web services and SOA infrastructure security needs such as enforcing authentication, authorization, integrity, and confidentiality. It is a standards-compliant solution delivered as part of Oracle Fusion Middleware – Oracle SOA Suite and Oracle Access Management Suite.

Oracle WSM allows companies to:

1. Centrally define and store declarative policies applied to the multiple web services making up a SOA infrastructure.
2. Locally enforce security and management policies through configurable agents.
3. Monitor runtime security events such as failed authentication or authorization.

Oracle WSM is a best-in-class web services security and management system that can be used by both developers at design time, and systems or security administrators in production environments. To learn more about Oracle WSM capabilities, you can read the whitepaper "Securing Web Services and Service-Oriented Architectures with Oracle Web Services Manager 11g" available on Oracle Technology Network (OTN) at:
http://www.oracle.com/technetwork/middleware/webservices-manager/index.html

Role and Relevance of Oracle SPARC Enterprise T-series Servers

As security has taken unprecedented importance in all facets of the IT industry, today organizations are proactively adopting to cryptographic mechanisms to protect their business information from unauthorized access and ensure its confidentiality and integrity during transit and storage. Cryptographic operations are heavily compute-intensive which burdens the host system with additional CPU cycles and network bandwidth resulting significant degradation of overall throughput of the system and its hosted applications. For example, a host server capable of processing 1000 transactions per second can perform only 10 transactions per sec after deploying SSL for securing the hosted application. To speed up cryptographic performance, security experts often recommend and use cryptographic accelerator appliances to offload cryptographic operations and save CPU cycles for enhancing the system throughput and its hosted applications. While useful, adopting a specialized appliance for offloading cryptographic operations introduces a new set of complexities and issues in terms of additional installation, configuration and testing procedures that significantly increases the power demands and costs of deployment projects.

Foreseeing the need for special-purpose hardware that can outpace workload demands, Oracle introduced the industry's fastest on-chip hardware cryptographic capabilities into its family of Oracle SPARC Enterprise T-series servers with CoolThreads™ technology. The following figure shows the single socket Oracle SPARC Enterprise T3-1 server (Figure 1).

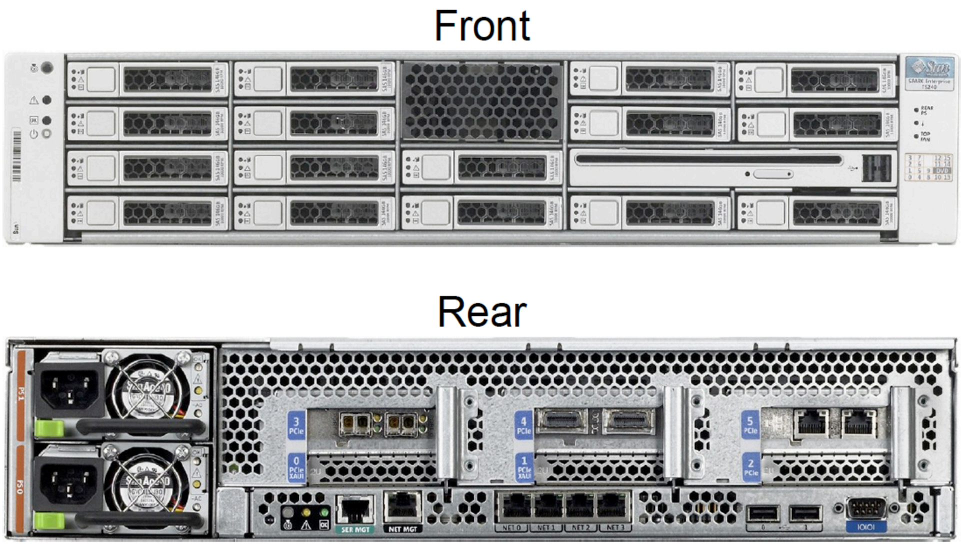


Figure 1: The Oracle SPARC Enterprise T3-1 Server

The Oracle SPARC Enterprise T-series servers are equipped with one of the UltraSPARC® T1, T2, T2 Plus and T3 processors. Each T-series has multiple cores, which resides on the same chip base. Each core has multiple threads where the processor is able to switch threads on every clock cycle in a round robin ordered fashion, and skip threads that are stalled (e.g. those threads waiting for a memory access). As a result, the T-series processor combines multiprocessing at the processor core level and hardware multithreading inside of each core with an efficient instruction pipeline to enable what Oracle calls Chip Level Multi-Threading (CMT). These processors present a unique "System-on-a-Chip" design principle that incorporates specialized features such as on-chip cryptographic acceleration, on-chip 10 Gigabit Ethernet networking and hardware-enabled virtualization capabilities. The following table compares the number of cores, threads and its on-chip cryptographic accelerator units of the T-series processor family (Table 2).

Processor Feature	**UltraSPARC T1**	**UltraSPARC T2/ T2 Plus**	**UltraSPARC T3**
No. of Cores	8	8	16
No. of Threads/Core	4	8	8
No. of Cryptographic Accelerator Units/ Processor	8	8	16

Table 2: Oracle SPARC Enterprise T-Series Processors - Comparison

Integrated Cryptographic Acceleration

Each core of the UltraSPARC T-Series processor includes a Modular Arithmetic Unit (MAU) that acts as a built-in hardware cryptographic accelerator unit that facilitates running cryptographic operations. This means a compute-intensive cryptographic algorithm operation can be off-loaded to the MAU. For example, the MAU is capable of performing 30,000 RSA-1024 operations per second with an UltraSPARC T2 plus processor. As RSA is the core component of an SSL communication, delegating compute-intensive RSA operations to the MAU speeds up a typical web application's SSL/TLS performance and in turn frees up the CPU to support other application-specific computations.

To facilitate delegation of cryptographic operations to the MAU of the UltraSPARC T1, T2, and T3 processors can, this process is carried out and managed by the Oracle Solaris OS using the Solaris Cryptographic Framework (SCF). With the built-in hardware cryptographic accelerator units on the chip these UltraSPARC T-series processors performs the delegated cryptographic operations in parallel with CPU speed and technically eliminates the need for additional special purpose cryptographic accelerators such as PCIe cards or network appliances.

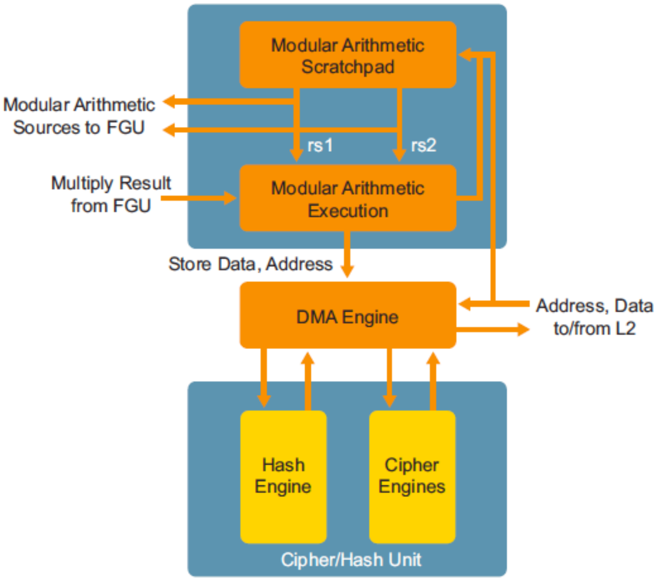


Figure 2: Oracle SPARC Enterprise T-Series processor - Modular Arithmetic Unit

The MAU on each T-series processor includes a dedicated Cipher and Hash engine that facilitates performing the cryptographic operations in parallel with the CPU speed (Figure 2). This is accomplished using dedicated cryptographic accelerator drivers, called the Niagara Crypto Provider (NCP), Niagara 2 Crypto Provider (N2CP) and Niagara 2 Random Number Generator (N2RNG). In practice, the NCP and N2CP accelerators uses the Oracle Solaris Cryptographic Framework to allow user-level applications to off-load cryptographic operations and take advantage of NCP and N2CP based on-chip cryptographic acceleration.

As a result, deploying Oracle SPARC Enterprise T-Series servers for application security eases the enabling of cryptographic acceleration mechanisms by providing an application-transparent facility to access the hardware accelerator, in most cases without adding any new code to the application at all. The following table shows the cryptographic algorithms supported by the Oracle SPARC Enterprise T-series processors (Table 3).

On-chip Accelerator Support	**UltraSPARC T1**	**UltraSPARC T2/ T2 Plus**	**UltraSPARC T3**
Accelerator driver	NCP	NCP, N2CP, N2RNG	NCP, N2CP, N2RNG
Public Key Encryption	RSA, DSA	RSA,DSA,ECC	RSA,DSA,ECC
Symmetric key / Bulk Encryption	-	AES, DES, 3DES, RC4	AES, DES, 3DES, RC4, Kasumi
Message Digests	-	MD5, SHA-1, SHA-256	MD5, SHA-1, SHA-256, SHA-512 and HMAC
APIs	PKCS#11 (via SCF)	PKCS#11 (via SCF)	PKCS#11 (via SCF)
Random Number Generation	-	Supported	Supported

Table 3: Oracle SPARC Enterprise T-Series Processors - Supported Algorithms

Role Of Solaris Cryptographic Framework

The Solaris Cryptographic Framework (SCF) library plays a vital role for providing applications access to NCP and N2CP accelerators. SCF provides a set of cryptographic services for kernel-level and user-level consumers. Based on the PKCS#11 cryptography standard, the framework provides mechanisms and APIs whereby both kernel and userland based cryptographic functions can transparently use hardware accelerators configured on the system. SCF supports three types of cryptographic token providers, which include a user-level provider (a PKCS#11 shared library), a kernel software provider, and a kernel hardware provider (For example, NCP or N2CP accelerators of Oracle SPARC Enterprise T-Series processor). Applications looking to take advantage of hardware cryptography acceleration must go through the operating system kernel hardware provider. (Figure 3).

