Secure XML Processing Using Chip Multi-Threaded Processors

Biswadeep Nag, Kim LiChong, Pallab Bhattacharya

Java Performance Engineering
Sun Microsystems, Inc.
http://java.sun.com/performance

TS-6264

Agenda

- The Typical CMT Architecture
- The XMLTest Benchmark
  - StAX and JAXB
  - CMT vs. SMP Scalability
  - JVM Tuning
  - OS Tuning
- XML Signature Performance
  - Introduction to XML Signatures
  - Accelerating Signatures on UltraSPARC-T1

Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- JVM Tuning
- OS Tuning

XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

Why CMT?

- Processor clocks getting faster quickly
  - But memory speeds increasing slowly
- Processor pipeline spends most of the cycles
  - Waiting for loads/stores to memory
  - Burning power

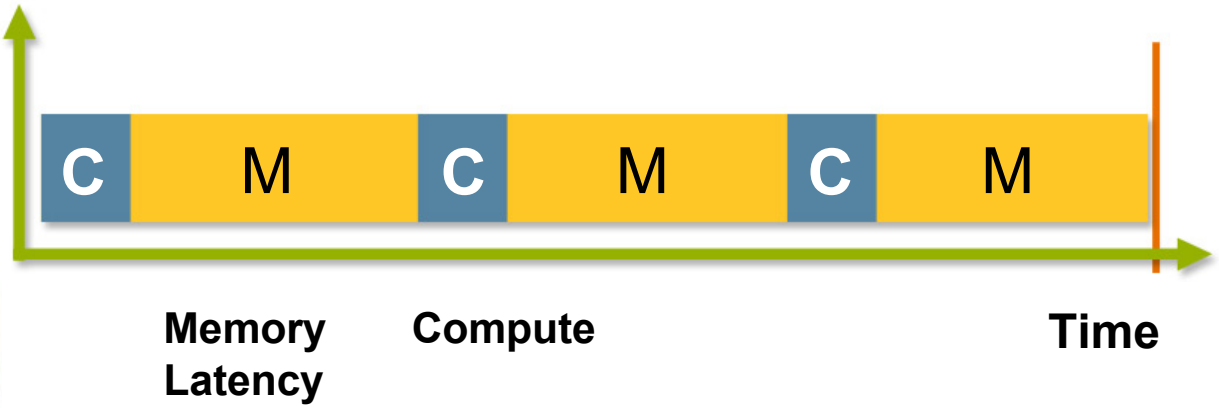
Single Threading

Up to 75% Cycles Waiting for Memory

HURRY UP AND WAIT!

Single Threaded Performance

Typical Processor Utilization:15–25%

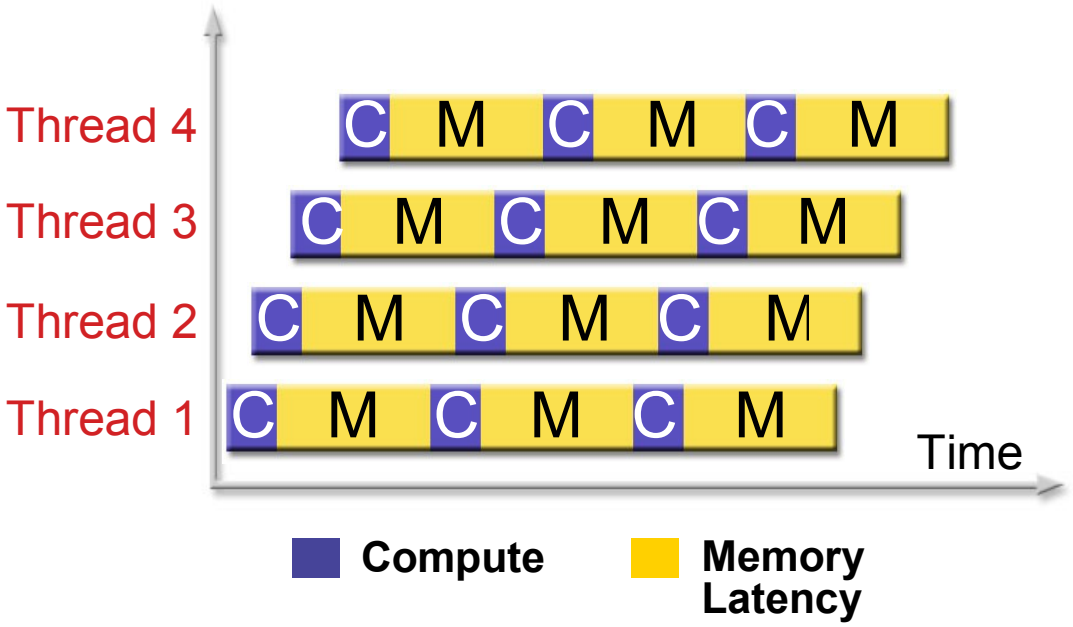


Hardware Multi-Threading

- Multiple application threads execute simultaneously
- Multiple hardware contexts
- Efficient switching between hardware threads
- To the operating system each hardware thread is a CPU!
- Same principle as multi-programming!

The Power of CMT

Processor Utilization:Up to 85%

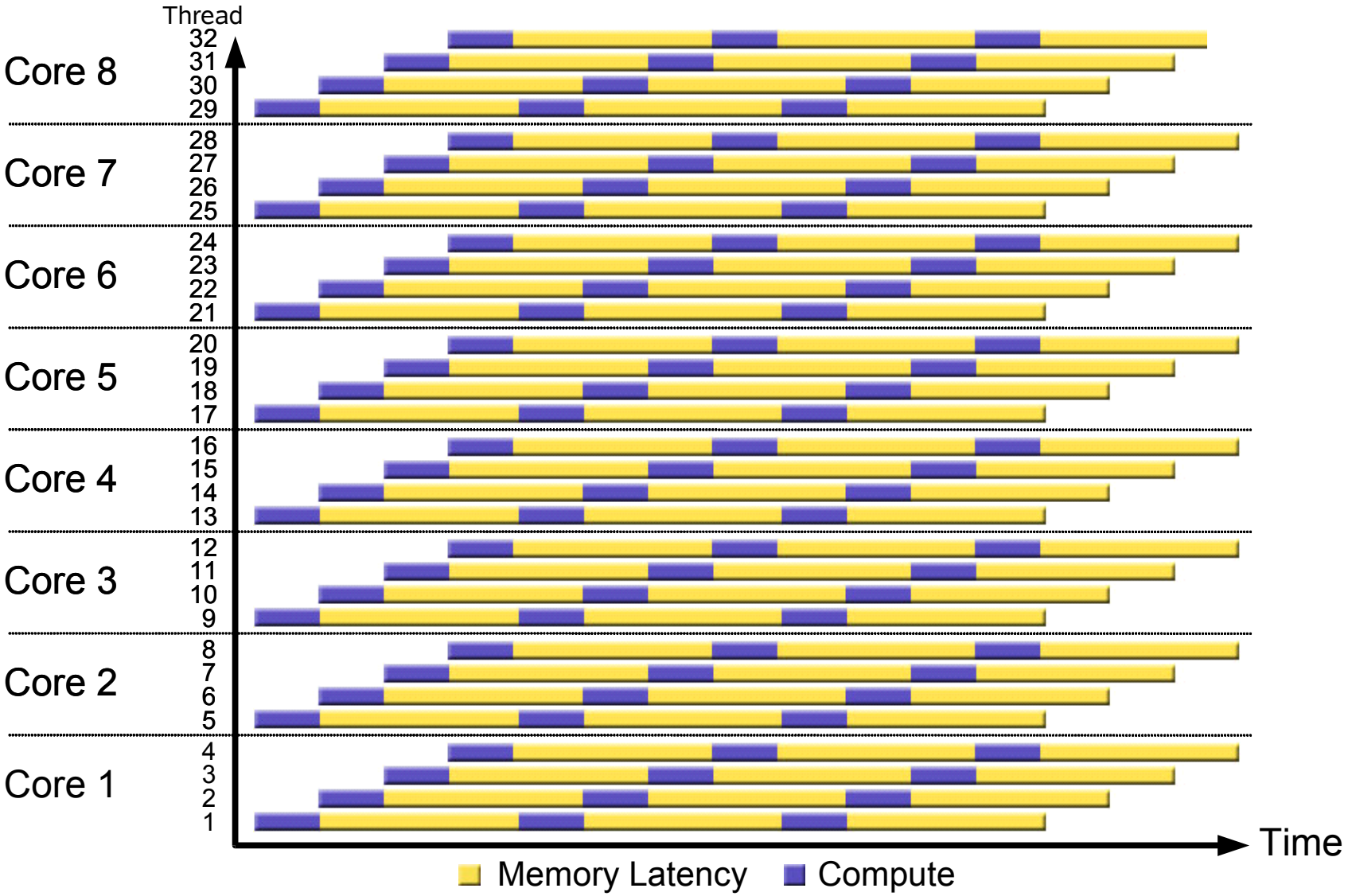


Chip Multi-Threaded (CMT) Performance

Chip Multi-Threading

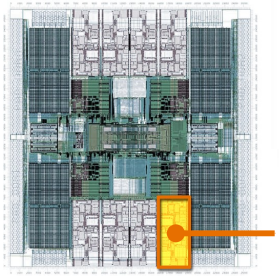
- Replicate this model several times
- Multiple processors in a single chip
- High processor throughput
- Requires heavily multi-threaded applications
- Silicon area used for multiple processing elements
- Simpler processor design
  - Relatively small caches
  - In-order pipelines

