



An Oracle White Paper
January 2011

High Performance Security For Oracle WebLogic Applications Using Oracle SPARC Enterprise T-series Server

Introduction	1
Target Audience and Assumed Knowledge	1
Hardware and Software Environments	2
Role and Relevance of Oracle SPARC Enterprise T-series Servers	3
Integrated Cryptographic Acceleration.....	5
Role Of Solaris Cryptographic Framework	7
Solaris Cryptographic Framework Components	7
Cryptographic Acceleration for Oracle WebLogic Applications.....	9
Applied Security Mechanisms and Usage Scenarios	10
Transport-Layer Security Acceleration using Solaris KSSL.....	10
Configuring KSSL for WebLogic SSL Acceleration.....	11
Transport and Message Layer Security Acceleration using Sun JCE	14
Accelerating SSL using SunPKCS11 Provider	15
Accelerating WS-Security using SunPKCS11 Provider	17
Examining On-Chip Cryptographic Accelerator Operations	19
Performance Characteristics.....	19
Scenario 1: SSL Performance	19
Scenario 2: Overall Application Performance	21
Conclusions	22
Further References	24

Introduction

This document details the high performance security strategies for Oracle WebLogic server based applications and XML Web services using the on-chip cryptographic acceleration capabilities of Oracle SPARC Enterprise T-Series Servers. This documents presents the technical pre-requisites, configuration, deployment and verification guidelines for Oracle WebLogic server, Java runtime environment and Oracle Solaris Cryptographic Framework for supporting the cryptographic operations involved with encryption/decryption, digital signature, key management functions of SSL and WS-Security application scenarios.

Target Audience and Assumed Knowledge

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

Hardware and Software Environments

The Oracle WebLogic 11g and its 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 WEBLOGIC VERSION
Oracle Solaris 10 Update 8 and Solaris 10 Update 9	UltraSPARC T3 and T2 Plus servers	Oracle WebLogic 11g Suite 10.3.3

TABLE 1: SUPPORTED HARDWARE AND SOFTWARE ENVIRONMENTS

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 Sun SPARC Enterprise T-series servers with CoolThreads™ technology. The following figure shows the single socket Oracle SPARC Enterprise T3-1 server (Figure 1).

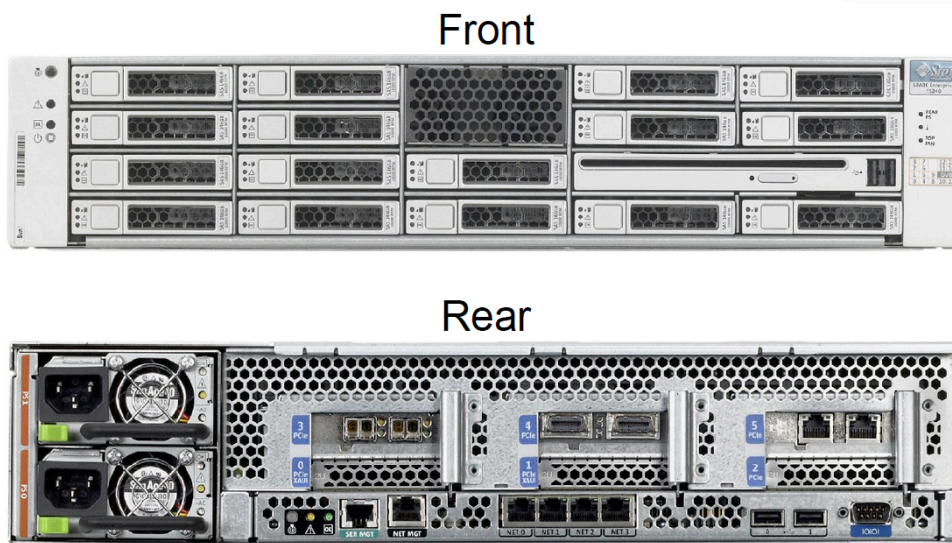


Figure 1: The Oracle SPARC Enterprise T3-1 Server

The Sun SPARC Enterprise T-series servers are equipped with one of the UltraSPARC® T1, T2, T2 Plus and T3 processors. Each T-series processor 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 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.