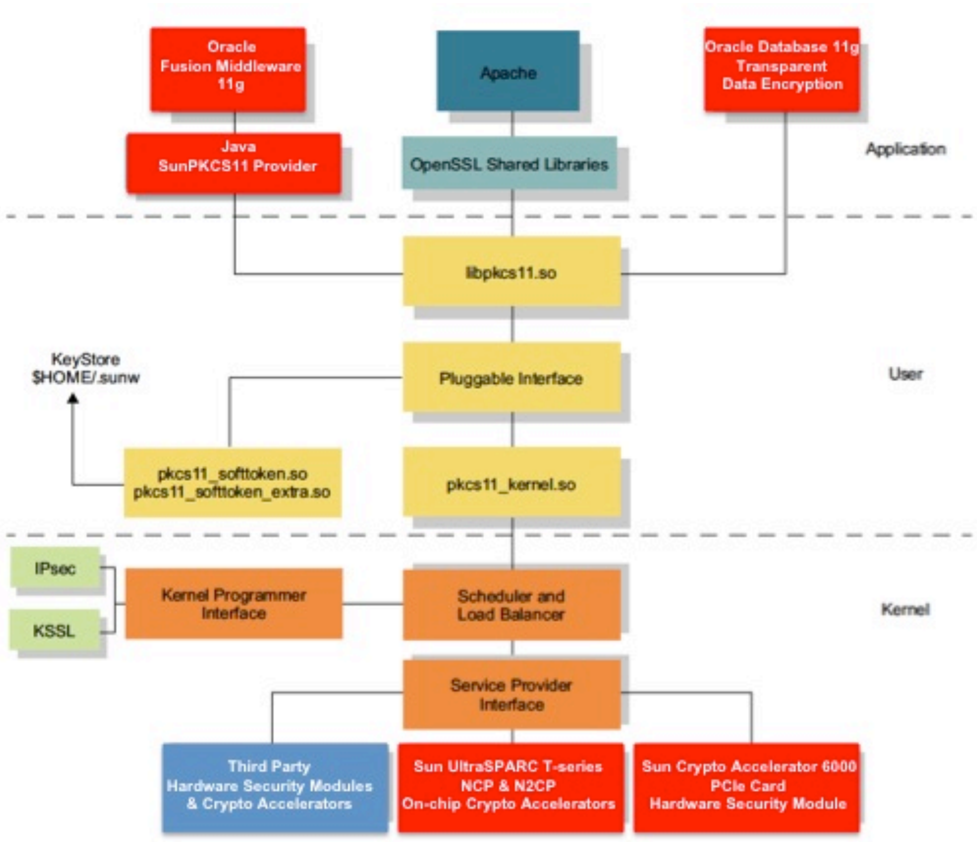

Oracle Fusion Middleware 11g
Apache
Oracle Database 11g Transparent Data Encryption
Java SunPKCS11 Provider
OpenSSL Shared Libraries
Application
libpkcs11.so
KeyStore $HOME/.sunw
Pluggable Interface
User
pkcs11_softtoken.so pkcs11_softtoken_extra.so
pkcs11_kernel.so
IPsec
KSSL
Kernel Programmer Interface
Scheduler and Load Balancer
Kernel
Service Provider Interface
Third Party Hardware Security Modules & Crypto Accelerators
Sun UltraSPARC T-series NCP & N2CP On-chip Crypto Accelerators
Sun Crypto Accelerator 6000 PCIe Card Hardware Security Module

Figure 3: Solaris Cryptographic Framework and its core components

Solaris Cryptographic Framework Components

Figure 3 describes the Solaris Cryptographic Framework (SCF). Consisting of a user-level framework, and a kernel-level framework, the SCF includes the following components:

- `libpkcs11.so`, the interface for user applications can link to in order to call functions in the user-level framework. For example, by default Oracle WebLogic server deployed on Sun SPARC servers use Java SunPKCS11 provider for getting access to *`libpkcs11.so`*.

- `Pluggable interface`, a PKCS#11-based interface that enables user-level providers to be plugged into the user-level framework.

- `pkcs11_softtoken.so`, user-level cryptographic mechanisms provided by the Solaris OS.
- `pkcs11_softtoken_extra.so,` a user-level library that is identical to `pkcs11_softtoken.so`, but supports stronger keys with bigger key sizes. `$HOME/.sunw/pkcs11_softtoken`, the default file system location of the per-user keystore in the Solaris OS.
- `pkcs11_kernel.so`, provides the hardware accelerated algorithms in the kernel-level framework to the user-level framework. Note that it does not contain any cryptography, and does not expose the software-only kernel implementation to the user-level framework.
- `Scheduler and load balancer,` the kernel software responsible for coordinating use, load balancing, and dispatching of the kernel cryptographic service requests.
- `Kernel provider interface`, the interface for kernel-level consumers of cryptographic services. IPSec and Kernel SSL (KSSL) are example consumers. The Kernel SSL proxy (KSSL), off-loads SSL processing from user applications by performing SSL handshakes at the Solaris kernel and enables them to transparently take advantage of hardware accelerators including on-chip accelerators of Oracle SPARC Enterprise T-Series processors.
- `Service provider interface (SPI)`, an interface that enables kernel providers to be plugged into the kernel-level framework. This includes hardware- or software-based cryptographic services.

To administer the multiple cryptographic providers (Accelerators and Hardware Security Modules) and its supported mechanisms, SCF provides a notion of `metaslot` that represents a virtual provider that aggregates all available cryptographic mechanisms to the user-level cryptographic framework. To support identifying, administering and managing cryptographic providers, SCF provides the `cryptoadm` utility that helps to perform the following tasks:

- Installing and uninstalling cryptographic providers
- Configuring the mechanism policy for each provider
- Displaying information about the framework

The `cryptoadm` utility provides a set of command-line options that are available for installing and uninstalling a cryptographic provider, and enabling and disabling the `metaslot`'s features and mechanisms of the cryptographic token provider. The following commands can be used to explore the cryptographic capabilities of Sun chip multithreading servers with cryptographic accelerators.

- To display the list of installed software- and hardware-based cryptographic providers.

  ```
  # cryptoadm list -v
  ```

  On Sun servers with UltraSPARC T2 processors, the displayed list of kernel hardware providers is similar to the following list.

  ```
  Kernel hardware providers:
  ncp/0
  n2rng/0
  ```

```
n2cp/0
```

- Use the `cryptoadm list -m` command to display the mechanisms provided by the available cryptographic providers. Use the following command to display the list of mechanisms provided by NCP on Sun servers with UltraSPARC T2 processors.

  ```
  # cryptoadm list -m  provider=ncp/0

  ncp/0:
  CKM_DSA, CKM_RSA_X_509, CKM_RSA_PKCS,
  CKM_RSA_PKCS_KEY_PAIR_GEN, CKM_DH_PKCS_KEY_PAIR_GEN,
  CKM_DH_PKCS_DERIVE, CKM_EC_KEY_PAIR_GEN,
  CKM_ECDH1_DERIVE, CKM_ECDSA
  ```

- To disable a selected mechanism policy (ex. `AES_CBC_PAD`) on an installed cryptographic provider (Softtoken) use the ***`enable`*** or ***`disable`*** sub-command. This helps to demonstrate the difference in encryption speed of hardware and software based cryptographic providers.

  ```
  # cryptoadm disable
           provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
                                       mechanism=CKM_AES_CBC_PAD
  ```

- To enable a selected mechanism policy on an installed cryptographic provider

  ```
  # cryptoadm enable provider=n2cp/0 mechanism=CKM_AES_CBC_PAD
  ```

Cryptographic Acceleration for SOA and Web Services Security

Both SOA and XML Web services based applications can significantly gain on security performance by offloading and delegating their cryptographic operations to the on-chip cryptographic accelerators of Oracle SPARC Enterprise T-Series servers.

Applied Security Mechanisms and Usage Scenarios

To enhance security performance, both the Oracle WebLogic server and Oracle WSM secured applications can offload select cryptographic operations to address the following security scenarios.

- Transport-layer Security
  - SSL/TLS acceleration offloads computationally intensive public-key cryptographic operations such as RSA, DH and ECC.
  - RMI over IIOP with SSL uses SSL/TLS to protect IIOP connections to RMI remote objects.
- Message-Layer Security
  - Acceleration of cryptographic operations intended for supporting XML Web Services security standards such as WS-Security, WS-SecurityPolicy. XML Web services security

relies on public-key encryption, digital signature (ex. RSA, DSA), bulk encryption (ex. AES, 3DES,DES) and message digest (ex. SHA-1, SHA-2, MD5) functions intended for supporting XML encryption, XML digital signature and related cryptographic operations.

