A Java API for Using a Native PGP Implementation

THOMAS JAMPEN, MANUEL GÜNTER, TORSTEN BRAUN
Computer Networks and Distributed Systems
Institute of Computer Science and Applied Mathematics
University of Bern
Neubrückstrasse 10, 3012 Bern
SWITZERLAND
[jampen|mguenter|braun]@iam.unibe.ch http://www.iam.unibe.ch/~rvs/

Abstract: - This paper describes a Java API that provides the possibility of accessing a native implementation of PGP. The interaction between C and Java is based on the Java Native Interface (JNI) which has been included in the Java Development Kit (JDK) since version 1.1. The implementation allows to use available PGP implementations in C and, therefore, provides significant performance and security benefits. The paper describes the implementation and compares its performance with another approach based on invoking shell scripts.

Key-Words: - PGP, Java, JNI, Cryptography, Security, Secure Communication

1 Introduction

The work described is this paper has been motivated by the need to transmit Java based software agents (serialized byte code) over an IP network. In our application, software agents are used by customers for monitoring quality-of-service (QoS) provided by Internet Service Providers (ISPs) [1, 2]. We assume that users have to pay for those services and are, therefore, highly interested whether the service is really provided. Thus, authorization is needed to protect these agents from being modified. It is useful and sometimes even necessary to encrypt the agents in order to prevent them from being analyzed and attacked. The same functionality is also required when agents are used for negotiating, ordering and buying services and/or goods.

The proposed solution is a Java API [3] that uses JNI in order to provide access to a powerful and secure PGP implementation developed in C.

The structure of the paper is as follows: In the second section the involved technologies will be presented. The third section discusses a few aspects of the implementation. Performance tests and results will be presented in section 4. Finally, section 5 discusses open problems and concludes the paper.

2 Basic Technologies

2.1 PGP

PGP (Pretty Good Privacy) [5] is a well-known and widely used public key encryption program originally written by Phil Zimmermann in 1991. Now it is distributed by Network Associated Technology, Inc. (NAI) [7]. PGP is freeware for private and educational use, but it has to be licensed for commercial use. PGP is the de-facto standard for email and file encryption today, with millions of users worldwide.

The library used in this project is *PGP Tools 1.0* [6]. PGP Tools is a crypto library that is based on the PGP 2.3a source code and is compatible to all PGP 2.x versions. The reason for using version 2.3a is that this is the latest version distributed under GPL (General Public License) which means that it can be used and changed freely for own applications.

2.2 Java Native Interface (JNI)

The *Java Native Interface (JNI)* has been developed by Sun Microsystems, Inc. and is the advancement of the *Native Method Interface (NMI)* introduced in Java Development Kit (JDK) 1.0.

According to Sun, "JNI is a standard programming interface for writing Java native methods and embed-

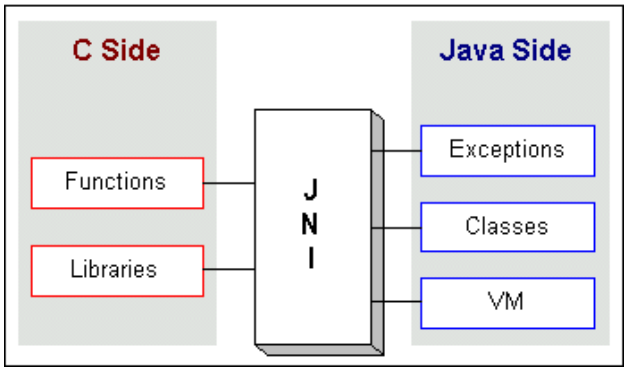

Figure 1: The Java Native Interface [11]

ding the Java^{TM} Virtual Machine into native applications [...]" [12].

JNI is a two-way interface. That means on the one hand that JNI enables Java applications to access native methods written in C, C++ or assembler language. On the other hand it is also possible to embed a Java VM within a native application (see Fig. 1).

The use of native code in Java applications provides several advantages:

- Java does not support platform-dependent features required by some applications.
- There are time-consuming calculations that are processed faster in C/C++ than in Java.
- Some applications require time-critical code that should be better implemented in C or C++ (real-time applications).
- There are existing C or C++ implementations one would like to access from Java applications.