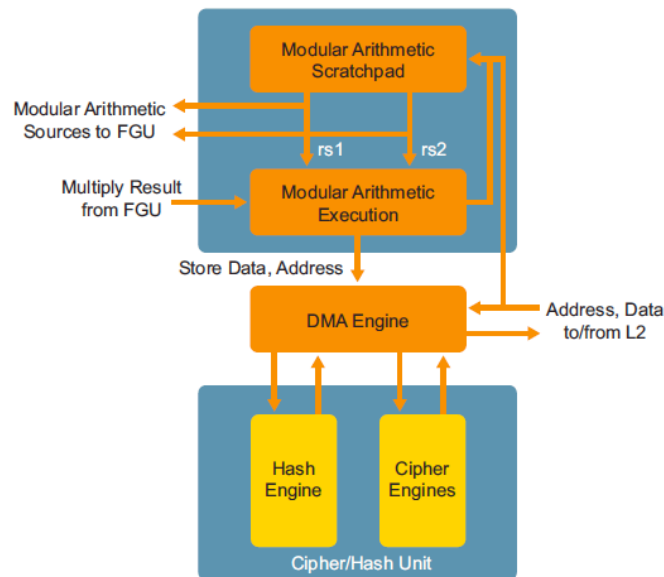


Figure 2: Oracle SPARC Enterprise T-Series processor - Modular Arithmetic Unit

The MAU on each T-series processor includes a dedicated Cipher and Hash engine that facilitates performing the cryptographic operations and it runs 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).

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
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	PKCS#11	PKCS#11
RANDOM NUMBER GENERATION	-	N2RNG	N2RNG

Table 3: Oracle SPARC Enterprise T-Series Processors Comparison

In practice, the Oracle Solaris Cryptographic Framework acts as the intermediary that allow user-level applications to off-load cryptographic operations to take advantage of NCP and N2CP accelerator drivers for performing on-chip cryptographic acceleration. As a result, deploying Sun SPARC Enterprise T-Series servers provides an application-transparent facility to access the hardware accelerator for performing cryptographic acceleration - without adding any new code to the application at all.

Role Of Solaris Cryptographic Framework

The Oracle Solaris Cryptographic Framework library plays a vital role for providing application-level access to NCP and N2CP accelerators. The framework 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. It 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).