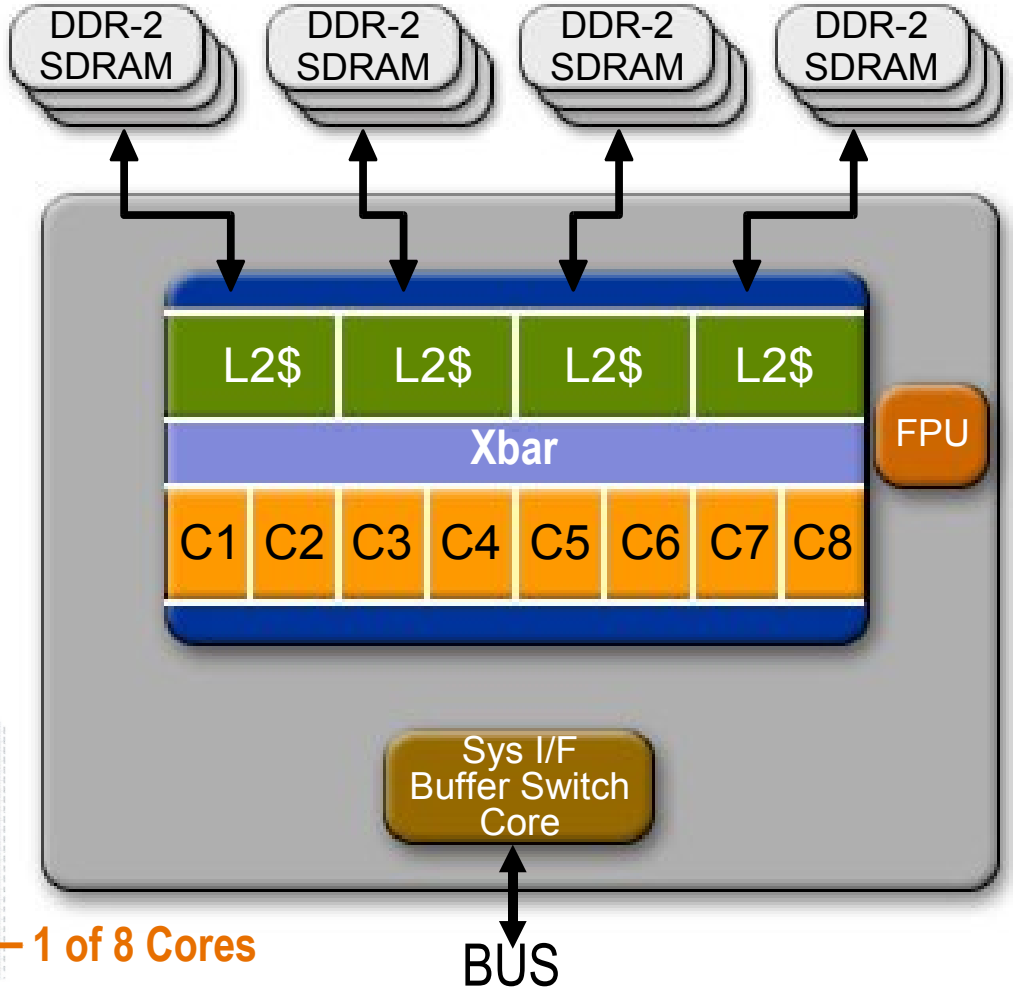
CMT—Multiple Multi-Threaded Cores



Introducing the UltraSPARC-T1

- SPARC® V9 implementation
- Up to eight 4-way multi-threaded cores for up to 32 simultaneous threads
- All cores connected through a 90GB/sec crossbar switch
- High-bandwidth 4-way shared 3MB Level-2 cache on chip
- 4 DDR2 channels
- Power: < 70W !
- ~ 300M transistors
- 378 sq. mm die





Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- JVM Tuning
- OS Tuning

XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

Goal of This Section

What Java options increase StAX and JAXB performance in CMT systems?

How does CMT performance compare to SMP for JAXB and StAX?

Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- JVM Tuning
- OS Tuning

XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

XMLTest

XML Processing Benchmark

- Standalone multi-threaded Java-based program
- No File I/O—XML is read from memory streams
- No think time
- Measures the throughput of a system processing XML documents
- Throughput = Average number of XML transactions executed per second
- Transaction is the time taken to parse through a document

XMLTest

- Supports
  - Various document sizes and schemas
  - Fast Infoset for SAX parsing
  - Java XML signature generation and validation
  - Canonicalization
- https://xmltest.dev.java.net

Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- JVM Tuning
- OS Tuning

XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

What Is StAX?

Streaming API for XML

- Streaming API for XML (StAX), a bi-directional API for reading and writing XML
- Specified by JSR 173
- “Pull parsing”—Developer pulls next XML construct in the document
- Sun’s implementation is Sun Java Streaming XML Parser (SJSXP)

What Is JAXB?

Java Architecture for XML Binding

- Provides an API and tool that allow automatic two-way mapping between XML documents and Java objects
- The JAXB compiler can generate a set of Java based classes from XML
- Developers can build applications and do not to write any logic to process XML elements

How Is the XML Being Processed?

StAX Parsing

- Measuring parsing without serialization

JAXB Binding

- Measuring unmarshalling operation
  - Building Java-based object tree in memory

Agenda

- The Typical CMT Architecture
- **The XMLTest Benchmark**
  - StAX and JAXB
  - CMT vs. SMP Scalability
  - JVM Tuning
  - OS Tuning
- XML Signature Performance
  - Introduction to XML Signatures
  - Accelerating Signatures on UltraSPARC-T1

Benchmark Characteristics

Software

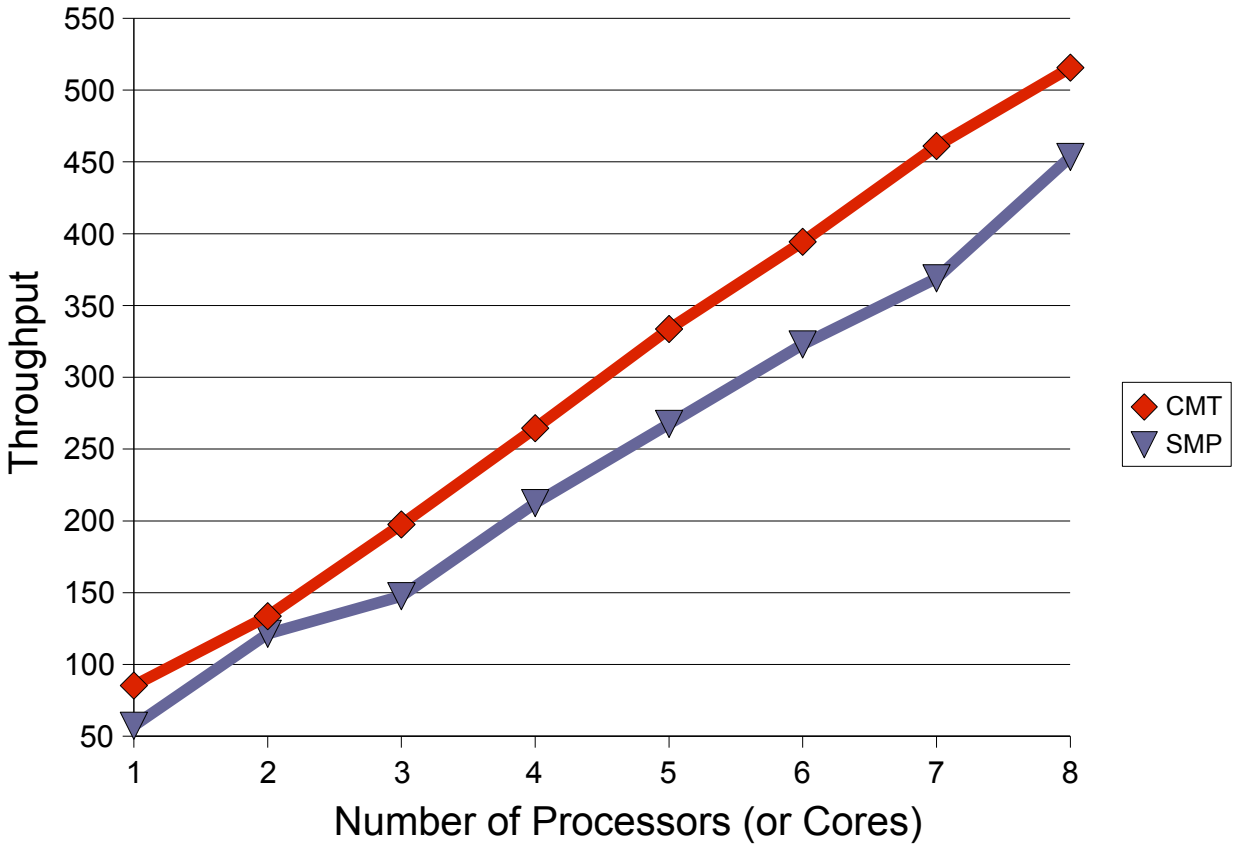
- Solaris 10
- Java Web Services Developer Pack (Java WSDP) 2.0
- Java SE 1.5.0_06

