

codeartisan

The Art of Writing Software

Friday, May 15, 2009

RSA Public Key Cryptography in Java

[Public key cryptography](#) is a well-known concept, but for some reason the [JCE \(Java Cryptography Extensions\) documentation](#) doesn't at all make it clear how to interoperate with common public key formats such as those produced by [openssl](#). If you try to do a search on the web for how to make [RSA](#) public key cryptography work in Java, you quickly find a lot of people asking questions and not a lot of people answering them. In this post, I'm going to try to lay out very clearly how I got this working.

Just to set expectations, this is not a tutorial about how to *use* the cryptography APIs themselves in [javax.crypto](#) (look at the JCE tutorials from Sun for this); nor is this a primer about how public key cryptography works. This article is really about how to manage the keys with off-the-shelf utilities available to your friendly, neighborhood sysadmin and still make use of them from Java programs. Really, this boils down to "how do I get these darn keys loaded into a Java program where they can be used?" This is the article I wish I had when I started trying to muck around with this stuff....

Managing the keys

Openssl. This is the de-facto tool sysadmins use for managing public/private keys, [X.509 certificates](#), etc. This is what we want to create/manage our keys with, so that they can be stored in formats that are common across most Un*x systems and utilities (like, say, C programs using the `openssl` library...). Java has this notion of its own keystore, and Sun will give you the [keytool command](#) with Java, but that doesn't do you much good outside of Java world.

Creating the keypair. We are going to create a keypair, saving it in openssl's preferred PEM format. PEM formats are ASCII and hence easy to email around as needed. However, we will need to save the keys in the binary DER format so Java can read them. Without further ado, here is the magical incantation for creating the keys we'll use:

```
# generate a 2048-bit RSA private key
$ openssl genrsa -out private_key.pem 2048

# convert private Key to PKCS#8 format (so Java can read it)
$ openssl pkcs8 -topk8 -inform PEM -outform DER -in private_
```

Contributors

[Jon Moore](#)

[Jon Moore](#)

Subscribe

 [Raw RSS feed](#)



Monthly Archive

- ▶ [2012](#) (4)
- ▶ [2010](#) (10)
- ▼ [2009](#) (7)
 - ▶ [October](#) (2)
 - ▶ [August](#) (1)
 - ▼ [May](#) (1)
 - [RSA Public Key Cryptography in Java](#)
- ▶ [February](#) (1)
- ▶ [January](#) (2)
- ▶ [2008](#) (11)
- ▶ [2007](#) (27)

```
key.pem \
    -out private_key.der -nocrypt

# output public key portion in DER format (so Java can read
it)
$ openssl rsa -in private_key.pem -pubout -outform DER -out
public_key.der
```

You keep `private_key.pem` around for reference, but you hand the DER versions to your Java programs.

Loading the keys into Java

Really, this boils down to knowing what type of `KeySpec` to use when reading in the keys. To read in the private key:

```
import java.io.*;
import java.security.*;
import java.security.spec.*;

public class PrivateKeyReader {

    public static PrivateKey get(String filename)
        throws Exception {

        File f = new File(filename);
        FileInputStream fis = new FileInputStream(f);
        DataInputStream dis = new DataInputStream(fis);
        byte[] keyBytes = new byte[(int)f.length()];
        dis.readFully(keyBytes);
        dis.close();

        PKCS8EncodedKeySpec spec =
            new PKCS8EncodedKeySpec(keyBytes);
        KeyFactory kf = KeyFactory.getInstance("RSA");
        return kf.generatePrivate(spec);
    }
}
```

And now, to read in the public key:

```
import java.io.*;
import java.security.*;
import java.security.spec.*;

public class PublicKeyReader {

    public static PublicKey get(String filename)
        throws Exception {

        File f = new File(filename);
        FileInputStream fis = new FileInputStream(f);
        DataInputStream dis = new DataInputStream(fis);
        byte[] keyBytes = new byte[(int)f.length()];
        dis.readFully(keyBytes);
        dis.close();

        X509EncodedKeySpec spec =
```


```
new X509EncodedKeySpec(keyBytes);  
KeyFactory kf = KeyFactory.getInstance("RSA");  
return kf.generatePublic(spec);  
}  
}
```

That's about it. The hard part was figuring out a compatible set of:

1. openssl DER output options (particularly the [PKCS#8 encoding](#))
2. which type of KeySpec Java needed to use (strangely enough, the public key needs the "X509" keyspec, even though you would normally handle X.509 certificates with the [openssl x509](#) command, not the [openssl rsa](#) command. Real intuitive.)