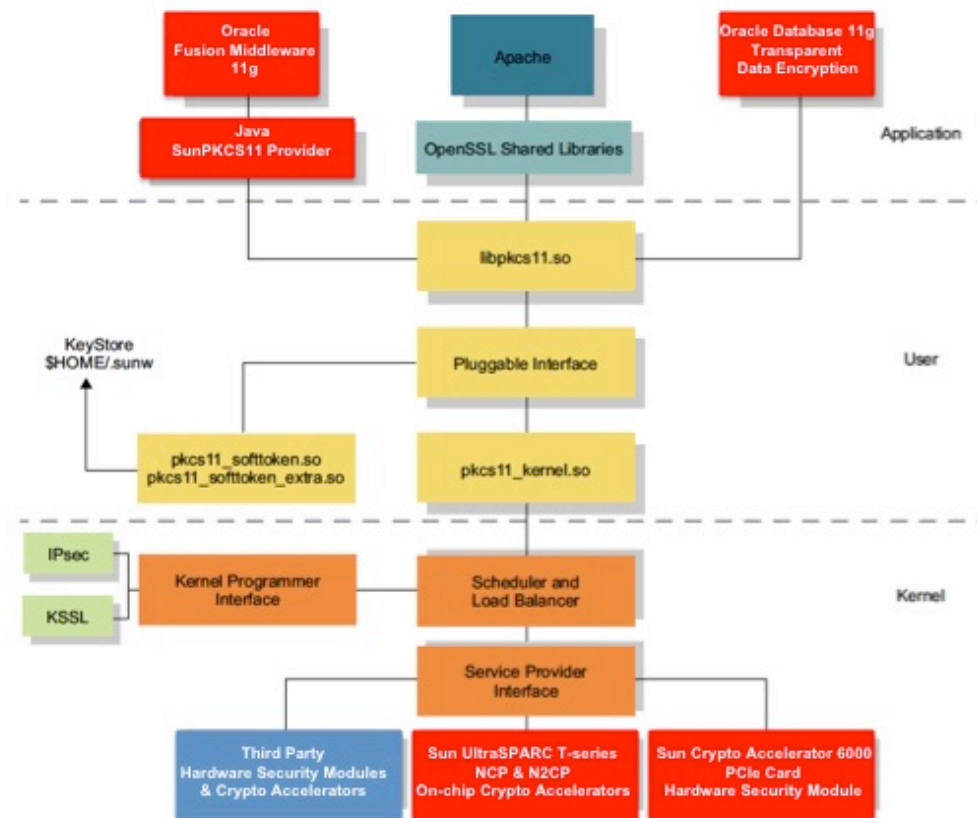


Figure 3: Solaris Cryptographic Framework and its core components

Solaris Cryptographic Framework Components

Figure 3 describes the Solaris Cryptographic Framework. Consisting of a user-level framework, and a kernel-level framework, it 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. The userland part of the framework (**libpkcs11**) provides a **metaslot** that represents a virtual provider aggregating all available cryptographic algorithms 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
```

On Sun servers with UltraSPARC T2 or SPARC T3 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 -v` command to display the list of available cryptographic providers.
- 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 on Sun servers with UltraSPARC T2 or SPARC T3 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 (eg. `AES_CBC_PAD`) on an installed cryptographic provider 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 Oracle WebLogic Applications

The Oracle WebLogic server 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

The Oracle WebLogic server applications can offload select cryptographic operations for 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, DES) and message digest (ex. SHA-1, SHA-2, MD5) functions intended for supporting XML encryption, XML digital signature and related cryptographic operations.

The Oracle SPARC Enterprise T-Series processor's on-chip cryptographic acceleration capabilities can be accessed in a variety of ways by the Oracle WebLogic server deployment, depending on the applied security scenarios and its requirements. The Oracle Solaris kernel modules - Kernel SSL (KSSL) and IPSec provides kernel-level security mechanisms for delivering SSL and IPSec functions and supports using underlying hardware cryptographic accelerators. In addition, the availability of PKCS#11 interfaces via Java Cryptographic Extension (JCE) framework also enables the Oracle WebLogic server deployed service-oriented architectures and Java EE applications can take advantage of hardware accelerators by off-loading SSL and WS-Security based cryptographic workloads.

Transport-Layer Security Acceleration 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.

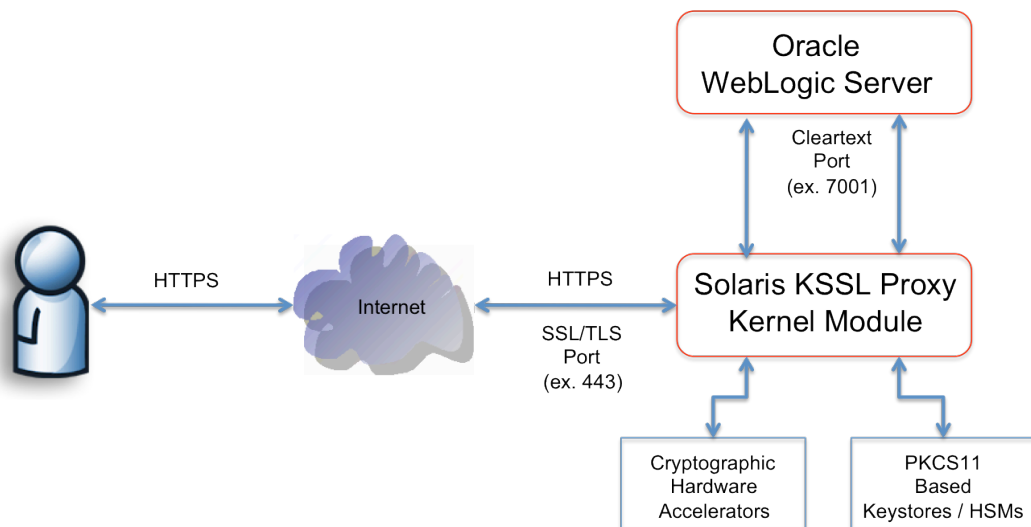


Figure 4: Using Solaris KSSL Proxy for WebLogic SSL

Configuring KSSL for WebLogic SSL Acceleration

Using the KSSL kernel module as an SSL proxy (Figure 4) requires obtaining and installing a certificate from a Certificate Authority¹. 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

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

