

# Java implementation of MASH hash function family

Marek Grądzki

Military University of Technology,  
Faculty of Cybernetics,  
Institute of Mathematics and Cryptology  
Kaliskiego, 2, 00-908 Warsaw, Poland

## Extended abstract

MASH (Modular Arithmetic Secure Hash) is a hash function family based on modular arithmetic. The family consists of two unkeyed cryptographic hash functions MASH-1 and MASH-2 that have been included in Part 4 of ISO/IEC 10118 standard [5].

Motivation to use modular arithmetic in hash function is based on two factors: possibility to re-use existing software or hardware (in public-key systems) for modular arithmetic, and scalability to match required security level and desired hash value size.

In the article, two Java implementations of the functions are presented. The idea of the first implementation is based on MASH implementation published in [1]. The implementation uses standard `BigInteger` class present in Java Development Kit. Published version does not implement reduction procedure and has some minor errors, therefore we made some modifications. The result was tested against test vectors published in [5].

The implementation was then used to test general purpose parallel collision search framework based on distinguished points method (see: [3]). Framework was tested on Java-optimized 368-core Azul Compute Appliance with 128 GB of memory (see: [8]). Tests showed that only about 67% of processor cores are utilized.

Low utilization is caused by high memory consumption of `BigInteger` class. Object of `BigInteger` class is immutable - once created cannot change its state, each operation on `BigInteger` object produces new object instance.

Java uses automatic memory management. Memory is allocated when new object is created. When object is no longer used, memory is automatically reclaimed by garbage collector.

After increasing number of computing threads, number of created objects grows very fast, consuming all available memory. Full utilization of Azul appliance is not possible because cores are involved in garbage collection instead of computation.

function	class	KB/s
MASH-1	BigInteger	484.1
MASH-1	BigNum	<b>5548.23</b>
MASH-2	BigInteger	335.3
MASH-2	BigNum	<b>1250.18</b>

**Table 1.** Hash functions implementation speed.

High memory consumption of immutable `BigInteger` class motivated us to provide second implementation of MASH hash functions using mutable `BigNum` class we created. The class operates directly on vectors of primitive `int` type, which are created once and then reused. We implemented multiprecision arithmetic algorithms (addition, subtraction, division, modular exponentiation using squaring and Montgomery’s reduction) as described in [6], squaring algorithm from Java JDK 6 was used<sup>1</sup>.

Inspection of JDK sources showed, that the same algorithms are used in `BigNum` and `BigInteger` implementations. Our implementations was designed to be used in MASH hash functions. This assumption allowed for some optimizations:

- working on vectors of the same, fixed length,
- working on unsigned numbers,
- modular exponentiation only for exponents 2 and 257,
- modular reduction only for odd modulus.

Our second implementation not only allows full utilization of 368 cores of Azul compute appliance, but is faster more significantly that it was expected. We tested both implementations hashing random files of 100 MB<sup>2</sup>. Average results are presented in Table 1. Implementation based on our library is almost 4 times faster in case of MASH-2 and over 11 times faster<sup>3</sup> in case of MASH-1. Faster implementation should be possible if better algorithm for modular multiplication is used (e.g. see: [4]).

Hash functions from MASH family are slower than custom hash functions, but they can be easily changed to match required security level. As we showed, efficient Java implementations should use mutable modular arithmetic libraries optimized for usage in MASH hash functions. Immutable objects have many advantages in standard applications (simplicity, reusability, thread safety, etc. see [2]), however usage in cryptography heavily affects performance. The same optimization methods we used for MASH should also apply to other algorithms using modular arithmetic e.g. RSA.

<sup>1</sup> JDK sources are available at [9].

<sup>2</sup> Tests were performed on 2.5 GHz Core 2 Duo laptop using 128-bit  $p$  and 1028-bit  $N$

<sup>3</sup> The difference is caused by different exponents used by MASH functions. Proportion of modular exponentiation time consumption to memory management is higher in case of MASH-2.

## References

1. D. Bishop, *Introduction to Cryptography with Java Applets*, 2003.
2. Joshua Bloch, *Effective Java: Programming Language Guide*, Addison Wesley, 2001.
3. M. Grądzki, *Selected methods of one-way hash function cryptanalysis*. Master's thesis. Military University of Technology, Warsaw, 2009.
4. S.M.Hong, S.Y.Oh and H.S.Yoon, New modular multiplication algorithms for fast modular exponentiation, In *Advances in Cryptology-EUROCRYPT'96*, LNCS 1070, Springer-Verlag, 1996, pp.166-177.
5. International Standard Organization. ISO/IEC 10118-4:1998. ISO Standard - Information technology – Security techniques – Hash-functions – Part 4: Hash-functions using modular arithmetic, 1998.
6. Alfred J. Menezes, Paul C. van Oorschot, Scot A. Vanstone, *Handbook of Applied Cryptography*, CRC Press LLC, 1997.
7. P. van Oorschot and M. Wiener, *Parallel collision search with cryptanalytic applications*. *Journal of Cryptology*, 12(1):1–28, 1999.
8. Azul Systems, Documentation and specifications available at <http://www.azulsystems.com>.
9. Sun Developer Network, Documentation, specifications and source codes available at <http://java.sun.com/>.