Transport-Layer Security Acceleration

Both the Oracle WSM and its hosting WebLogic server rely on the underlying Java Cryptographic Extensions (JCE) provider for supporting its SSLv3/TLSv1 based secure communication. On Oracle Solaris 10 SPARC deployments, the Sun JCE provider is bundled with the Sun Java runtime environment, which facilitates the PKCS#11 interfaces for delegating the cryptographic operations intended for SSL/TLS protocols. The availability of Java PKCS#11 interfaces helps by off-loading and accelerating the cryptographic workloads of SSL/TLS protocols using NCP and N2CP accelerators of UltraSPARC T-series processors. Alternatively, Oracle WSM and the WebLogic server can make use of the Solaris KSSL as an SSL proxy for handling transport-layer security operations using SSL/TLS protocol. Like any other kernel module, KSSL tightly integrates with the Solaris kernel and automatically makes use of cryptographic acceleration provided by NCP and N2CP accelerators.

Using Sun JCE

By default, the Oracle WebLogic server on Oracle SPARC Enterprise T-Series servers relies on the JDK and its Sun JCE provider environment for handling cryptographic operations. The Sun JCE contains a PKCS#11 implementation (Java SunPKCS11) that enables Java applications to access hardware cryptographic tokens – such as Secure Sockets Layer (SSL) accelerators, Smartcards, and Hardware Security Modules (HSM). The SunPKCS11 provider is a Java based PKCS#11 implementation that integrates with underlying PKCS#11 implementations provided by the SCF and its exposed cryptographic providers. The SunPKCS11 provider does not implement its own cryptographic algorithms.

In a typical WebLogic server installation on Solaris, the Java runtime environment is pre-configured to make use of the SunPKCS11 provider. To verify this refer to the Java security properties file located at `$JAVA_HOME/jre/lib/security/java.security` properties file and make sure it identifies SunPKCS11 as the default provider.

```
security.provider.1=sun.security.pkcs11.SunPKCS11
                        ${java.home}/lib/security/sunpkcs11solaris.cfg
```

The `$JAVA_HOME/jre/lib/security/sunpkcs11-solaris.cfg` file contains the configuration information used by the SunPKCS11 provider for accessing the SCF. To help Java applications benefit from cryptographic acceleration, the `sunpkcs11-solaris.cfg` file allows users to disable the soft-token (user-level provider) mechanisms and in turn delegate them to the underlying hardware-based cryptographic token provider and the mechanisms. This enables the WebLogic server hosted applications and XML Web services to automatically delegate their cryptographic-intensive operations to the underlying cryptographic provider via the SCF (Figure 4).

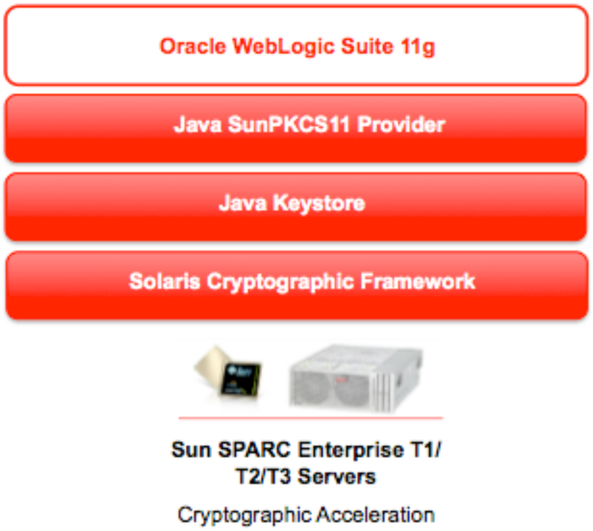

Oracle WebLogic Suite 11g
Java SunPKCS11 Provider
Java Keystore
Solaris Cryptographic Framework
Sun SPARC Enterprise T1/ T2/T3 Servers
Cryptographic Acceleration

Figure 4: Oracle WebLogic SSL using Hardware Assisted Cryptographic Acceleration

Accelerating SSL using SunPKCS11 Provider

The following steps explain how to configure Oracle WebLogic Server for SSL acceleration using the on-chip cryptographic acceleration capabilities of Oracle SPARC Enterprise T-Series servers.

1. Configure WebLogic Server to listen for SSL. Before configuration, make sure you obtain the private keys, server certificate including the public key and trust CA certificates from a Certificate Authority (CA) and then store them into the Java keystore (Identity and Trust keystores) configured within the WebLogic server environment. In case of development and testing, you may choose to use a self-signed certificate, private key and trusted CA certificate created using Java key tool. Use the WebLogic Server Administration Console to configure the identity and trust keystores.

   Follow the SSL configuration guidelines specified in the *Oracle Fusion Middleware - Securing WebLogic Web Services for Oracle WebLogic Server 11g* guide.

2. Verify and confirm that the Oracle WebLogic Server is listening and responds over the SSL port.

3. Enable cryptographic acceleration by editing the Java SunPKCS11 provider configuration file located at `$weblogic-javahome/jre/lib/security/sunpkcs11-solaris.cfg`. This file contains vital attributes for allowing Oracle WebLogic server to access the cryptographic mechanisms and attributes supported by the underlying on-chip cryptographic accelerator, which can be enabled or disabled in the SunPKCS11 configuration file.

4. It is important to enable and enforce delegation of the required cryptographic mechanisms to the underlying PKCS#11 provider that facilitates the hardware accelerator support. Make sure to include the required public key cryptographic mechanisms (ex. `CKM_RSA_PKCS`) in the

Java SunPKCS11 provider configuration file that lists as part of `enabledMechanisms` list or removes the mechanisms from the list of `disabledMechanisms` of the Java SunPKCS11 configuration file. Doing so forces the required public key operations to be performed by the NCP. Additionally, you may enable or disable the bulk encryption and message digest algorithms in the list that forces those operations performed by the N2CP accelerator. The following example shows a sample SunPKCS11 provider configuration.

```
name = Solaris
description = SunPKCS11 accessing Solaris Cryptographic
Framework
library = /usr/lib/$ISA/libpkcs11.so
handleStartupErrors = ignoreAll
attributes = compatibility
disabledMechanisms = {
CKM_MD2
CKM_MD5
CKM_SHA256
CKM_SHA384
CKM_SHA512
CKM_DSA_KEY_PAIR_GEN
CKM_TLS_KEY_AND_MAC_DERIVE
CKM_RSA_PKCS_KEY_PAIR_GEN
CKM_SSL3_KEY_AND_MAC_DERIVE
}
```

5. Restart the Oracle WebLogic server

6. The `cryptoadm` administration utility provided in the Oracle Solaris OS can be used to verify and ensure the off-loading of cryptographic operations, install or uninstall softtoken providers, and enable or disable mechanisms at softtoken providers to enforce acceleration at hardware accelerators (ex. ncp/0 and n2cp/0) and its exposed cryptographic mechanisms.

   - To display the mechanisms supported by the installed cryptographic providers

     ```
     # cryptoadm list -m
     ```

   - To uninstall and install the PKCS11 softtoken implementation.

     ```
     # cryptoadm   uninstall
     provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so

     # cryptoadm   install
     provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
     ```

   - To disable and enable cryptographic mechanisms in the provider implementation. For example, to disable the `CKM_MD5_RSA_PKCS` and `CKM_SHA1_RSA_PKCS` mechanisms from the softtoken implementation and enable them on ncp/0. And disable `CKM_AES_CBC` and `CKM_AES_CBC_PAD` mechanisms from the softtoken implementation and enable them on n2cp/0.

     ```
     # cryptoadm  disable
     ```

```
provider=/usr/lib/security/\$ISA/pkcs11_softtoken.so
            mechanism=CKM_MD5_RSA_PKCS, CKM_SHA1_RSA_PKCS

# cryptoadm  enable
         provider=ncp/0
           mechanism= CKM_MD5_RSA_PKCS, CKM_SHA1_RSA_PKCS
```

Using Solaris KSSL

KSSL is a Solaris kernel module that acts as a server-side SSL protocol for offloading operations such as SSL/TLS-based communication, SSL/TLS termination, and reverse proxies for end-user applications. KSSL takes advantage of the SCF to act as an SSL proxy server, performing complete SSL handshake processing in the Solaris OS kernel. KSSL uses the underlying hardware cryptographic accelerators (NCP and N2CP), PKCS#11 keystores, and Hardware Security Modules for enabling SSL acceleration and secure key storage.

The key technology aspects and the security benefits of using KSSL include:

- Helps to introduce—non-intrusively—an SSL proxy server for Web servers, Java EE application servers, and applications that do not support SSL.

- Listens to secured requests on the designated SSL port (ex. http://:443) and renders cleartext traffic via a reverse proxy port (ex. http://:7001) for an underlying WebLogic application server or a load balancer that supports multiple instances of application servers (Oracle WebLogic Managed Servers). In a real-world scenario, KSSL proxy can reside in the Solaris OS global zone and redirect to load balancer that performs round-robin delivery of requests/responses to a set of WebLogic managed servers running in non-global zones.

- All SSL operations, including the SSL handshake and session state, are performed asynchronously in the Solaris kernel and without the knowledge of the target application server. Automatically uses the Solaris Cryptographic Framework for off-loading operations to the underlying hardware cryptographic accelerators (NCP and N2CP). No extra effort is required.

- Manages all SSL certificates independently and supports most standard formats, including PKCS12 and PEM. Key artifacts can be stored in a flat file (OpenSSL) or a PKCS#11 conforming keystore (ex. HSMs, NSS, Solaris PKCS#11 Sofftoken) to help ensure the protection of private keys.

- Supports the use of Solaris Zones. Each IP-identified zone can be configured with a KSSL proxy.