```
# /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
```

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

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

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

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

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

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

```
# svcs -a | grep "kssl"
```

- Alternatively, use the Solaris `'netstat -an'` command to verify KSSL is listening on port 443.

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

- 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 sofftoken). The following command and options are typically used for configuring PKCS#11 based keystores.

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

- Configure a "Sun Software PKCS#11 Sofftoken" keystore using `pktool` utility.

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

- 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 Softtoken"
```

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 Softtoken"
    -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 `ksslconfig` command must use `-c` option followed by the required list of `<ciphersuites>` in a sorted order. For example:

```
# ksslcfg create -c rsa_3des_edc_cbc_sha,rsa_des_cbc_sha
-f pem -i /etc/pki/mySSLCerts.pem
-x 7001 -p /etc/pki/passwordfile serverhostname 443
```

Transport and Message Layer Security Acceleration 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 HSMs. The SunPKCS11 provider is a Java based PKCS#11 implementation that integrates with underlying PKCS#11 implementations provided by the Solaris Cryptographic Framework 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 enable or disable the PKCS#11 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 (Figure 5).



Figure 5: Oracle WebLogic Server using Oracle SPARC Enterprise T-Series Servers

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 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, 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 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
```

Accelerating WS-Security using SunPKCS11 Provider

WS-Security plays a critical role in providing message-level security of XML Web services by ensuring confidentiality, integrity and access control of SOAP messages. The Oracle WebLogic servers relies on the JCE provider for supporting cryptographic operations involved with message-level security of XML Web services and the SunPKCS11 provider for facilitating cryptographic acceleration associated of encryption, signature and message digest operations.

WebLogic server strongly recommends the use of WS-SecurityPolicy standard and it provides predefined WS-SecurityPolicy files to specify message-level security requirements. The WS-SecurityPolicy describes the message-level security requirements of the SOAP messages and how the requested operation should be digitally signed or encrypted using relevant cryptographic algorithm suites defined in the policy. The exposed Web service makes the associates security policy available via the WSDL.

The following steps explain how to configure WebLogic Server for XML Web services security operations and its acceleration using the on-chip cryptographic acceleration capabilities of Oracle SPARC Enterprise T-Series servers.

1. It is assumed that the Web service created is deployed as a JWS file that implement using Oracle WebLogic Web service and/or JAX-WS APIs.
2. Update your JWS file, including WebLogic-specific `@Policy` and `@Policies` JWS annotations to specify the predefined policy files that represents WS-Policy and WS-SecurityPolicy definitions for performing the required WS-Security mechanisms.

```
@WebService(name="Simple",
targetNamespace="http://oracle.org")
@WLHttpTransport(contextPath="/wssp12/wss10",
serviceUri="UsernameTokenPlainX509SignAndEncrypt")
@Policy(uri="policy:Wssp1.2-2007-Wss1.0-UsernameToken-Plain-
X509-Basic256.xml")