Hardware

- UltraSPARC T2000 UltraSPARC-T1 1200 MHz
  - 32 GB Memory
- Sun Fire 880 UltraSPARC-III+ 1200 MHz
  - 16 GB Memory

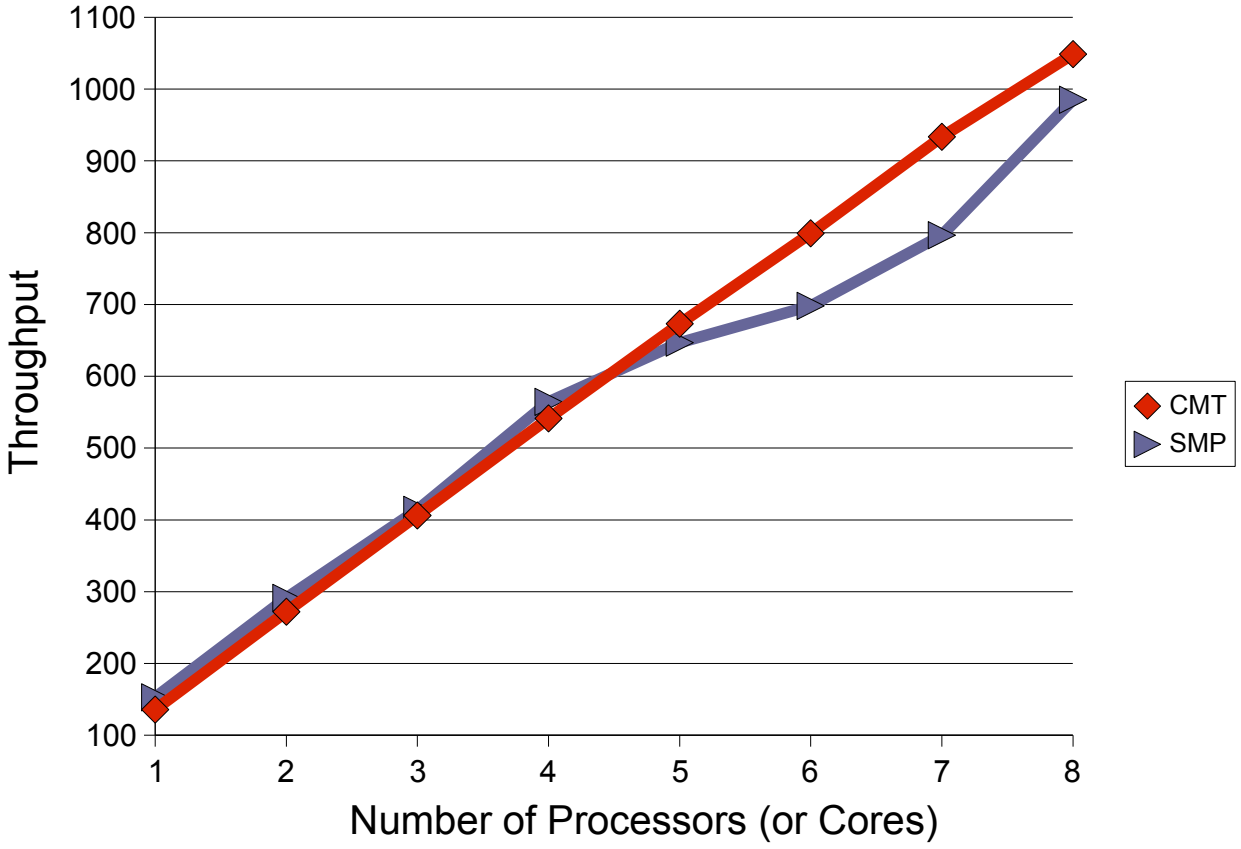
CMT vs. SMP Throughput

JAXB 2 With 100 KB XML Document



CMT vs. SMP Throughput

SJSXP With 100 KB XML Document



Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- JVM Tuning
- OS Tuning

XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

Java VM Tunings

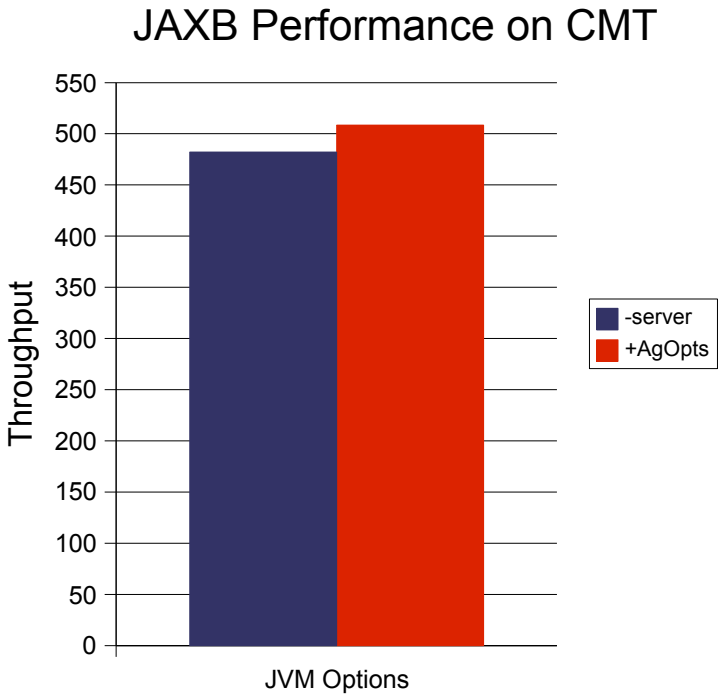
Which Ones Did We Try?

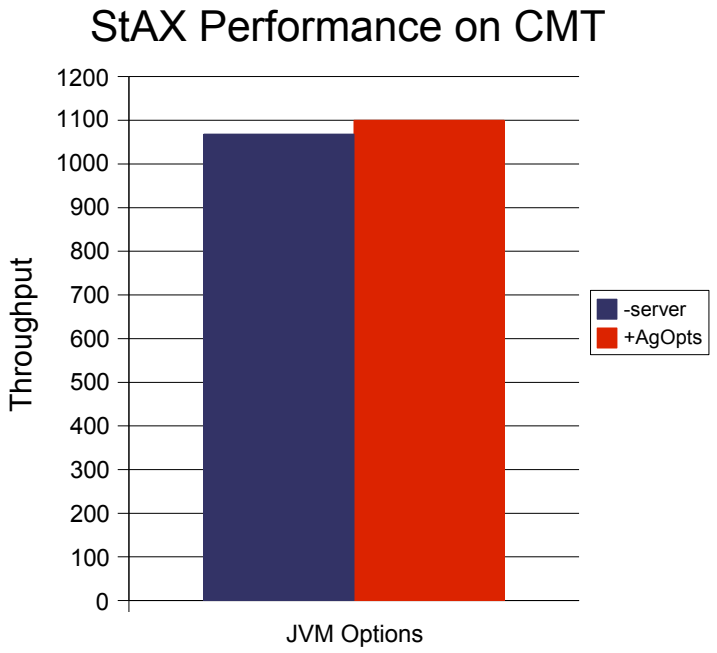
- -XX:+UseBiasedLocking
- -XX:+UseParallelGC
- -XX:+UseParallelGCThreads=<n>
- -XX:LargePageSizeInBytes=256m
- -Xmx -Xms -Xmn -Xss
- -XX:UseParallelOldGC
- -XX:+AggressiveOpts

And the Winner Is…

Modest Improvement

- -Xmx3550m -Xms3550m -Xmn2g -Xss128k -XX:+UseParallelGC -XX:ParallelGCThreads=8 -XX:+UseParallelOldGC -XX:+AggressiveOpts
- % Improvement apparent only at 8 cores (32 threads)
- Scores were not significantly different for other tunings for JAXB and StAX
- No major GC collection occurs during benchmark, similar number of minor GC collections