From here, signing and verifying work as described in the JCE documentation; the only other thing you need to know is that you can use the "SHA1withRSA" algorithm when you get your [java.security.Signature](#) instance for signing/verifying, and that you want the "RSA" algorithm when you get your [javax.crypto.Cipher](#) instance for encrypting/decrypting.

Many happy security returns to you.

Posted by Jon Moore at [4:18 PM](#) 

Tags: [java](#), [ice](#), [openssl](#), [public key cryptography](#), [rsa](#)

21 comments:

[Nyuk Fah](#) said...

Well Done!!

After months of searching, finally solve the openssl DER format.

Thank you very much

[June 3, 2009 at 4:29 AM](#)

Anonymous said...

Thanks a lot!!!

[June 24, 2009 at 6:35 AM](#)

[towkach](#) said...

You saved my job!!!

[October 7, 2009 at 5:31 PM](#)

[Fred Sauter](#) said...

Had I had this advice some six years earlier! :-)

[November 26, 2009 at 3:40 PM](#)

Anonymous said...

How we generate public privet key without openssl. can we use tynica as somekind of CA to generate keys.
thanks in advance.

[December 2, 2009 at 1:08 PM](#)

[Jon Moore](#) said...

@Anonymous: Here you go: <http://tinyurl.com/ygwvqow>.

[December 4, 2009 at 11:49 PM](#)

[JetForMe](#) said...

Thanks, Jon, for a clear and concise post. I'm trying to duplicate the functionality of this command in Java, and I've gotten most of the way there thanks to your help, but in the end, I get a different signature:

```
$ openssl dgst -sha1 -binary < SomeFileToSign | openssl dgst  
-dss1 -sign myprivatekey.pem > openssl.sig
```

I did the PEM->DER conversion as you suggest and am able to sign without exceptions, but the resulting bytes are different.

I've verified that I get the same SHA1 digest. I'm creating a KeyFactory of "DSA" type, and a Signature of "DSS" type. I figure I've left out some configuration of Signature, and I'm hoping you might know what it is off the top of your head. Thanks much!

[January 5, 2010 at 6:06 PM](#)

[JetForMe](#) said...

Well, that was dumb. I realize now that the signature is different every time. Clearly, I need to figure out how to verify the signature generated by Java using OpenSSL, but I can't figure out the right command.

Thanks anyway!

[January 5, 2010 at 7:14 PM](#)

[Jon Moore](#) said...

@JetForMe: Not being as familiar with the DSA-DSS signatures as I am with RSA-SHA1, I'm not sure I can directly help.

There are often configuration options you can pass to the Signature.getInstance after the normal algorithm name that affect things like padding schemes, etc. Perhaps you need to more specifically specify the variant of Signature algorithm you want so that it matches what OpenSSL is doing.

[January 5, 2010 at 8:25 PM](#)

Federico said...

I think I love you! I wasted one week trying to understand the

right key format to give to Java to get this stupid PublicKey object, and nothing was working!

By the way, can anyone explain me why if I use `PublicKey.toString()` I get the key in nice clear text, but not if I use `new String(PublicKey.getEncoded())`?
How to I decode a DER key to see it in clear text?(and where the hell is the `toString` function overridden in `PublicKey`?)

[November 30, 2010 at 9:01 AM](#)

[Jon Moore](#) said...

@Federico: glad the article helped.

DER format is a binary format, which explains why you aren't seeing anything intelligible from `new String(key.getEncoded())` -- you're basically saying "take this fairly random set of bytes and pretend it's a string", whereas `PublicKey.toString()` is explicitly meant to give you a proper String representation of the key.

`toString()` is probably being overridden in your cryptography provider (e.g. BouncyCastle or the like); that's where the object implementing the `Key/PublicKey/RSAPublicKey` interface hierarchy lives.