- Delivers 25% to 35% faster SSL performance compared to traditional SSL configurations used in popular Web servers and Java application servers, such as Oracle WebLogic Server and Oracle GlassFish application Server.

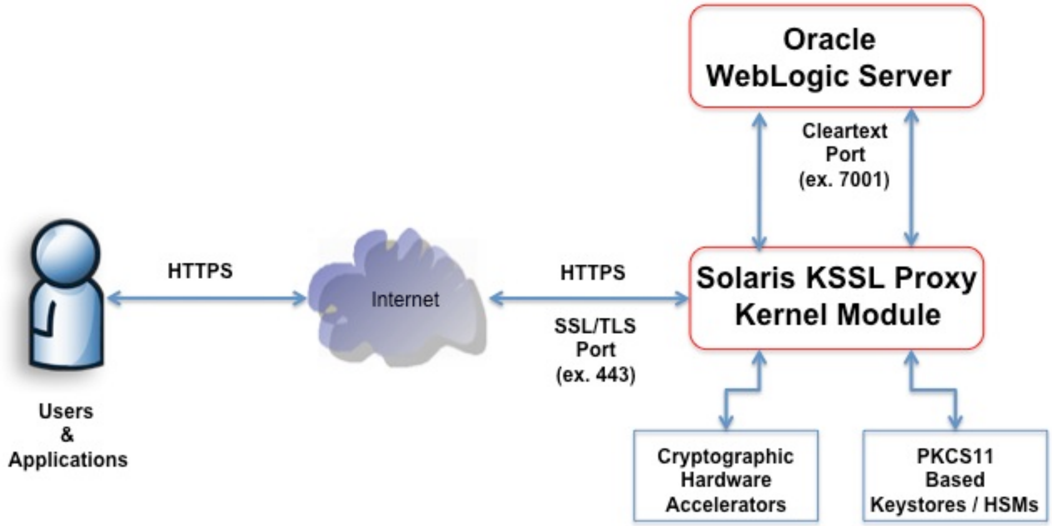

Oracle WebLogic Server
Cleartext Port (ex. 7001)
Users & Applications
HTTPS
Internet
HTTPS
SSL/TLS Port (ex. 443)
Solaris KSSL Proxy Kernel Module
Cryptographic Hardware Accelerators
PKCS11 Based Keystores / HSMs

Figure 5: Using Solaris KSSL for Oracle WebLogic SSL

Configuring KSSL for WebLogic SSL Acceleration

Using the KSSL kernel module as an SSL proxy (Figure 5) requires obtaining and installing a certificate from a Certificate Authority[1]. Here are the steps to configure a testing KSSL accelerator:

Using OpenSSL Certificates (Flat-file Keystore)

1. Create a self-signed certificate using `openssl` utility

```
# /usr/sfw/bin/openssl req -x509 -nodes -days 365
           -subj  /C=US/ST=State/L=City/CN=serverhostname
               -newkey rsa:1024 -keyout  /etc/pki/key00.pem
                  -out  /etc/pki/cert00.pem
```

2. Concatenate and place all certificate artifacts in a single file.

```
# cat cert00.pem key00.pem > /etc/pki/mySSLCerts.pem
```

3. Move to the `/etc/pki` directory and execute the following command

```
# chown 600 mySSLCerts.pem
```

4. Configure the KSSL proxy and its redirect HTTP cleartext port. Assuming the Oracle

---

[1] For production deployments the use of a Certificate Authority is essential. However, a self-signed certificate can also be used for testing purposes.

WebLogic server default listen port, or cleartext port, is port 7001. Make sure the `/etc/pki/passwordfile` includes the password of the keystore.

```
# ksslcfg create -f pem -i /etc/pki/mySSLCerts.pem
           -x 7001 –p /etc/pki/passwordfile serverhostname 443
```

5. Use the Service Management Facility (SMF) to verify that KSSL service is enabled.

```
# svcs -a | grep “kssl”
```

6. Alternatively, use the Solaris ‘netstat –an’ command to verify KSSL is listening on port 443.

```
# netstat -an | grep 443
```

7. Use a Web browser to check that the Oracle WebLogic server listens to the KSSL secured port. Go to `https://myservername.com:443`

Using PKCS#11 based Keystores and Hardware Security Modules

To ensure the security of private-key and server certificates and tamper-proof keystores, it is often recommended to use Hardware Security Modules (HSM). KSSL supports usage of PKCS#11 based HSMs (ex. Sun Crypto Accelerator 6000 PCIe Card), Software keystores (For ex. NSS, Solaris PKCS#11 softtoken). The following command and options are typically used for configuring PKCS#11 based keystores.

To configure KSSL using a Solaris PKCS#11 softtoken the steps are as follows:

1. Configure a “Sun Software PKCS#11 Softtoken” keystore using SCF `pktool` utility.

```
# pktool setpin keystore=pkcs11
```

2. Create a self-signed certificate

```
# pktool  gencert  keystore=pkcs11  label=”ksslCert”
        subject=”C=US,O=Oracle, OU=ISVE, CN=serverhostname”
                      serial=0x000000001
```

3. Enable “Sun Software PKCS#11 Softtoken” token as metaslot

```
# cryptoadm enable metaslot
                token=“Sun Software PKCS#11 Sofftoken”
```

4. Configure the KSSL service identifying the PKCS#11 keystore assuming the Softtoken is created in the home directory of the user and the certificate alias is `ksslCert`. Make sure the `/etc/pki/passwordfile` includes the password of the keystore.

```
# ksslcfg create –f pkcs11 –d $HOME/.sunw
      -T “Sun Software PKCS#11 Sofftoken”
        -C “ksslCert”  -p /etc/pki/passwordfile
           -x 7001 serverhostname  443
```

5. To verify that KSSL service is enabled, execute the **`svcs`** command as follows:

```
# svcs -a | grep “kssl”
```

To configure KSSL using Sun Crypto Accelerator 6000 PCIe card as a HSM keystore - the steps are as follows:

1. Configure and verify that the Sun Crypto Accelerator 6000 PCIe card is initialized for use as a HSM. Refer *to Sun Cryptographic Accelerator 6000 PCIe Card Version 1.1 Documentation* for installation and configuration.
    a. The card is configured to use in FIPS mode and with a Primary Security Officer.
    b. A keystore is made available for use as a PKCS#11 token
    c. Create self-signed certificate using `pktool` or import SSL certificates (in PKCS12 format) obtained from a certificate authority.
2. Enable the newly created keystore as a metaslot.

```
# cryptoadm enable metaslot
                token=“sca-keystore”
```

3. Configure the KSSL service identifying the PKCS#11 keystore assuming the password is stored in a file and the certificate alias is `ksslCert`. Make sure the `/etc/pki/passwordfile` includes the password of the keystore.

```
# ksslcfg create –f pkcs11
      -T “Sun Metaslot”
            -C “ksslCert”
                  -p /etc/pki/passwordfile
                        -x 7001 serverhostname  443
```

4. To verify that KSSL service is enabled, use the **`svcs`** command as follows:

```
# svcs -a | grep “kssl”
```

Using SSL Cipher Suites

To enforce the KSSL service negotiates with the end-user client (ex. Web browser) using specific SSL3/TLSv1 ciphersuites, the `kssl config` command must use `-c` option followed by the required list of `<ciphersuites>` in a sorted order. For example:

```
# ksslcfg create –c rsa_3des_ede_cbc_sha,rsa_des_cbc_sha
           -f pem -i /etc/pki/mySSLCerts.pem
            -x 7001 –p /etc/pki/passwordfile serverhostname 443
```

Message-Layer Security Acceleration

Oracle WSM plays a vital role in configuring and deploying message-layer security for SOA and XML Web services applications. The availability of PKCS#11 interfaces via Java Cryptographic Extension (JCE) framework enables the Oracle WSM to take advantage of NCP and N2CP accelerators by off-loading WS-Security based cryptographic workloads.

Accelerating Message-Layer Security using Oracle WSM

The following steps explain how to configure Oracle WSM for message-layer security acceleration using the on-chip cryptographic acceleration capabilities of Oracle SPARC Enterprise T-Series servers. As a pre-requisite, it is assumed that Oracle WSM is installed and pre-configured with installation Oracle Fusion Middleware – SOA suite

1. To begin with, we will create and use a JKS keystore for supporting the initial storage of keys and certificates used with Oracle WSM. The steps for creating the JKS keystore and the key pairs for use with Oracle WSM is as follows:

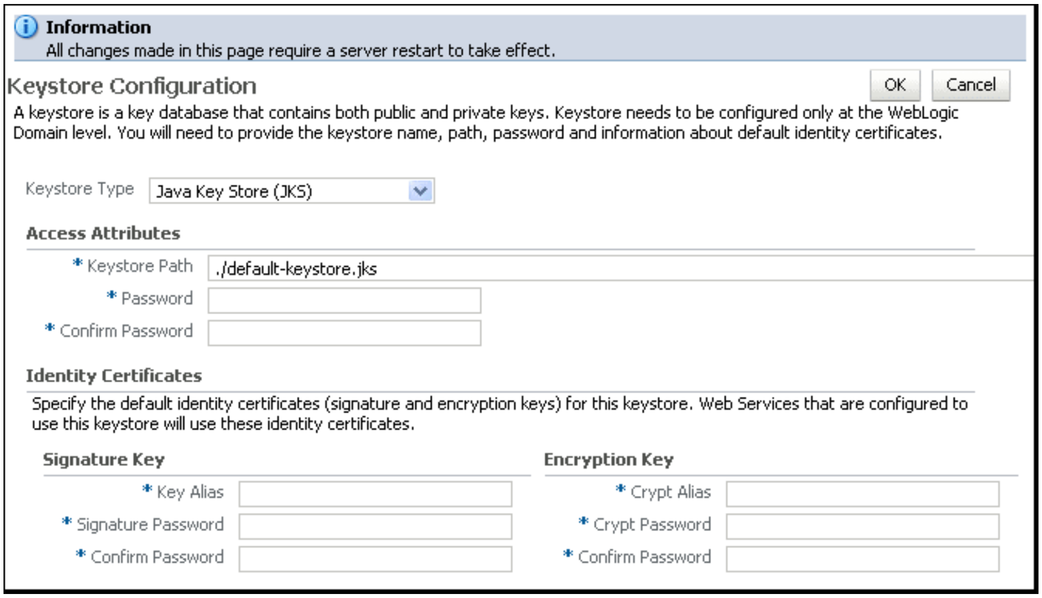
    - Create a new RSA key pair (private and public keys) for use with Oracle WSM to support signing and encrypting SOAP messages. Use the Java keytool with -genkey option to create a new RSA key pair with RSA-SHA1 as the signature algorithm. The following example shows creating a Java keystore `default-keystore.jks` and a key pair with alias `orakey` using key algorithm `RSA` and signature `SHA1withRSA`.