public class UsernameTokenPlainX509SignAndEncrypt {
@WebMethod
@Policies({
@Policy(uri="policy:Wssp1.2-2007-SignBody.xml"),
@Policy(uri="policy:Wssp1.2-2007-EncryptBody.xml")})
public String echo(String s)
{
return s;
}
```

The above WS-Policy identifies the WS-SecurityPolicy specifying the service authenticates the client using a username token and both the request and response messages are signed and encrypted with X509 certificates. The algorithm suite used will be **Basic256** that represents AES-256 algorithm for bulk encryption. For testing purposes, the Web services runtime uses the out-of-the-box private key and X.509 certificate pairs, store in the default keystore (bundled with the WebLogic server), for its encryption and digital signatures. In case of production, it is highly recommended to acquire keys and certificates from a Certificate

Authority.

3. After including the policies, recompile and deploy the Web service.
4. It is assumed that you have created the client application that invokes a deployed Web service, and that you want to update it by associating a client-side policy file. Create the client-side security policy files and save them in a location accessible by the client application. Typically, the security policy files are the same as those configured for the server-side Web service you are invoking, but because the server-side files are not exposed to the client runtime, the client application must load its own local copies. You may choose to use the `weblogic.jws.jaxws.ClientPolicyFeature` class in the client application to override the effective policy defined for a service.
5. Make sure to enable the cryptographic mechanisms specified in the WS-SecurityPolicy algorithm suite by include them in the list of `enabledMechanisms` or by removing them from `disabledMechanisms` of the Java SunPKCS11 configuration file. If the specified algorithm suite is Basic256, it uses `Aes256` encryption, `Sha1` for message digest, and `Rsa-oaep-mgf1p` for key wrap.

For example, you may remove the bulk encryption and message digest algorithms in the `disabledMechanisms` list that forces those operations (ex. AES, SHA-1) performed by the N2CP accelerator.

6. Update your Java client application to load the client-side policy files and rebuild/redeploy the client application. If the deployed client is a Web application, restart the Oracle WebLogic server instance before running the tests.
7. In some cases, the SunPKCS11 provider may not be 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 either enabled delegation of the `CKM_SHA1_RSA_PKCS` mechanism in the `PKCS#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 and it is 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

nep/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 nep/0 (since the last reboot), run the Oracle Solaris `kstat` command:

```
# kstat -n nep0 -s rsaprivate
```

```
# kstat -n nep0 -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: SSL Performance

The following graph (Figure 6) represents the SSL operations performance characteristics of the following WebLogic SSL scenarios.

- a. Use KSSL as SSL proxy and Sun Software PKCS#11 Softtoken based keystore – By default on Oracle SPARC Enterprise T-Series servers, KSSL automatically use NCP and N2CP for cryptographic acceleration.
- b. WebLogic Managed Server SSL listener configured with JKS keystore and use SunPKCS11 provider for enabling CMT based cryptographic acceleration.
- c. WebLogic Managed Server configured to use SSL (With no SunPKCS11 provider for cryptographic 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,