The possibilities offered by JNI are manifold. Native methods are able to create, access and change Java objects (e.g. strings as well as arrays of objects), to call Java methods and to catch and throw exceptions. Furthermore, it is possible to load classes, to obtain class information or to perform runtime type checking.

2.3 Alternative Solutions

2.3.1 Script-based Solution

We implemented also an alternative solution that is limited to Unix environments. It is a script-based solution because the Java application calls PGP via a shell script.

One problem of this solution is that the shell output has to be parsed in order to determine whether the message has been encrypted or signed, who signed the message and whether the signature is valid.

2.3.2 JCE

Since the version 1.4 *Java Cryptography Extension (JCE)* is a part of the JDK. This version has just been released a few weeks ago and thus has not been widely tested and analysed. Any solution that is entirely based on Java has the problem that the ten years' experience of the C implementation is missing. Many security problems could have been analyzed, solved and corrected in the C implementation during the last few years, while the JCE API is still new and needs to prove its security during the next decades.

JCE neither provides an implementation of the *PGP Message Exchange Formats* (RFC 1991) [8] nor an implementation of the *OpenPGP Message Format*, defined in RFC 2440 [9]. It just provides a framework for encryption, key generation and message authentication code (MAC) algorithms. New functionality (e.g. PGP Message Format) could be added using the *Provider* interface. There is an existing provider implementation from Cryptix [10], but this implementation is based on their own JCE because until a short time ago, strong cryptography has not been allowed to be exported from America. That is why the JCE has not been a part of the JDK until version 1.4. Thus, there will still be lots of compatibility problems.

3 The API

The Java PGP API has been implemented within the Java class `PGPi`. This class provides the interface between Java applications and the C library. `PGPi` offers five different types of methods that are explained in more detail in the following sections:

- Constructors
- Native methods
- Control methods
- Status methods
- Error handling

An API documentation produced with *javadoc* can be found on online [4].

3.1 Constructors

Our API provides two constructors. The default constructor and a constructor where you can directly specify certain parameters like the data format (ASCII or binary) and if compression is desired.

3.2 Native Methods

This group contains all the native methods, that means all the methods implemented in the C library. These are the PGP-specific methods, dealing with encryption, signature generation, decryption and key management.

3.3 Control Methods

The control methods allow you to access and change parameters like the input/output format and compression.

3.4 Status Methods

Status methods inform about the validity of a signature and the name of the signer. It is also possible to determine whether a message was encrypted, signed or encrypted and signed.

3.5 Error Handling

Every native method returns an error code. In order to learn more about the error that occured, you have the possibility to get additional information on specific error codes.

4 Performance Evaluation

After programming the API a Java class has been implemented that intends to compare the existing script-based solution with the JNI solution. During this test both solutions accessed the same PGP implementation (PGP Tools). That means that for the script-based approach, PGP Tools has been compiled as an executable. The performance test involves

- encrypting and signing a plaintext file,
- decoding the ciphertext file,
- verifying the signature,
- identifying the signer and
- reading the decrypted and verified message.

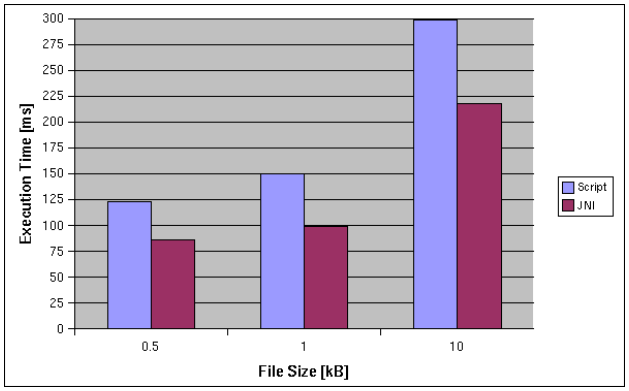

Figure 2: Performance of small file sizes.