5 % Improvement

p<0.05
T Value = -11.533
DF: 18

3 % Improvement

p<0.05
T Value = -4.66
DF: 18

Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- JVM Tuning
- OS Tuning

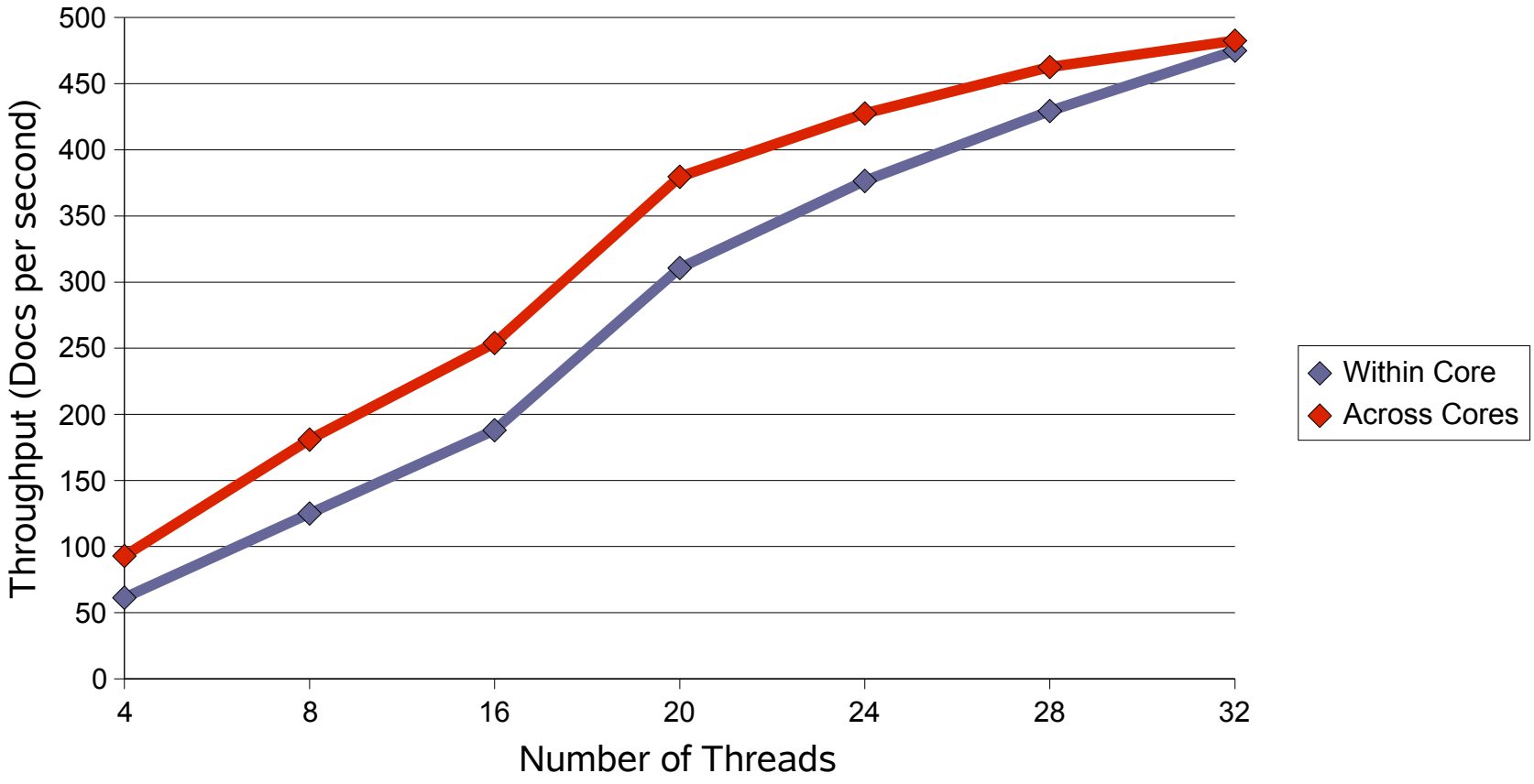
XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

CMT Scheduling

- Resources shared by hardware threads within a core
  - I-cache
  - D-cache
  - Integer unit
- Better to schedule application threads first across hardware threads in different processor cores
- Scheduling threads across cores
  - Higher throughput for lower thread counts
  - Better core utilization

CMT Scheduling and Scaling



Page Coloring

- L2 cache is shared by threads across all cores
  - May create conflicts for some cache lines
- `set consistent_coloring=2` in /etc/system
- However no effect on Java code/XML processing

Large Page Sizes

- More pressure on TLB entries because of large number of threads
- Using large pages requires fewer TLB entries
- Can produce undesired effects if large pages used for wrong memory segment
  - e.g., 256MB instead of 4MB pages for Java heap
- Defaults out-of-the-box work well

Process Maps (pmap -xs <pid>)

Default

Kbytes	Pgsz Mode	Mapped File
815104	4M rwx--	[anon]
12288	4M rwx--	[anon]
3840	64K rwx--	[heap]
28672	4M rwx--	[heap]

-XX:LargePageSizeInBytes=268435456

Kbytes	Pgsz Mode	Mapped File
786432	256M rwx--	[anon]
262144	256M rwx--	[anon]
3840	64K rwx--	[heap]
36864	4M rwx--	[heap]

Large Page Results

	Default (4 MB)	Large Page (256 MB)
TLB Miss Time	5.10%	21.70%
Throughput	471	418

Summary

- No need to change code to take advantage of CMT
- Performance improvement can differ from application to application as compared to SMP
- -XX:+AggressiveOpts gives a 5% improvement in JAXB 2, and 3% for StAX at full core utilization
- Most out of the box defaults for Solaris works well
- No need to tweak system parameters

Agenda

The Typical CMT Architecture

The XMLTest Benchmark

- StAX and JAXB
- CMT vs. SMP Scalability
- Java VM Tuning
- OS Tuning

XML Signature Performance

- Introduction to XML Signatures
- Accelerating Signatures on UltraSPARC-T1

Goal of This Section

Learn about how to accelerate the Java-based XML Digital Signature performance using Cryptographic Hardware Accelerators

Agenda of This Section

- Introduction to XML Signatures
- Accelerating Java XML Signature Performance
- Crypto Acceleration on UltraSPARC T1

Agenda of This Section

- Introduction to XML Signatures
- Accelerating Java XML Signature Performance
- Crypto Acceleration on UltraSPARC T1