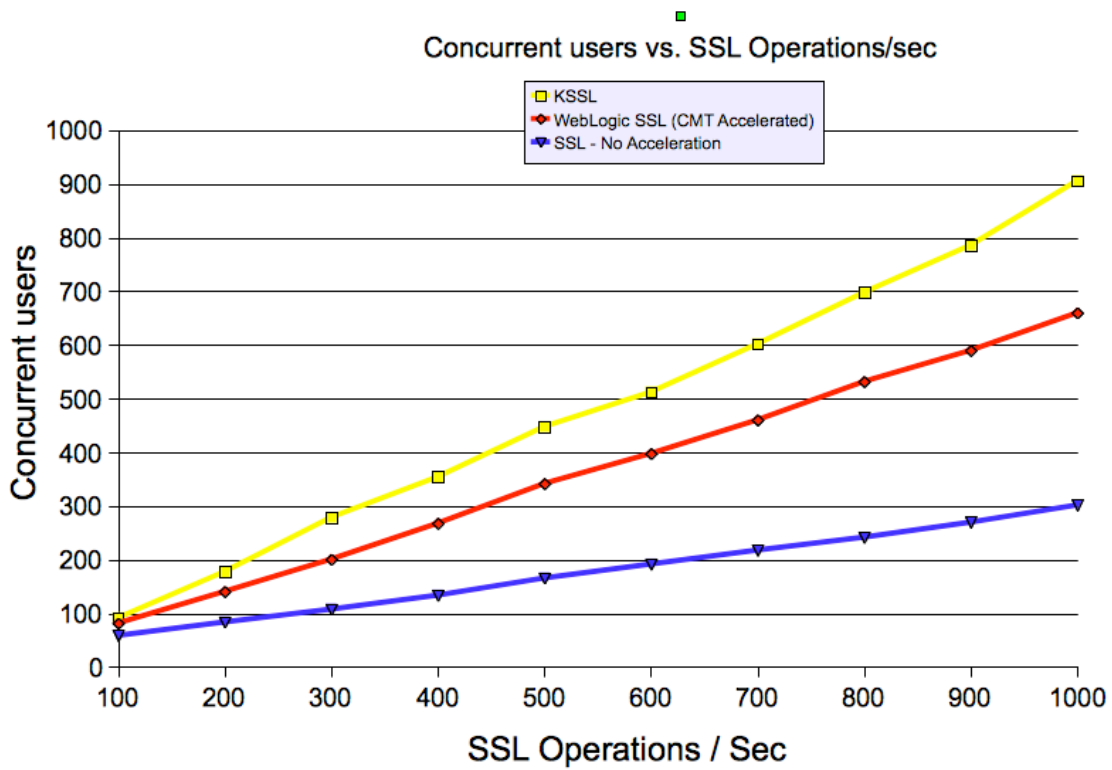


Figure 6: WebLogic SSL - SSL Performance Characteristics

Machine used:
T3 (Single socket), 64 Gb Memory

Software configuration:
Oracle WebLogic 10.3.3 (1 Admin server, 4 Managed servers)
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_edc_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: Overall Application Performance

The following graph (Figure 7) represents the overall performance characteristics of after and before using CMT based cryptographic acceleration and study the effect of cryptographic overheads on the application. The tests ran using the existing setup described in Scenario 1. Apache JMeter was used as the load driver for driving the workload, simulating 1000 concurrent users ramped up in increments of 10 users per minute until reaching 1000 users. Each user queried the web application as many times as possible per minute, clearing caches in between. Once 1000 concurrent users were reached, the workload was sustained for 10 minutes. The load test run captured numbers for three key aspects of any web transaction: Throughput (or Peak Transfer), Hits per second and Tests per minute. This Jmeter load test was not intended to push the upper limits of the server but rather to demonstrate the overheads of cryptography at a reasonable load and the effects of using hardware assisted cryptographic acceleration.