    ```
    keytool -genkey -alias orakey -keyalg RSA
                -sigalg SHA1withRSA -dname
                         "CN=orakey, OU=Testing, C=US"
                              -keystore  default-keystore.jks
    ```

    In case of production deployments, it is strongly suggested to acquire the Private/Public key pair and its certificate from a Certificate authority.

2. Configure Oracle WSM to use the newly created keystore. Here are the steps to configure the keystore for use with Oracle WSM:

    - Login to **`Enterprise Manager`**, which is available by default as it is installed with Oracle Fusion Middleware suite at http://localhost:7001/em
    - In the navigator pane, expand **`WebLogic Domain`** to show the domain for which you need to configure the keystore. Select the domain.
    - Using Fusion Middleware Control, click **`Weblogic Domain`**, then **`Security`**, and then **`Security Provider Configuration`**.
    - Click the plus sign (+) to expand the **`Keystore`** control near the bottom of the page, then click **`Configure`**. The Web Services Manager Keystore Configuration page is displayed, as shown in (Figure 6).



Information

All changes made in this page require a server restart to take effect.

Keystore Configuration

A keystore is a key database that contains both public and private keys. Keystore needs to be configured only at the WebLogic Domain level. You will need to provide the keystore name, path, password and information about default identity certificates.

Keystore Type: Java Key Store (JKS)

Access Attributes

* Keystore Path: ./default-keystore.jks

* Password

* Confirm Password

Identity Certificates

Specify the default identity certificates (signature and encryption keys) for this keystore. Web Services that are configured to use this keystore will use these identity certificates.

Signature Key

* Key Alias

* Signature Password

* Confirm Password

Encryption Key

* Crypt Alias

* Crypt Password

* Confirm Password

Figure 6: Web Services Manager Keystore Configuration

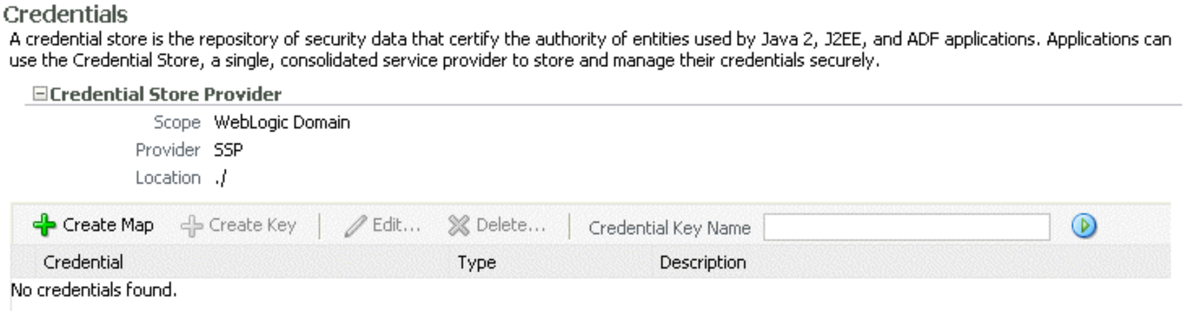
- Select Keystore Type as JKS. Enter the path and name of the keystore - `default-keystore.jks` created for use with Oracle WSM. Also, enter the password for the keystore and re-confirm it.

- Next, enter the alias of the signature and encryption keys and the corresponding key passwords. It is important to the alias and password for the signature and encryption keys define the string alias and passwords used to store and retrieve the keys.
- Click on OK to submit and save the changes. Restart the SOA server and then relaunch the Oracle Enterprise Manager Fusion Middleware Control to take effect on the changes.

3. Optionally, configure an application-specific credential store provider to support storing, retrieving, and deleting credentials intended for Web services and other applications. For example, if you are using any username token policy, it is required to configure `csf-key` property, with a default value of `basic.credentials`. Oracle WSM uses this key to create and retrieve username/password token. The following steps are required to create a credential store key:

   - Login to **`Enterprise Manager`**. In the navigator pane, expand **`WebLogic Domain`** to show the domain for which you need to configure the credential store. Select the domain.

   - Using Fusion Middleware Control, click **`Weblogic Domain`**, then **`Security`**, and then **`Credentials`**. The Credential Store Provider Configuration page is displayed, as shown in Figure 7.



Credentials

A credential store is the repository of security data that certify the authority of entities used by Java 2, J2EE, and ADF applications. Applications can use the Credential Store, a single, consolidated service provider to store and manage their credentials securely.

Credential Store Provider

Scope WebLogic Domain

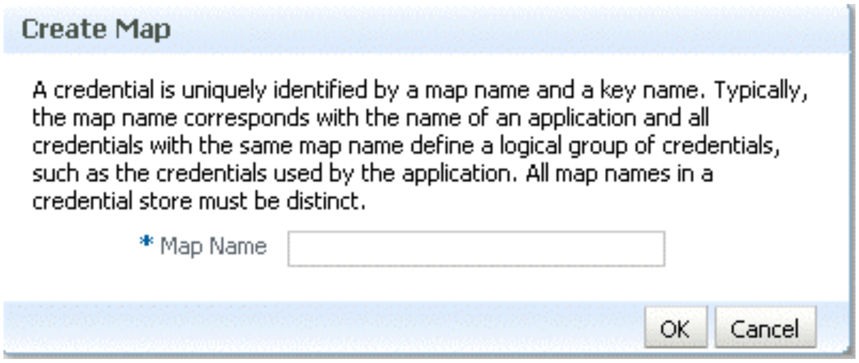
Provider SSP

Location ./

No credentials found.

Figure 7: Credential Store Provider Configuration Page

- Click `Create Map` and enter the map name as `oracle.wsm.security`, as shown in Figure 8. If this map is already created, then just skip this step.

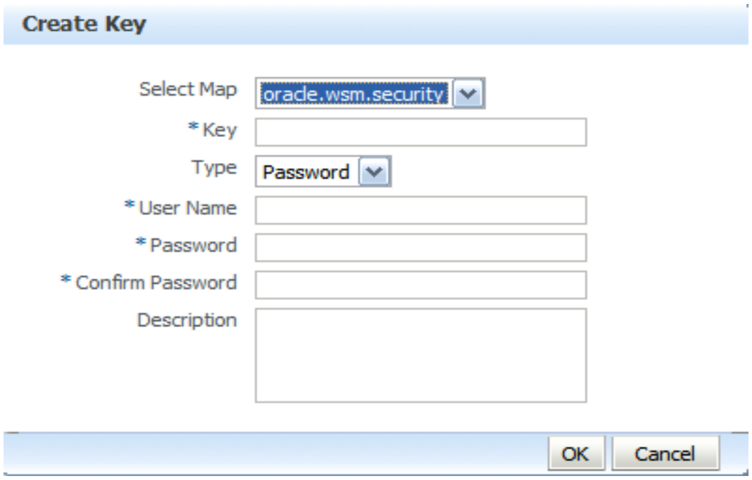


Create Map

A credential is uniquely identified by a map name and a key name. Typically, the map name corresponds with the name of an application and all credentials with the same map name define a logical group of credentials, such as the credentials used by the application. All map names in a credential store must be distinct.

Figure 8: Create Map Dialog

- Next, Click `Create key` to launch the Create Key dialog as shown in Figure 9.



Select Map: oracle.wsm.security

Type: Password

Figure 9: Create Key Dialog

- Select the map name `oracle.wsm.security`, if it is not already selected, enter the key name - `basic.credentials`.

- Select the key type, either `Password` or `Generic`. A password credential can store a username and password. A generic credential can store any other credential object. For a password credential, enter the username and password. Click OK.

- The above configuration is intended to support JKS keystore for verifying Oracle WSM setup. To enable cryptographic acceleration of Oracle WSM, it is required to migrate the keys to a PKCS#11 keystore.

- To verify the configuration using JKS keystore, it is recommended to test the configuration by deploying a sample Web service and attaching a OWSM policy. Refer to Steps 8 and 9 for test-driving a sample Web services and attaching policies.