Introduction to XML Signatures

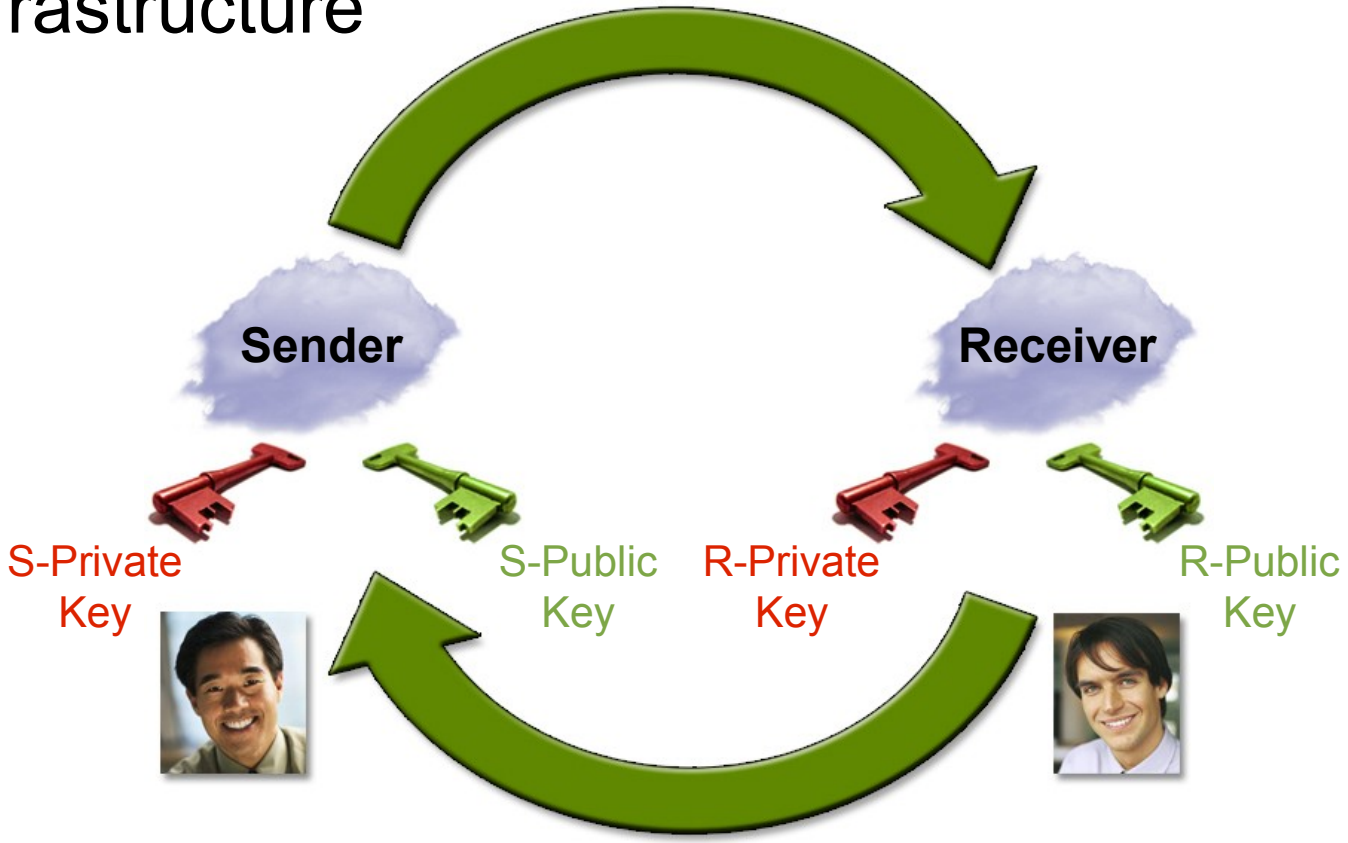
Introduction

- Digital Signatures are electronic messages with a mechanism analogous to signatures in the paper world
- Digital Signatures provide a means for an entity to bind its identity to a piece of information
- Digital Signatures ensures that the security requirements (end-point authentication, message integrity and non-repudiation) required for exchanging electronic messages are met

Introduction to XML Signatures

Introduction

- Based on the public key cryptography infrastructure



Introduction to XML Signatures

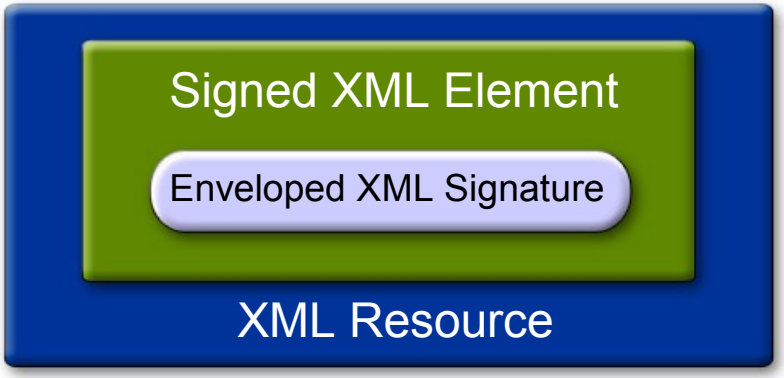
Introduction

- XML Digital Signatures will enable a sender to cryptographically sign data, which can then be used as the authentication credentials or a way to check the data integrity
- XML Signatures can be applied to any XML Resource, such as XML, an HTML page, binary-encoded data such as a GIF and XML-encoded data
- XML Digital Signature provides the flexibility to Sign only specific portions of the XML document

Introduction to XML Signatures

Introduction

- Classified as enveloped, enveloping, or detached
- An enveloped signature is the signature, applied over the XML content that contains the signature as an element



- The signature element is excluded from the calculation of the enveloped signature value

Introduction to Java Digital XML Signature API Specifications

Introduction

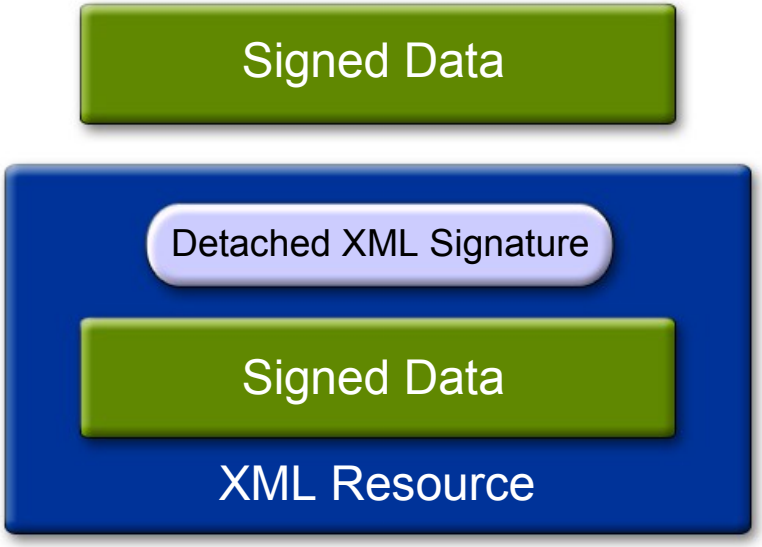
- An enveloping signature is the signature, applied over the content found within an object element of the signature itself

- The object (or its content) is identified via a Reference

Introduction to Java Digital XML Signature API Specifications

Introduction

- A detached signature is the signature applied over the content external to the signature element, and can be identified via a URI or a transform



Introduction to XML Signatures

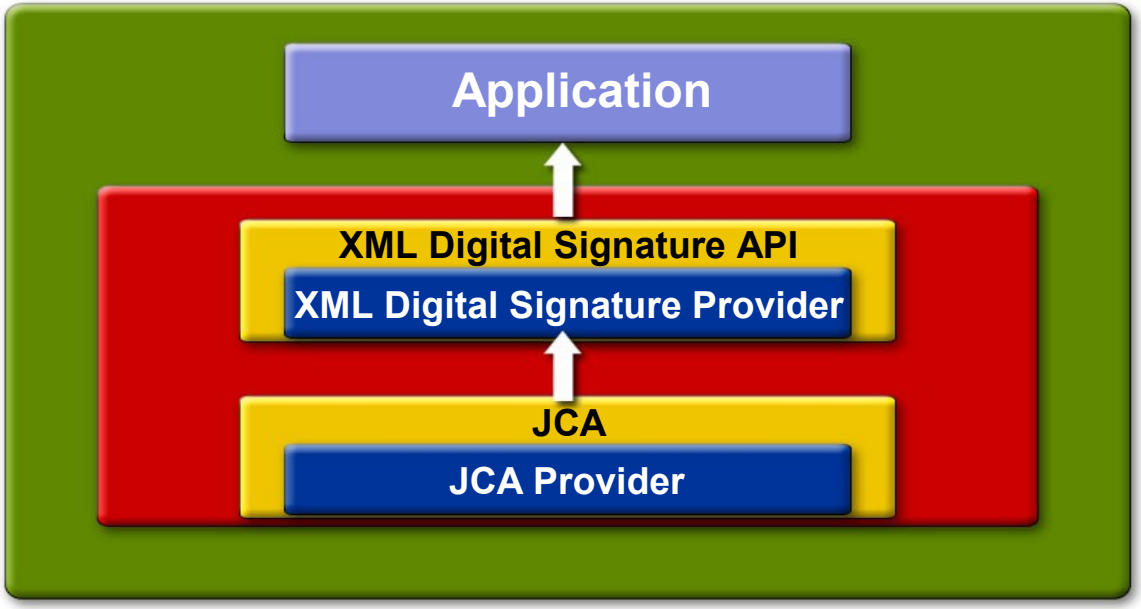
Java Digital XML Signature API Specifications

- Sun provides a standard set of Java API's to sign and verify XML and binary documents
  - These API's were defined under Java Community Process℠ along with various other organizations
  - Sun ships these API's with Java Web Services Developer Pack, Project GlassFish, and Java SE 6
- The Java XML Digital Signature Reference Implementation (JSR 105) from SUN is a pluggable framework built on the Java Cryptographic Architecture (JCA)