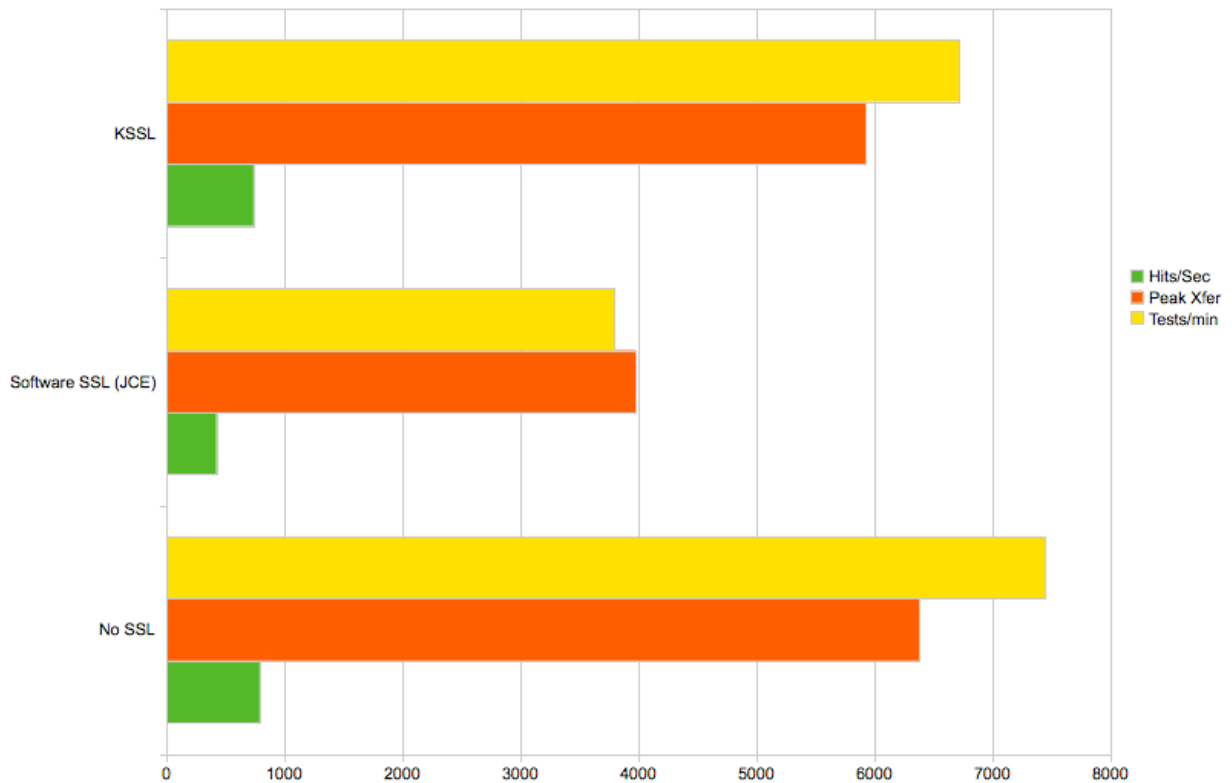


Figure 7: Overall Application Performance – Effect of Cryptographic Acceleration

The results showed only a minor difference due to SSL overhead between the unsecured application versus on-chip cryptographic accelerated solution, which yielded tangible, immediate and cost-efficient results in the form of faster secure transactions and better response times – all without adding any additional security equipment costs, changes in power usage profiles or elaborate system

configurations. Additionally, the results clarify the massive burden un-accelerated cryptographic workloads can have on a server.

Conclusions

This whitepaper presented on the on-chip cryptographic accelerator features of Oracle SPARC Enterprise T-Series servers that support securing Oracle WebLogic 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 for enabling hardware-assisted cryptographic acceleration. SSL has become the de-facto industry standard for delivering transport-layer security in Web applications and XML Web services. The SSL performance characteristics presented in this whitepaper clearly showed the compelling security performance benefits of adopting to T-series processor based cryptographic accelerator mechanisms for securing Web applications and XML Web services scenarios.

To summarize, the Oracle SPARC Enterprise T-Series servers and blades with Chip Multithreading Technology (CMT) has proven demonstrating high performance security with consistent scalability for Oracle WebLogic applications 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) Sun Cryptographic Accelerator 6000 PCIe Card Version 1.1 Documentation
 - <http://www.oracle.com/us/products/servers-storage/networking/031146.htm>
- 3) Securing Oracle WebLogic Server
 - http://download-llnw.oracle.com/docs/cd/E12840_01/wls/docs103/secmanage/
- 4) Java PKCS#11 Reference Guide
 - <http://download.oracle.com/javase/6/docs/technotes/guides/security/p11guide.html>
- 5) Oracle Solaris 10 Security for System Administrators
 - <http://www.oracle.com/technetwork/server-storage/solaris/overview/security-163473.html>
- 6) Oracle Solaris Cryptographic Framework – Overview
 - <http://docs.sun.com/app/docs/doc/816-4557/scf-1?a=view>



High Performance Security for
Oracle WebLogic Applications
Using Oracle SPARC Enterprise
T-Series Servers
January 2011
Author: Ramesh Nagappan
Contributing Authors: Chad Prucha

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



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

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0110

SOFTWARE. HARDWARE. COMPLETE.