An Industry Standard Benchmark Consortium

DENBench™ Version 1.0 — Benchmark Name: RSA

Highlights

- **Benchmarks the Rivest, Shamir, and Adleman (RSA) cryptography algorithm**
- **Created in part from SSLEAY (the open-source Netscape Secure Socket Layer source code base courtesy of Eric Young)**
- **Roundtrip implementation and self-checking assures accuracy**
- **A component of the DENBench cryptography sub-suite**
- **Computationally intensive and accurate implementation of RSA algorithm modified to PKCS standards**
- **Uses Optimal Asymmetric Encryption Padding (OAEP)**

Applications and Restrictions

The RSA algorithm was first described in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT. The letters RSA are the initials of their surnames. According to Wikipedia, RSA was one of the first "strong encryption" public key cryptography schemes. It can be used for both digital signatures and encryption. The RSA cipher is used in numerous cryptographic protocols, including Transport Layer Security (TLS), Secure Socket Layer, (SSL), Secure Shell (SSH), and Internet Protocol Security (IPSEC).

RSA is much slower and therefore more computationally intensive than DES, and unlike DES is not symmetrical. Thus, there are different keys for encryption and decryption.

Although it has been proven to be vulnerable to certain attacks (including timing, man-in-the-middle, and adaptive chosen cipher attacks), it is an extremely popular algorithm used in many e-commerce (internet) and m-commerce (mobile) applications. Some people choose to implement DES, Triple-DES, or AES for stronger encryption. The major concern is really the "shared secret key" nature of the asymmetric system. Because RSA is part of the Secure Socket Layer system used so widely on the internet, and because it can be hacked by determined foes, it is now often paired with Optimal Asymmetric Encryption Padding (hence the term RSA-OAEP), and in fact EEMBC has implemented the benchmark as an RSA-OAEP system. In the benchmark, RSA-OAEP are used together with the Public Key Cryptography Standards (PKCS). The EEMBC code is based on PKCS 1.5 and OAEP 2.0R1 and implements Shoup's improvements to OAEP (in other words, EME-OAEP).

The EEMBC RSA benchmark is a cipher algorithm that provides an indication of the potential performance of a microprocessor or digital signal processor subsystem doing RSA cryptographic encryptions and decryptions.

This benchmark, and the source code, is subject to the following restrictions:

Applications and Restrictions

This software is subject to the following Export Restrictions (exportation from the United States of America to non-USA countries): *Implementations of cryptography are subject to United States Federal Government export controls. Export controls on commercial encryption products are administered by the Bureau of Export Administration (BXA) http://www.bxa.doc.gov/Encryption/ in the U.S. Department of Commerce. Regulations governing exports of encryption are found in the Export Administration Regulations (EAR), 15 C.F.R. Parts 730-774. Compliance with export restrictions is the responsibility of each individual EEMBC member, not EEMBC.*

Benchmark Description

The EEMBC RSA benchmark handling of private key operations does not depend on the private key components being present (for example, a key stored in external hardware). The recommended number of iterations is 30, and it takes about a second to run on a desktop x86 PC at about 1.7 GHz. Checking is by Cyclical Redundancy Checksum (CRC).

Analysis of Computing Resources

The benchmark is computationally challenging: addition, multiplication, extensive use of division, bit shifting, matrix math, bitwise operators such as `XOR`, and other operators are used. It is implemented in integer math. This benchmark is almost exclusively CPU bound, and the quality of the math library as well as memory library has an effect on performance. Memory moves are performed repeatedly, so optimized C library `mem*` functions would improve performance. Use of `malloc()` and heap is extensive, so optimizing memory management will yield better results. Sophisticated superscalar architectures scheduled by sophisticated compilers (or assembly language implementations) can take advantage of some parallelism. Architectures that require aligning for good performance but that do not automatically pad to obtain alignment will suffer. Odd C syntax with numerous breaks and jumps means this benchmark is unlikely to be optimized away by compiler trickery, although good standard optimization techniques (including loop unrolling and hoisting loads) would improve performance. A tool chain must implement a fair fraction of the standard C library, including `rand()` functionality.