Introduction to XML Signatures

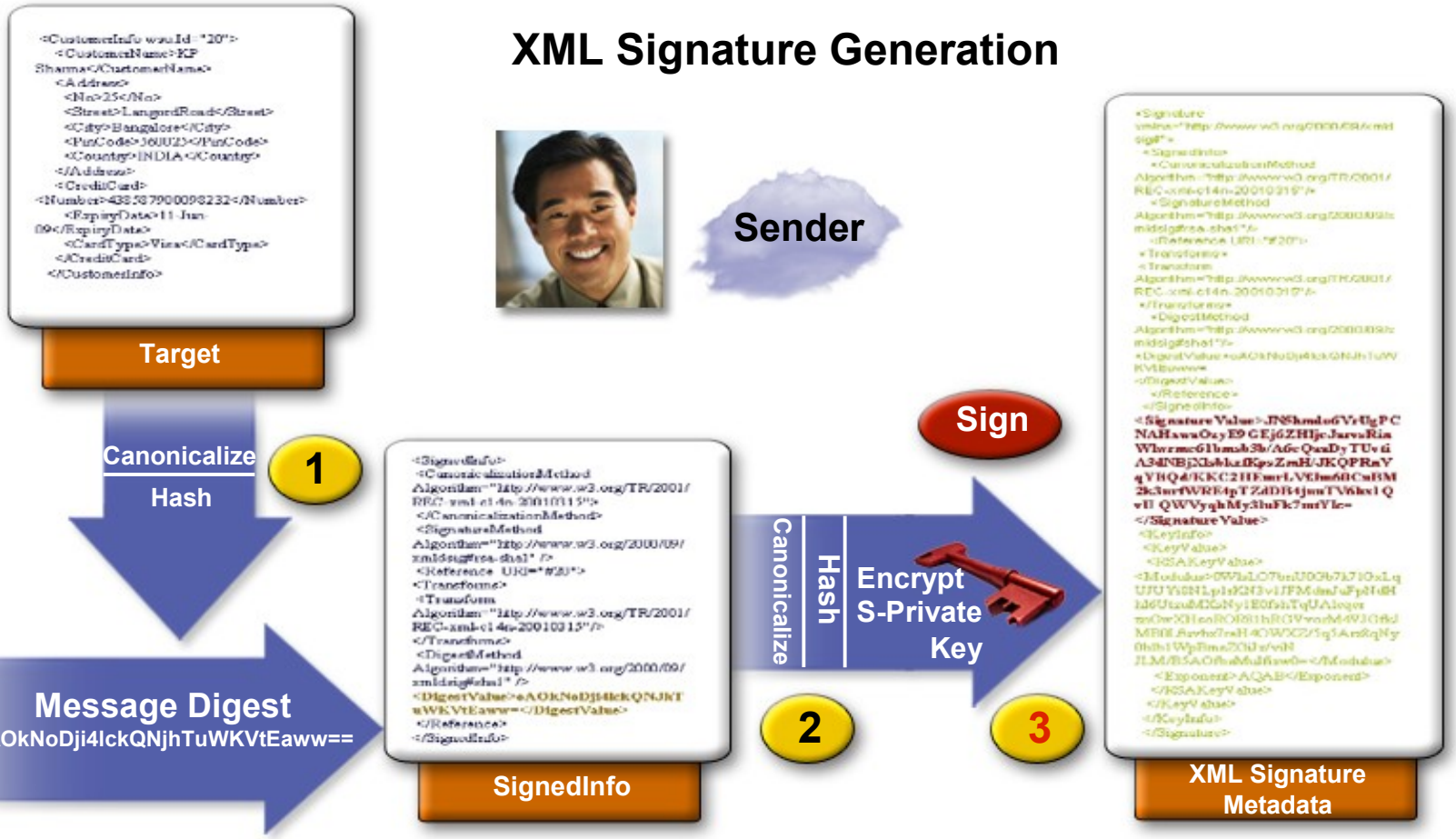
Java Digital XML Signature API Specifications

- JSR 105 provides support for various implementations of digital signature algorithms and transforms as specified by W3C XML Signature Syntax and processing specification



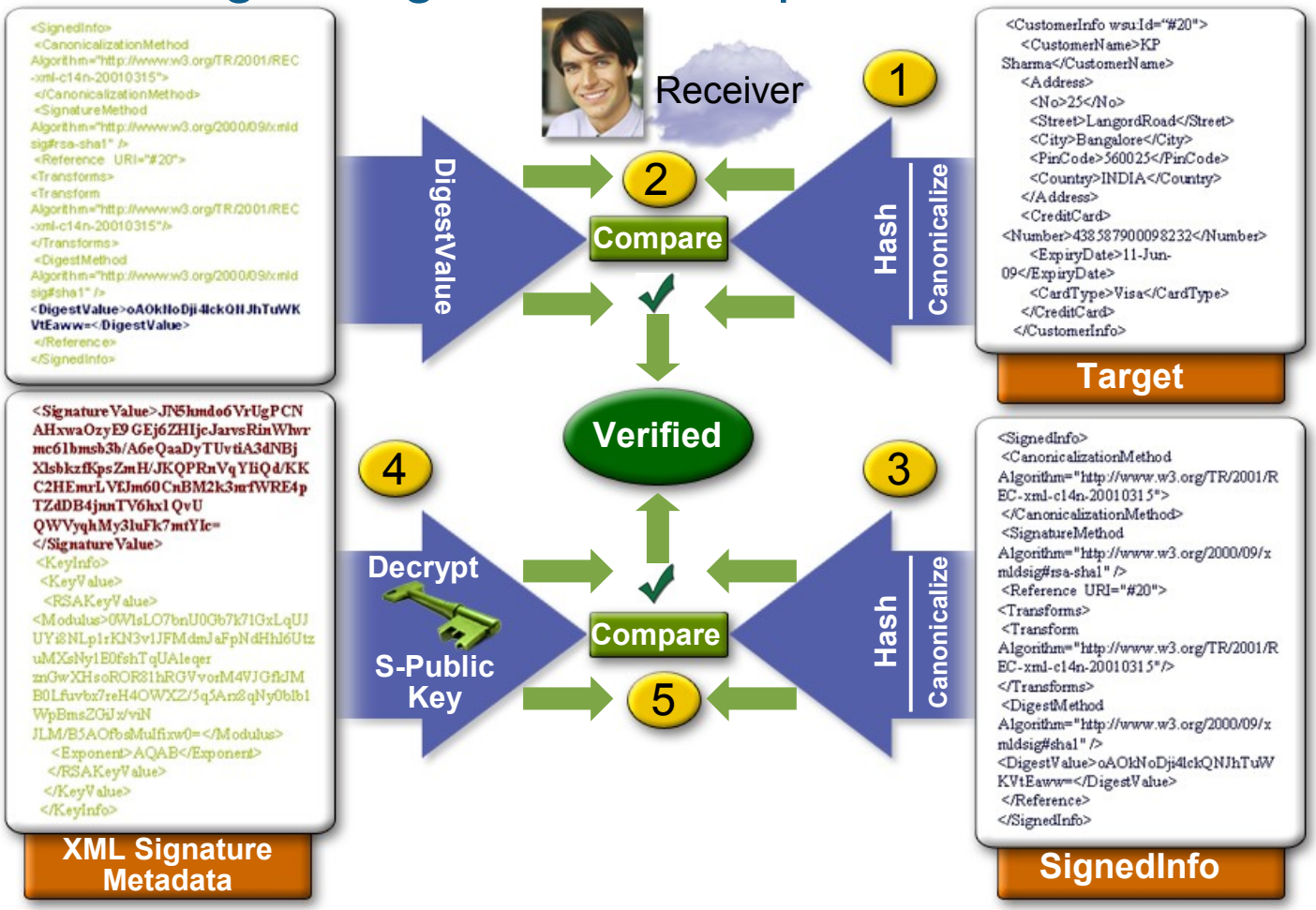
Introduction to XML Signatures

Java XML Digital Signature API Specifications Generation



Introduction to XML Signatures

Java XML Digital Signature API Specifications Generation



Agenda of this Section

- Introduction to XML Signatures
- Accelerating Java XML Signature Performance
- Crypto Acceleration on UltraSPARC T1

Accelerating Java XML Digital Signature API Specification Performance

PKCS#11 Framework

- The JSR 105 Sign/Validate operations are computationally expensive and more than 30% of the CPU time can be spent in these operations
- The Cryptographic Token Interface Standard, PKCS#11, defines the native programming interfaces to the cryptographic tokens such as hardware cryptographic accelerators and Smartcards
- PKCS#11 provides increased performance and scaling through transparent access to hardware cryptographic acceleration without modification of their applications