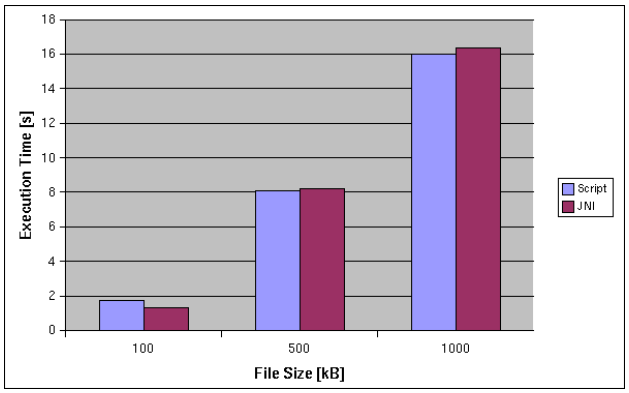

Figure 3: Performance of larger file sizes.

This procedure has been repeated several times with different input file sizes. In order to prevent falsification of the results due to temporary overload of the CPU caused by other processes, the whole test has been repeated several times.

The computer used for this performance test is a laptop with an Intel Pentium III 750MHz CPU and 256MB of RAM. The operating system is *Debian GNU/Linux "Woody"* with a 2.4.x kernel. The JDK used to compile and run the test classes was Sun's JDK 1.3.1.

Fig. 2 shows the results for small file sizes. The JNI solution is significantly faster than the script-based solution (27 - 34%). The results for larger file sizes are visible in Fig. 3 where one can see the slight performance advantage of the script-based solution over the JNI solution for files larger than 500kB (1.7% for 500kB, 2.3% for 1000kB).

5 Conclusion

5.1 Summary

With this API we offer Java application developers access to a native implementation of the widely used public key encryption application PGP. Our API makes it possible to use an approved encryption implementation not only for files and emails but also for network traffic. For most uses this API provides faster performance than the script-based solution because the encrypted and signed messages (software agents) sent over the network will be mostly smaller than 50kB. Furthermore, this API compiles and works under Windows, too. So there is no need to adapt the shell scripts and the output-parsing algorithm if one intends to use it on a different platform. The script-based solution needs to parse the output produced by PGP in order to determine whether a signature is valid, who is the signer and whether a message has been encrypted or just signed. This output can differ from platform to platform and it surely differs between different PGP versions.

As a conclusion it can be emphasized that our Java API provides a cleaner interface than a script-based solution because of the standardized Java Native Interface. With our API it is even possible to use existing key rings if they have been generated with PGP 2.x, to decrypt and verify messages encrypted with another PGP 2.x version or to encrypt and/or sign messages intended to be decrypted and verified with other PGP 2.x versions.

5.2 Open Problems

The only feature that has not been implemented yet is the key maintenance. That means that it is not yet possible to change the trust levels of public keys or to change the passphrase of your private key with this API. Of course it is possible to do so with a common version of PGP (like the famous PGP 2.6.3i, the most popular version of PGP).

Unfortunately, it seems that the JNI still needs some performance improvement because the advantage of the more direct access to the C implementation is less than expected. Hopefully, future version of the JNI will be more powerful.

References:

[1] M. Günter. *Management of Multi-Provider Internet Services with Software Agents*. PhD thesis, University of Berne, June 2001.

[2] M. Günter and T. Braun. Internet Service Delivery Control with Mobile Code. In H. R. van As, editor, *Telecommunication Network Intelligence*, pages 3–19. IFIP, Kluwer Academic Publishers, September 2000.

[3] Java API for PGP. `http://www.iam.unibe.ch/~jampen/`.

[4] Java API for PGP - API Documentation. `http://www.iam.unibe.ch/~jampen/pgpjava/PGPi.html`.

[5] The International PGP Home Page. `http://www.pgpi.org`.

[6] Programming Libraries for C/C++. `http://www.infonex.com/~mark/pgp/pgptools.html`.

[7] Network Associates Technology, Inc. `http://www.nai.com/`.

[8] PGP Message Exchange Formats. `http://www.ietf.org/rfc/rfc1991.txt`.

[9] OpenPGP Message Format. `http://www.ietf.org/rfc/rfc2440.txt`.

[10] Cryptix. `http://www.cryptix.org/`.

[11] Java Native Interface Tutorial. `http://java.sun.com/docs/books/tutorial/native1.1/`.

[12] JNI - Java Native Interface. `http://java.sun.com/j2se/1.3/docs/guide/jni/`.