4. It is quite critical to create PKCS#11 based keystore, before migrating the keys. Solaris 10 supports creating PKCS#11 keystore "Sun Software PKCS#11 Sofftoken" and NSS.

    - Configure a "Sun Software PKCS#11 Sofftoken" keystore using SCF `pktool` utility. Set the PIN/Passphrase for access the

```
# pktool setpin keystore=pkcs11
  Create new passphrase:
```

```
Re-enter new passphrase:
```

- Enable "Sun Software PKCS#11 Softtoken" token as metaslot

```
# cryptoadm enable metaslot
    token="Sun Software PKCS#11 Softtoken"
```

- Make sure the "Sun Software PKCS#11 Softtoken" is available as a metaslot and available to access using Java keytool. If it is prompted, enter the passphrase of "Sun Software PKCS#11 Softtoken".

```
# cryptoadm list –v metaslot

# keytool -list -storetype pkcs11 –keystore NONE
```

5. Using the keytool to migrating the keys from the JKS keystore to "Sun Software PKCS#11 Softtoken".

```
# keytool –importkeystore
      -srckeystore /opt/Oracle/Middleware/default-keystore.jks
       -destkeystore NONE -srcstoretype JKS
         -deststoretype PKCS11
           -srcstorepass changeme -deststorepass scfpassword
```

6. Verify whether the keys are present in PKCS11/HSM are accessible via keytool.

```
#  keytool -list -storetype pkcs11 -keystore NONE

Enter keystore password:
Keystore type: PKCS11
Keystore provider: SunPKCS11-Solaris

Your keystore contains 1 entries
SecretKeyEntry,
orakey, PrivateKeyEntry,
Certificate fingerprint (MD5):
EC:75:AC:21:C5:6F:6C:B5:49:2A:75:81:BB:D1:49:94
```

7. Update the PKCS#11 keystore "Sun Software PKCS#11 Softtoken" password in `Enterprise Manager` and change the keystore type to PKCS11.

   - To update PKCS11 keystore password using `Enterprise Manager`, follow the steps discussed in Step 2 and update the keystore password to "Sun Software PKCS#11 Softtoken" password.

   - Leave existing keystore type as JKS or select it if not already selected. Leave existing keystore path or enter path if it is not entered. Changing keystore type is normally done using Enterprise manager where we updated the keystore password but the current release Enterprise manager supports JKS and third-party HSMs. For using PKCS#11 based HSM or Softtoken keystores it is required to update the `jps-config.xml`.

- To update keystore type as PKCS11 in `jps-config.xml`, the steps are as follows:

  a. Navigate to your `$SOA_DOMAIN_HOME/config/fmwconfig`

  ```
  # cd $SOA_DOMAIN_HOME/config/fmwconfig
  ```

  b. Edit the `jps-config.xml` using a texteditor or XML editor

  ```
  # vi jps-config.xml
  ```

  Go at the last line, you will find the `<jpsContexts default="default">` element.

  c. Under `jpsContexts` element you will find `serviceInstanceRef` elements. Identify the element, showing `ref` is `keystore.inst.0` or the `ref` is just `keystore.`

  ```
  <serviceInstanceRef ref="keystore.inst.0"/>
  ```

  d. Search for `keystore.inst.0` element in the file, you will end up with `keystore.serviceInstance.`

  e. In the `keystore.serviceInstance` element you will find the property `keystore.type` and its value as `JKS`, change its value to `PKCS11`.

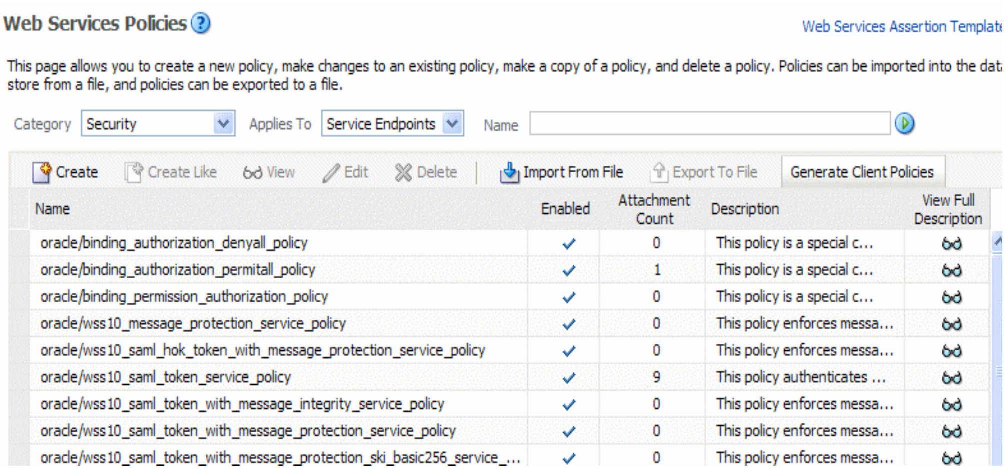
  f. Save and close the file. Finally, restart the SOA server.

8. To verify the setup, deploy an XML Web service application. Refer to documentation - Deploying Web Services Applications.
   - http://download.oracle.com/docs/cd/E12839_01/web.1111/b32511/deploy.htm

9. To secure the Web service, assign the deployed XML Web service with Oracle WSM WS-Security Policies. To verify and test the configuration using PKCS11 keystore and enabling cryptographic acceleration, we need assign any, policy which supports signing and encryption. Signing and encryption policies are identified with "`*message_protection*`" (For example: "`wss10_message_protection_service_policy`").

   - Go to the Enterprise Manager navigator pane, expand **`WebLogic Domain`** to show the domain for which you want to see the policies. Select the domain.

   - Using Fusion Middleware Control, click **`WebLogic Domain`**, then **`Web Services`** and then **`Policies`**. The Web Services Policies page is displayed as follows (Figure 10):



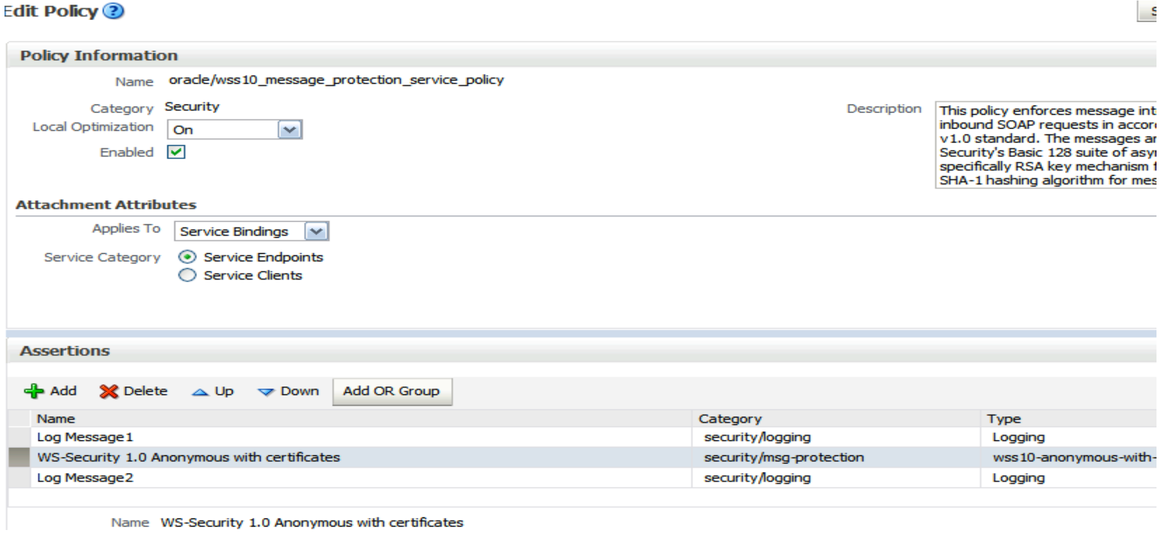
This page allows you to create a new policy, make changes to an existing policy, make a copy of a policy, and delete a policy. Policies can be imported into the dat store from a file, and policies can be exported to a file.

Category: Security
Applies To: Service Endpoints
Name:

Name	Enabled	Attachment Count	Description	View Full Description
oracle/binding_authorization_denyall_policy	✓	0	This policy is a special c...	👓
oracle/binding_authorization_permitall_policy	✓	1	This policy is a special c...	👓
oracle/binding_permission_authorization_policy	✓	0	This policy is a special c...	👓
oracle/wss10_message_protection_service_policy	✓	0	This policy enforces messa...	👓
oracle/wss10_saml_hok_token_with_message_protection_service_policy	✓	0	This policy enforces messa...	👓
oracle/wss10_saml_token_service_policy	✓	9	This policy authenticates ...	👓
oracle/wss10_saml_token_with_message_integrity_service_policy	✓	0	This policy enforces messa...	👓
oracle/wss10_saml_token_with_message_protection_service_policy	✓	0	This policy enforces messa...	👓
oracle/wss10_saml_token_with_message_protection_ski_basic256_service_...	✓	0	This policy enforces messa...	👓

Figure 10: Available Web Services Policies

- To attach a select security policy to a deployed Web Service, Refer to documentation: Attaching Policies to Web Services.
    - http://download.oracle.com/docs/cd/E12839_01/web.1111/b32511/attaching.htm
- To edit the OWSM policy and change the default algorithm suite (Default is `Basic128`) to use a different algorithm (ex. `Basic256Rsa15` refers to AES-256 and Rsa15 key wrap), the steps are follows:
    - Navigate to the `Web Services Policy` page, as described in section 7 of Attaching Policies to Web Services.
    - From the `Web Services Policies` page, select the attached policy from the `Policies` table and click Edit (in order to change algorithm suite, you should select any of the message protection policy).
    - On the `Edit Policy` page (Figure 11), in `Assertions` section select assertion that has `msg-protection` category. Generally such policy is selected by default.