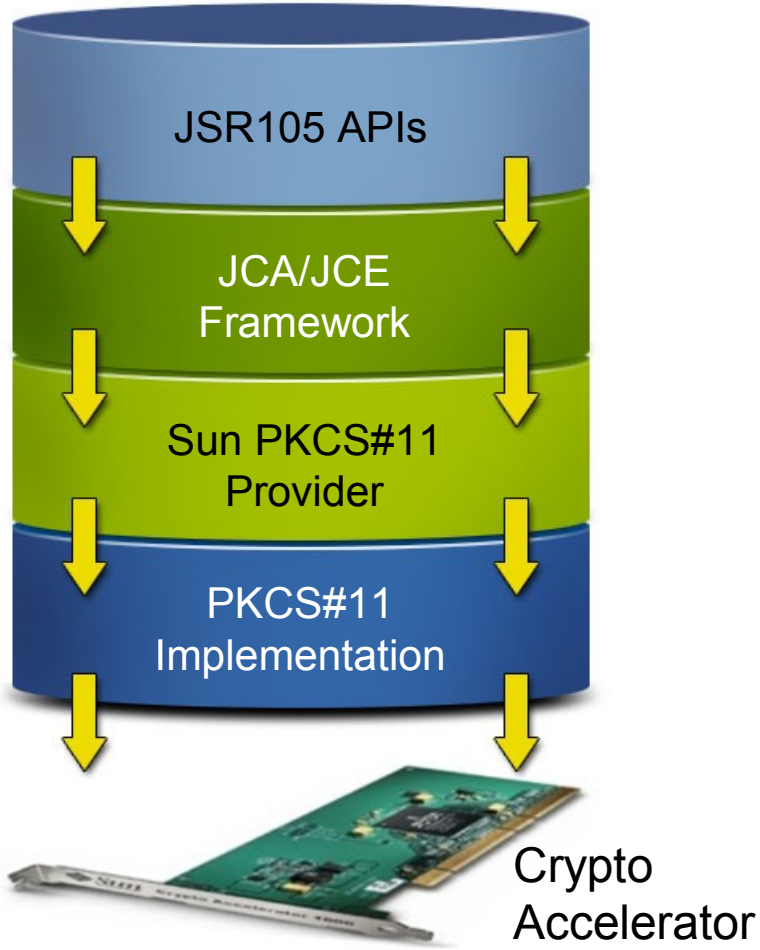
Accelerating Java XML Digital Signature API Specification Performance

Sun PKCS#11 Provider

- Starting from the JDK™ 5.0 release, Java based applications can access the cryptographic tokens using the cryptographic provider Sun PKCS#11 shipped with the JDK™ software
- SunPKCS#11 provider is a generic provider to utilize any PKCS#11 token
- The “Sun PKCS#11 provider”, does not implement cryptographic algorithms by itself—It is simply a bridge between the Java JCA, JCE APIs [4] and the underlying PKCS#11 implementations

Accelerating Java XML Signature Performance

Sun PKCS#11 Provider



Accelerating Java XML Digital Signature API Specification Performance

Sun PKCS#11 Provider

- RSA, DSA, Diffie-Hellman, AES, DES, 3DES, ARCFOUR, Blowfish, Keystore, MessageDigest, SecureRandom are some of the algorithms supported by the SunPKCS#11 provider
- The static provider installation information for the SunPKCS#11 provider can be found in the <javahome>/ jre/lib/security/java.security file

Accelerating Java XML Digital Signature API Specification Performance

Sun PKCS11 Configuration

- The Sun PKCS#11 provider is configured via the sunpkcs11 configuration file
- The sunpkcs11 configuration file contains the required property attributes for accessing the underlying PKCS#11 implementation
- The property “library” defines the pathname of PKCS#11 implementation
- Mechanisms/attributes supported by the underlying PKCS#11 implementation can be enabled or disabled from this file

Agenda of This Section

- Introduction to XML Signatures
- Accelerating Java XML Signature Performance
- Crypto Acceleration on UltraSPARC T1

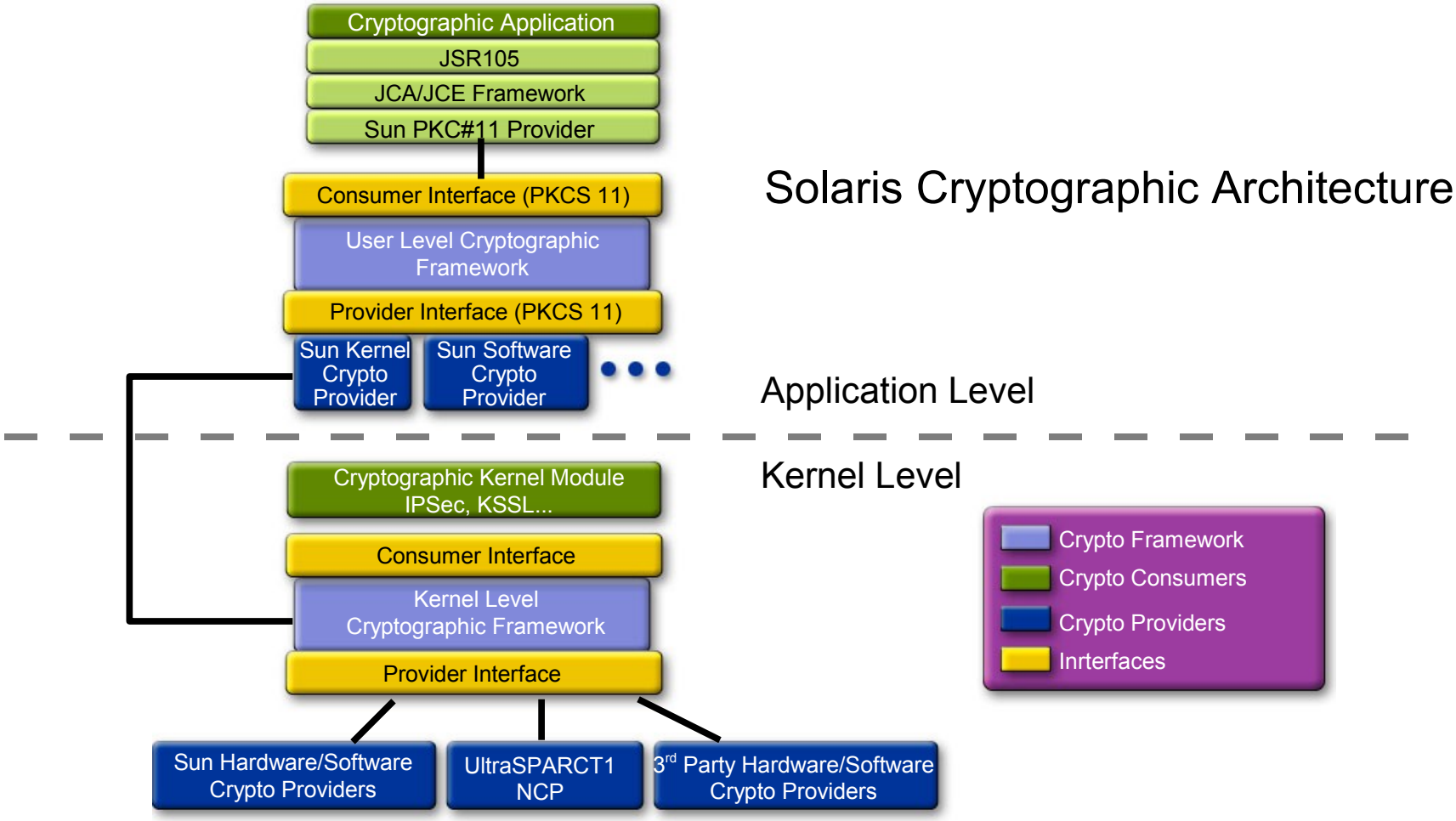
Crypto Acceleration on UltraSPARC-T1 Microprocessor

Introduction

- The UltraSPARC-T1 microprocessor comes with 8 on-chip Modular Arithmetic Unit (MAU) (1 per core), which extends the processor's capabilities to act as Cryptographic Accelerators
- The utilization of MAU has to go through Niagara Cryptographic Provider (NCP) within Solaris Cryptographic Framework (SCF)

Crypto Acceleration on UltraSPARC-T1 Microprocessor

Solaris™ Operating System Cryptographic Framework



Crypto Acceleration on UltraSPARC-T1 Microprocessor

UltraSPARC-T1 Microprocessor

- The UltraSPARC T1 microprocessor accelerates computationally expensive modular arithmetic operations found in public-key crypto algorithms such as RSA, DSA
- In the context of the Solaris Operating System Cryptographic Framework, the MAU is implemented as a Service Provider and all the 8 MAU units are made visible as a single device(/dev/ncp0) to the consumers
- This device implementation is highly available, it continues to process requests as long as at least one MAU is functional