[December 1, 2010 at 6:27 AM](#)

Federico said...

Thank you again for the explanation, and since you are so knowledgeable I would like to bother you with one more question if I may:

I tried to extract the public key from the CAcert root certificate in txt format

(<https://www.cacert.org/certs/root.txt>), and use it in java, but of course without success.

Do you have any idea about how one might convert it into the correct DER format (since it is the only one Java understands)?

Thanks in advance, and again, great article :)

[December 2, 2010 at 5:23 AM](#)

[Jon Moore](#) said...

@Federico: without having lots of time to debug, you may see a plaintext section of that cert URL that has "BEGIN CERTIFICATE" and "END CERTIFICATE" lines. This should be a PEM-format public key; you ought to be able to use an `openssl` command similar to the ones in the article here to convert that to DER format.

[December 5, 2010 at 9:42 AM](#)

Federico said...

I see, so that was indeed a key.

Thank you for the help, but as I feared, there is no pure programmatic way to handle and convert these keys in java.

One must always rely on openssl...

[December 7, 2010 at 8:59 AM](#)

Anonymous said...

Hi, thank you for this great article.

I managed to load my openssl RSA keys into java, but the original openssl key is 1024 bits, java is creating a PrivateKey with 4096 bits, thus resulting in a padding error in the cipher algorithm.

Any suggestions, why I got a mismatch in the keylength?

Thank you very much

[November 29, 2011 at 11:40 PM](#)

[loreij](#) said...

This comment has been removed by the author.

[March 14, 2012 at 9:22 AM](#)

Anonymous said...

Here is a small piece that accepts the PEM files directly no need of conversion to DER

```
public static PrivateKey getPriv(String filename) throws
Exception {
    File f = new File(filename);
    FileInputStream fis = new FileInputStream(f);
    DataInputStream dis = new DataInputStream(fis);
    byte[] keyBytes = new byte[(int) f.length()];
    dis.readFully(keyBytes);
    dis.close();

    //byte[] temp = new byte[keyBytes.length];
    String temp = new String(keyBytes);
    String privKeyPEM = temp.replace("-----BEGIN RSA PRIVATE
KEY-----\n", "");
    privKeyPEM = privKeyPEM.replace("-----END RSA PRIVATE KEY-
-----", "");
    //System.out.println("Private key\n"+privKeyPEM);

    Base64 b64 = new Base64();
    byte [] decoded = b64.decode(privKeyPEM);
    //dumphex(decoded,"PrivateKey");

    PKCS8EncodedKeySpec spec = new
    PKCS8EncodedKeySpec(decoded);
    KeyFactory kf = KeyFactory.getInstance("RSA");
    return kf.generatePrivate(spec);
}
```

[March 26, 2012 at 8:54 AM](#)

Anonymous said...

When I tried to read private_key.pem from the code specified,

I am getting
Exception in thread "main"
java.security.spec.InvalidKeySpecException:
java.security.InvalidKeyException: IOException : algid parse
error, not a sequence
at
sun.security.rsa.RSAKeyFactory.engineGeneratePrivate(Unknown
Source)
at java.security.KeyFactory.generatePrivate(Unknown Source)
at
AsymmetricCipherTest.getPriv(AsymmetricCipherTest.java:286)
at AsymmetricCipherTest.main(AsymmetricCipherTest.java:59)

[November 1, 2012 at 7:14 AM](#)

Anonymous said...

can any one give me complete code to use openssl generated
key in rsa and dsa encryption and decryption

[March 31, 2013 at 4:08 AM](#)

Anonymous said...

You're a genius my friend. Saved us hours of stupid
debugging!!! Thanks!

[May 23, 2013 at 7:53 AM](#)

[Amber Salm](#) said...

What a brilliant explanation you have shared in this article. I
am so happy that I found this post as I need not to search
anymore. Thanks a lot for sharing it.

[public key infrastructure](#)

[July 9, 2013 at 9:49 AM](#)

[Post a Comment](#)

Links to this post

[Create a Link](#)

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

All articles Copyright © 2007-2010 by Jon Moore. All rights reserved.