Name: oracle/wss10_message_protection_service_policy
Category: Security
Local Optimization: On
Enabled: ☑
Description: This policy enforces message int inbound SOAP requests in accor v1.0 standard. The messages ar Security's Basic 128 suite of asy specifically RSA key mechanism SHA-1 hashing algorithm for mes

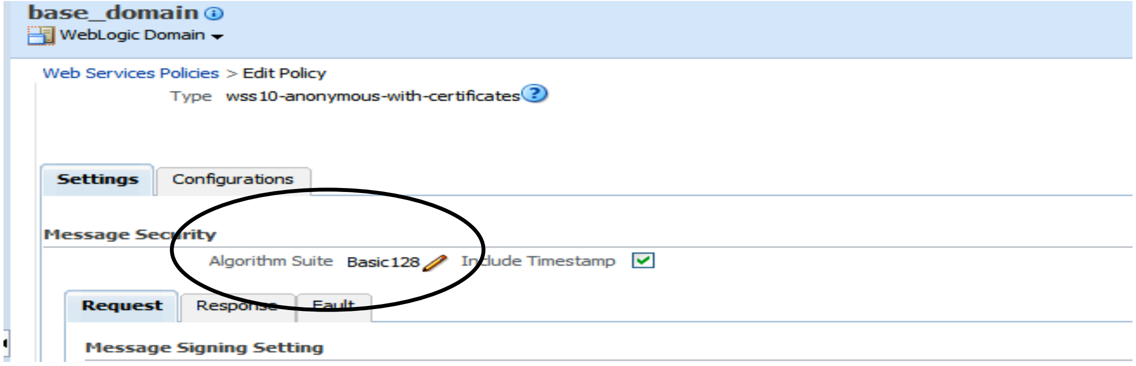
Applies To: Service Bindings
Service Category: ● Service Endpoints ○ Service Clients

Name	Category	Type
Log Message1	security/logging	Logging
WS-Security 1.0 Anonymous with certificates	security/msg-protection	wss10-anonymous-with-
Log Message2	security/logging	Logging

Name: WS-Security 1.0 Anonymous with certificates

Figure 11: Editing Web Services Policies

- Click on the `Settings` tab, you will find Algorithm Suite as `Basic128` (Figure 12).



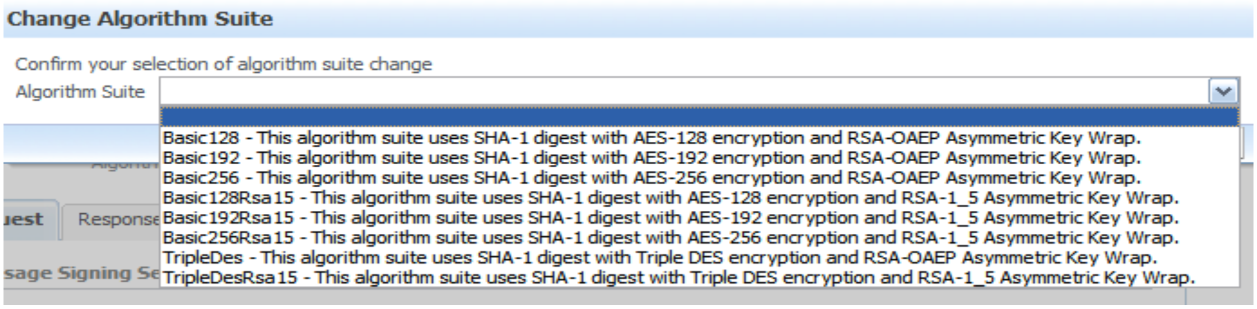
Type wss10-anonymous-with-certificates

Algorithm Suite Basic128

Include Timestamp ☑

Figure 12: Changing the Algorithm Suite

- Click on the pencil sign on the Algorithm Suite, `Change Algorithm Suite` page will appear. Select the desired algorithm (Figure 13) and Click Save. The changes will take effect at the next polling interval for policy changes. If you are using a database-based metadata repository, each time you save a change to your policy, a new version is created, and the older versions are retained.



Confirm your selection of algorithm suite change

Basic128 - This algorithm suite uses SHA-1 digest with AES-128 encryption and RSA-OAEP Asymmetric Key Wrap.
Basic192 - This algorithm suite uses SHA-1 digest with AES-192 encryption and RSA-OAEP Asymmetric Key Wrap.
Basic256 - This algorithm suite uses SHA-1 digest with AES-256 encryption and RSA-OAEP Asymmetric Key Wrap.
Basic128Rsa15 - This algorithm suite uses SHA-1 digest with AES-128 encryption and RSA-1_5 Asymmetric Key Wrap.
Basic192Rsa15 - This algorithm suite uses SHA-1 digest with AES-192 encryption and RSA-1_5 Asymmetric Key Wrap.
Basic256Rsa15 - This algorithm suite uses SHA-1 digest with AES-256 encryption and RSA-1_5 Asymmetric Key Wrap.
TripleDes - This algorithm suite uses SHA-1 digest with Triple DES encryption and RSA-OAEP Asymmetric Key Wrap.
TripleDesRsa15 - This algorithm suite uses SHA-1 digest with Triple DES encryption and RSA-1_5 Asymmetric Key Wrap.

Figure 13: Changing the Algorithm Suite

10. Make sure to include the cryptographic mechanisms specified in the WS-SecurityPolicy algorithm suite is include in the list of `enabledMechanisms` or by removing them from the list of

`disabledMechanisms` of the Java SunPKCS11 configuration file. If the specified algorithm suite is `Basic256Rsa15`, it uses `Aes256` encryption and `Rsa-oaep-mgf1p` for key wrap. To enable acceleration, you need to remove the required bulk encryption algorithms in the `disabledMechanisms` list that forces those operations (ex. `CKM_AES`) performed by the N2CP accelerator.

11. In some cases, the SunPKCS11 provider may not able to support algorithm suites where the mechanisms requires aggregated access to NCP and N2CP capabilities. For example, the XML signature operations with `SHA1withRSA`, algorithm will use the Solaris softtoken provider mechanism CKM_SHA1_RSA_PKCS, as advertised by the metaslot. Under normal circumstances, the metaslot will route `CKM_SHA1_RSA_PKCS` to the Softtoken provider because NCP does not implement it. But because sunpkcs11-solaris.cfg file has enabled the `CKM_SHA1_RSA_PKCS` mechanism in the SunPKCS#11 provider (or disable delegation of the `CKM_SHA1_RSA_PKCS` mechanism in the `Softtoken`), the Java technology security framework will fall back to using the NCP mechanism `CKM_RSA_PKCS` for performing the `RSA` signing operations and the `SHA1` hashing will be done separately.

Examining On-Chip Cryptographic Accelerator Operations

After deployment of SSL or WS-Security mechanisms, it is important to be verified and tested to be operational. The SCF commands `cryptoadm` and `kstat` can be used to verify and confirm NCP (`ncp/0`) or N2CP (`n2cp/0`) accelerator drivers are being utilized for performing the cryptographic operations. Since the Sun `metaslot` chooses the first hardware slot available for performing cryptographic operations, it is important to examine and confirm whether the configured on-chip hardware accelerator is acting on those delegated operations or any other software providers performing them. For example, in a KSSL usage scenario a positive and growing value of `rsaprivate` and `rsapublic` jobs indicates that ncp/0 is operational on the SSL workload. In all the deployed scenarios, it is assumed that the server is not running any other security applications such as SSH. To verify this and also check the number of operations performed by ncp/0 (since the last reboot), run the Oracle Solaris `kstat` command:

```
# kstat -n ncp0 -s rsaprivate

# kstat -n ncp0 -s rsapublic
```

In case of WS-Security operations, to verify and confirm the bulk encryption and message digest functions (ex. DES, 3DES, AES, RC4, SHA1, SHA256, MD5) performed on the systems with UltraSPARC T2 processors using n2cp/0 accelerator (since the last boot). For example, in an XML encryption scenario using AES algorithm, a positive and growing value of `aes` jobs indicates that n2cp/0 is operational on the target AES bulk encryption payloads.

```
# kstat -n n2cp0 —s aes
```

Performance Characteristics

Scenario 1: End-to-End SSL Performance

The following graph (Figure 14) represents the end-to-end SSL operations performance characteristics of the following WebLogic SSL scenarios.

a. Use KSSL as SSL proxy for the XML Web Services endpoint and enable Sun Software PKCS#11 Softtoken based keystore for supporting keys and certificates – By default on Oracle SPARC Enterprise T-Series servers, KSSL automatically use NCP and N2CP for cryptographic acceleration.

b. WebLogic Managed Server instances configured to use SSL with JKS keystore – configured to use SunPKCS11 provider for enabling hardware assisted cryptographic acceleration.

c. WebLogic Managed Server configured to use SSL (No acceleration)

HP Loadrunner was used as the load driver for deriving SSL performance, simulating from 100 to 1000 concurrent users for this test. The tests ran a Java EE/JAX-WS Web Services using a 500k XML payload (sample application bundled with WebLogic 10.3.3) deployed on Oracle WebLogic server 10.3.3 running on Oracle SPARC Enterprise T3-1 (Single socket) server.