Accelerating Java XML Digital Signature API Specification Performance

UltraSPARC-T1 Microprocessor

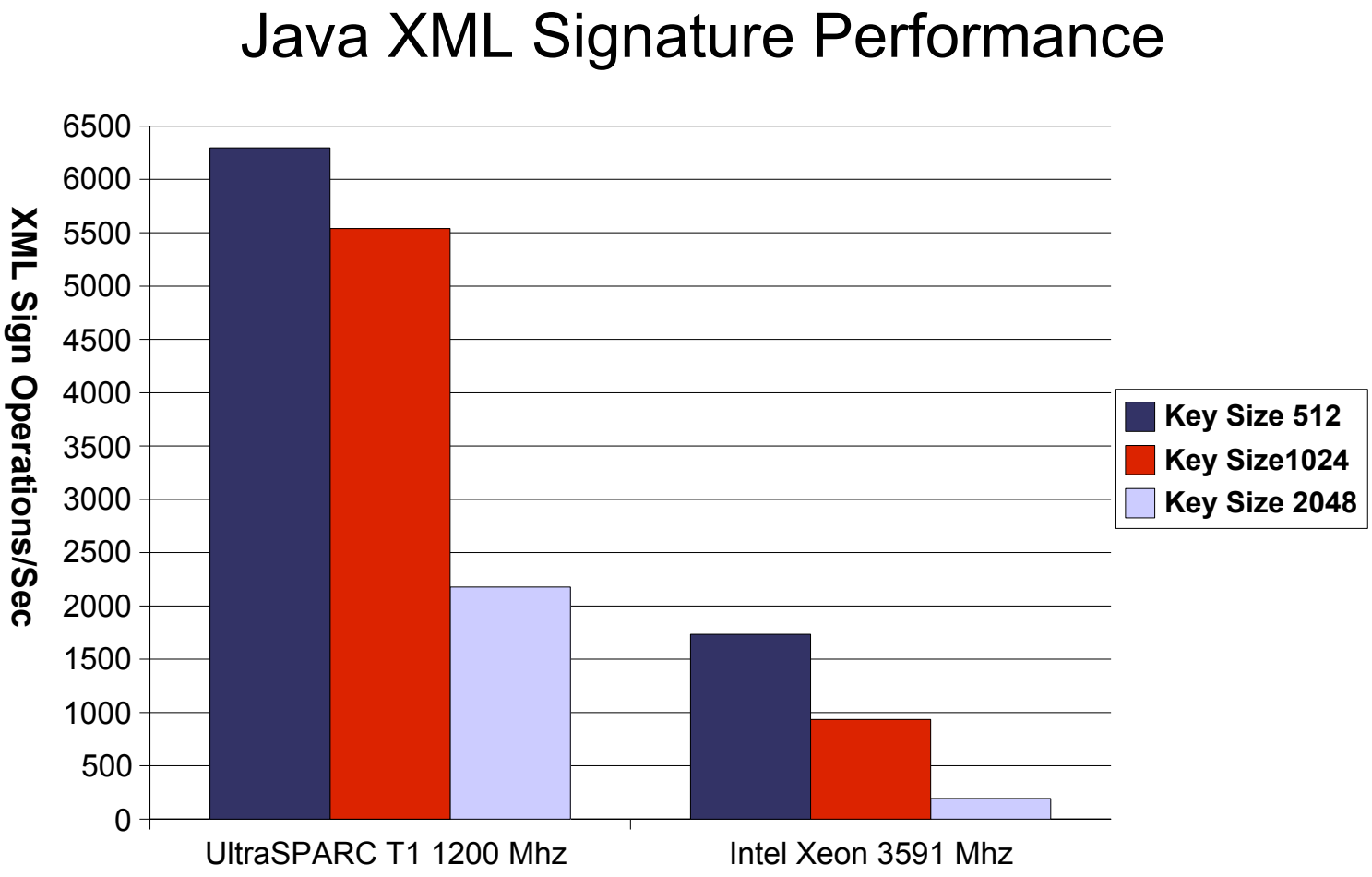
- The mechanisms are supported by the UltraSPARC T1 are CKM_DSA, CKM_RSA_X_509 and CKM_RSA_PKCS
- On an UltraSPARCT1 based system, the shipping Java VM, J2SE™ 1.5, has been pre-configured to use “SunPKCS#11 Provider”
- The Sun PKCS#11 Provider configuration file (<java-home>/jre/lib/security/sunpkcs11-solaris.cfg) contains the required information for the Sun PKCS#11 Provider to access the Solaris Operating System Cryptographic Framework (SCF)

Accelerating Java XML Digital Signature API Specification Performance

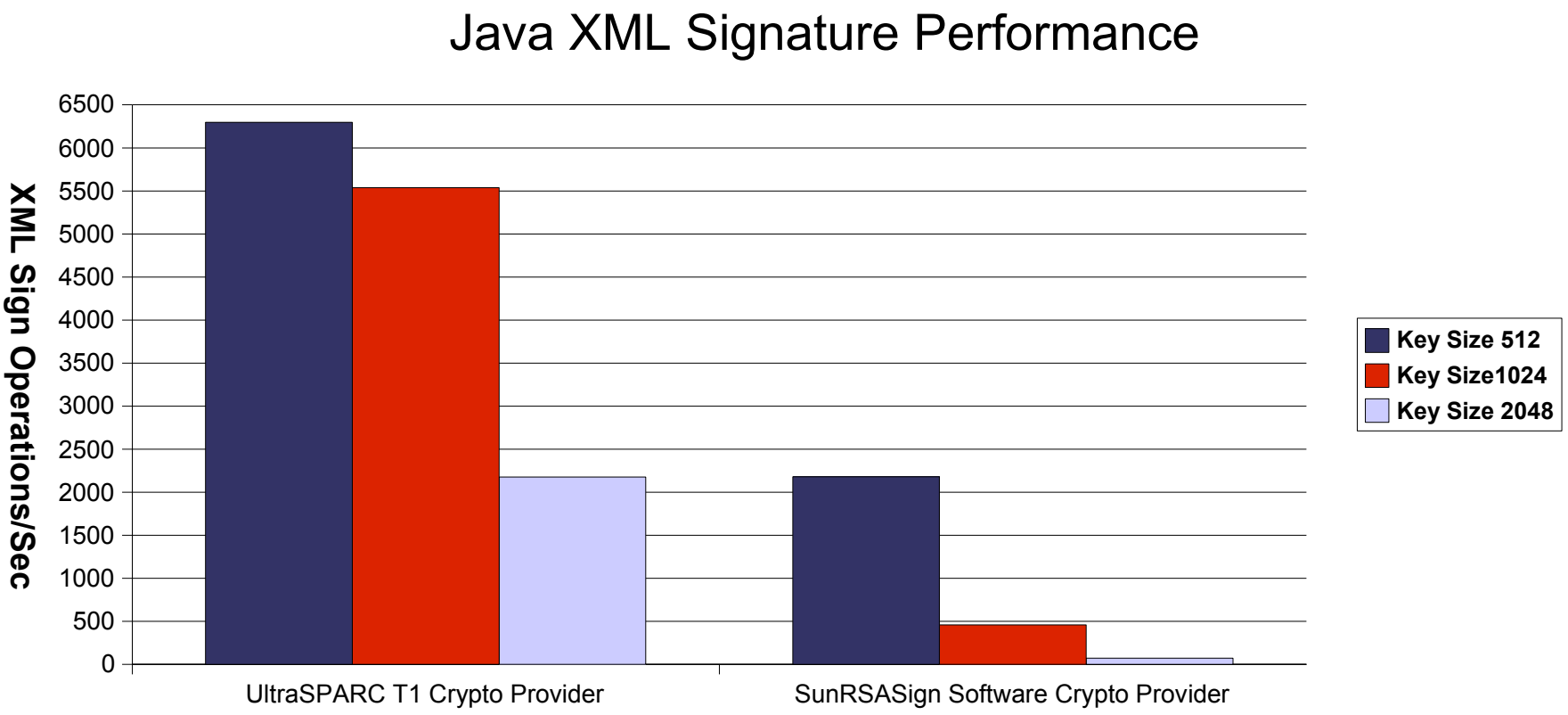
UltraSPARC-T1 Microprocessor

- One can verify that the Java-based application is indeed accessing the NCP, using the kstat command “kstat -n ncp/0”
- The kstat output will update the rsapublic and rsaprivate counters for every RSA sign and RSA verify operation respectively
- Every RSA sign/verify operation will be reflected with an increase in the kstat MAU counters

Accelerating Java XML Signature Performance



Accelerating Java XML Signature Performance



Accelerating Java XML Digital Signature API Specification Performance

Java XML Digital Signature API Specification Performance on UltraSPARC T1 Microprocessor

- From the above results one can observe the superior performance of the Sun Fire™ T2000 server
- The performance of XML Signature generation using RSA algorithm on Sun Fire T2000 is 4X to 5X of Xeon
- As the key-size increases one can observer a frog leap in the difference in performance

Accelerating Java XML Signature Performance

Java XML Signature Performance on UltraSPARC-T1

- XMLTest micro benchmark was used to measure the Java XML Signature performance
- The Throughput is defined as XML Sign operations/Sec.
- In addition to the actual signing operation, the XML Sign Operations also includes the creation of JSR 105 SignedInfo, KeyInfo, Reference, SignContext objects

For More Information

- David Dagastine's BOF-0623 Java™
- Developing and Tuning Applications on UltraSPARC Chip MultiThreading Systems
  http://www.sun.com/blueprints/1205/819-5144.pdf
- https://xmltest.dev.java.net

Q&A

Secure XML Processing Using Chip Multi-Threaded Processors

Biswadeep Nag, Kim LiChong, Pallab Bhattacharya

Java Performance Engineering
Sun Microsystems, Inc.
http://java.sun.com/performance

TS-6264