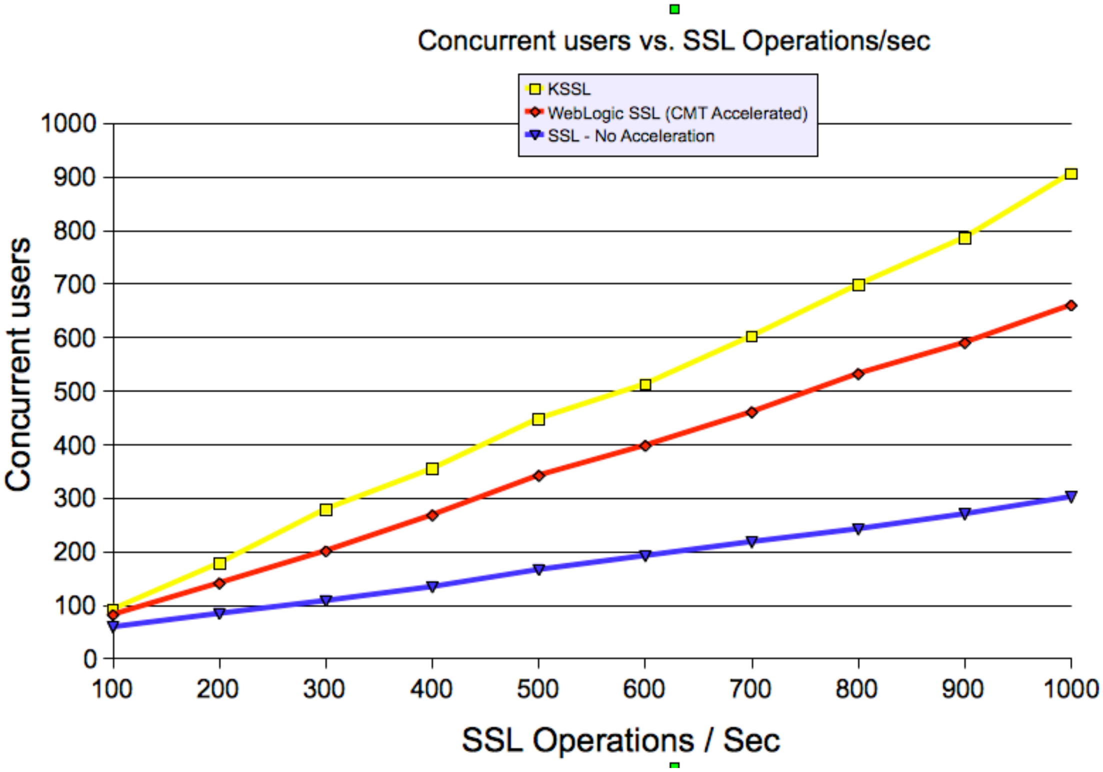


Figure 14: WebLogic SSL - SSL Performance Characteristics

Machine used:
Oracle SPARC Enterprise T3-1 (Single socket), 64 Gb Memory

Software configuration:
Oracle WebLogic 10.3.3 (1 Admin server, 4 Managed servers)
Oracle Fusion Middleware SOA Suite 11gR1 (11.1.1.4.0)
Solaris 10 Update 9

No. of Concurrent users:
`1000`

RSA Keysize used:
`1024 bit`

JCE enforced ciphersuites:
`SSL_RSA_WITH_3DES_EDE_CBC_SHA`

KSSL enforced ciphersuites:
`rsa_3des_ede_cbc_sha,rsa_des_cbc_sha`

As a result, enabling on-chip acceleration for SSL scenarios solidly delivered between **200% – 300% overall application performance gain including SSL operations** in comparison with Weblogic SSL running with no acceleration. More importantly, using Solaris KSSL as an SSL proxy provided an additional **performance gain of about 25-30%** outperforming WebLogic server SSL configured using SunPKCS11 provider for enabling CMT acceleration.

Scenario 2: Effect of SSL and WS-Security

The following graph (Figure 15) represents the comparison of end-to-end overall application throughput performance including both SSL and WS-Security operations using hardware-assisted cryptographic acceleration and without using them. The tests ran using the existing setup described in Scenario 1, with the addition of ~800k SOAP binary attachment payload and the service is configured with Oracle WSM enforced policy (`wss11_username_token_with_message_protection`) using Algorithm suite `Basic256Rsa15` (AES-256 for message encryption, SHA-1 for message digest, and Rsa15 for key wrap).

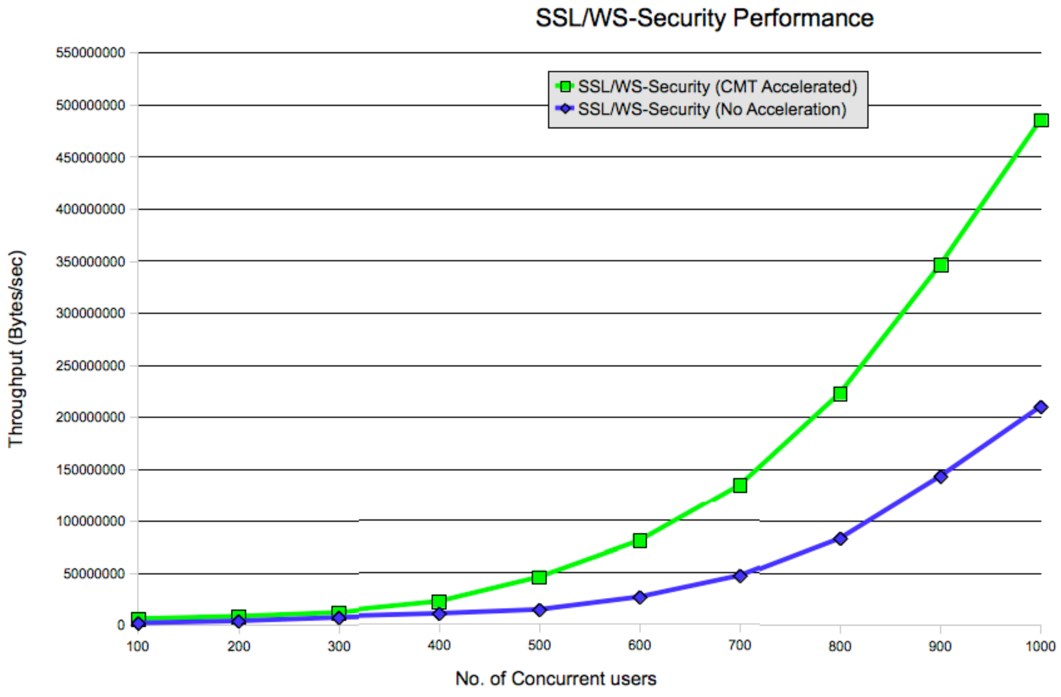


Figure 15: SSL/WS-Security Combined Performance - Effect of Acceleration

The results showed enabling on-chip acceleration for combined SSL/WS-Security scenarios solidly delivered more than **200% of overall application performance gain including both SSL and WS-Security operations** in comparison with SSL/WS-Security operations using no acceleration.

Conclusions

This whitepaper presented on the on-chip cryptographic accelerator features of Oracle SPARC Enterprise T-Series servers that supports Oracle Web Services Manager secured SOA applications and XML Web services. The paper unveiled the core mechanisms, configuration, deployment strategies and the role and relevance of using the Solaris Cryptographic Framework and Java Cryptographic Extensions based techniques. SSL and WS-Security has become the de-facto standard for delivering transport-layer security and message-layer security in Web applications and XML Web services. The SSL and WS-Security performance characteristics presented in this whitepaper clearly showed the compelling security performance aspects for adopting to on-chip cryptographic accelerator mechanisms of UltraSPARC T-series servers and integration of Oracle Web Services Manager for securing SOA and XML Web services.

To summarize, Oracle's Oracle SPARC Enterprise T-Series servers and blades with chip multithreading technology (CMT) has proven demonstrating high performance security with consistent scalability for SOA and XML Web services while also delivering reductions in space, power consumption, and cost.

Further References

1) Oracle SPARC Enterprise T-Series servers
   - http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/t-series/index.html
2) Oracle Fusion Middleware 11g R1 Documentation
   - http://www.oracle.com/technology/documentation/middleware.html
3) Securing Oracle WebLogic Server
   - http://download-llnw.oracle.com/docs/cd/E12840_01/wls/docs103/secmanage/
4) Deploying Web Services Applications
   - http://download.oracle.com/docs/cd/E12839_01/web.1111/b32511/deploy.htm
5) Attaching Policies to Web Services
   - http://download.oracle.com/docs/cd/E12839_01/web.1111/b32511/attaching.htm
6) Java PKCS#11 Reference Guide
   - http://download.oracle.com/javase/6/docs/technotes/guides/security/p11guide.html
7) Oracle Solaris 10 Security for System Administrators
   - http://www.oracle.com/technetwork/server-storage/solaris/overview/security-163473.html
8) Oracle Solaris Cryptographic Framework – Overview
   - http://docs.sun.com/app/docs/doc/816-4557/scf-1?a=view

Oracle is committed to developing practices and products that help protect the environment

High Performance Security For SOA and XML Web Services using Oracle Web Services Manager and Oracle SPARC Enterprise T-Series Servers
September 2010
Authors: Ramesh Nagappan, Vikas Jain, and Nitin Handa]

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

SOFTWARE. HARDWARE. COMPLETE.