Encryption

Find information about and sample codes for encryption in Relais D2D.

Encryption can be used

- When passing patron credentials to Relais via an OpenURL. See Security in Relais, Option 1.
- When using the Authentication service to obtain an 'aid'. See Authentication.

Encryption details:

- Algorithm: RSA
- Padding: PKCS1Padding
- Key size: 2048 bit

Sample code

Sample code for encrypting text using Relais public-key is provided below in five languages:

- Java
- Perl
- PHP
- Python
- Ruby

Disclaimer: The sample code does not use actual data. The sample code provided is for demonstrating the use of Relais public-key for encryption. The code may not be of production quality but is provided as a guide for getting one started.

The following warnings apply to all sample code:

1. To safeguard against replay attack the plain text of any of the attributes you will encrypt (patron id, patron surname, etc.) must have the UTC datetime appended to it in the following format before the entire string is encrypted:

   actual_value|yyyyMMdd HHmmss

   For example: "12391334|20150706 163237", where

   12391334 = patron id in plain text
   20150706 163237 = current UTC datetime. * Note there is a space between the yyyyMMdd and the HHmmss

   When doing the encryption be sure to encrypt the full text string, e.g., "12391334|20150706 163237"
After the value is un-encrypted, the plain text is split using the '|' character. If '|' is not found the Authentication service will fail.

The UTC datetime in the encrypted text must be within the last 5 minutes of the current UTC time. For example, if the UTC datetime in the encrypted text + 5 minutes is before the current UTC datetime, the Authentication process will fail. Similarly, if the UTC datetime in the encrypted text is after the current UTC datetime the Authentication process will fail.

2. When passing patron credentials to Relais via an OpenURL, all special characters must be percent-encoded as per RFC 3986 URI Generic Syntax. See URL encoding for more details on coding special characters.

Java

Version: Java 1.7 and 1.8

Dependency:
Apache Commons Codec
The Legion of the Bouncy Castle

Apache Maven:

<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.10</version>
</dependency>

<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcpkix-jdk15on</artifactId>
    <version>1.51</version>
</dependency>

<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcprov-jdk15on</artifactId>
    <version>1.51</version>
</dependency>

Code Sample:

import java.io.FileReader;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PublicKey;
import java.security.Security;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import org.apache.commons.codec.binary.Base64;
import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.openssl.jcajce.JcaPEMKeyConverter;

public class RSACipher {
    public static void main(String[] args) {
        // ------------------------------
        // add BouncyCastle JCE provider
        // ------------------------------
        Security.addProvider(new BouncyCastleProvider());

        // ------------
        // read public key
        // ------------
        PublicKey publicKey = null;
        try {
            PEMParser pemParser = new PEMParser(new FileReader(
                "/etc/pki/relais/orii-public.key"));
            Object pemObject = pemParser.readObject();
            pemParser.close();

            JcaPEMKeyConverter converter = new JcaPEMKeyConverter()
                .setProvider("BC");
            publicKey = converter
                .getPublicKey((SubjectPublicKeyInfo) pemObject);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // -----------------------------------
        // encrypt plaintext using public key
        // -----------------------------------
        String plaintext = "As cunning as a fox who's just been appointed Professor "
            + "of Cunning at Oxford University? - Blackadder";
    }
}
String ciphertext = null;
try {
    Cipher rsaCipher = Cipher
        .getInstance("RSA/NONE/PKCS1Padding", "BC");
    rsaCipher.init(Cipher.ENCRYPT_MODE, publicKey);

    byte[] encryptedVal = rsaCipher
        .doFinal(plaintext.getBytes("UTF-8"));

    // URL safe Base64 encoded string
    ciphertext = Base64.encodeBase64URLSafeString(encryptedVal);
} catch (NoSuchAlgorithmException | NoSuchProviderException
    | NoSuchPaddingException | InvalidKeyException
    | IllegalBlockSizeException | BadPaddingException
    | UnsupportedEncodingException e) {
    e.printStackTrace();
}
System.out.println("Ciphertext:
" + ciphertext);
}

Perl

Version: Perl 5.14.4

Dependency:
Crypt::OpenSSL::RSA module (https://metacpan.org/pod/Crypt::OpenSSL::RSA)
MIME::Base64 module (https://metacpan.org/pod/MIME::Base64)
File::Slurp module (https://metacpan.org/pod/File::Slurp)

Code Sample:

#!/usr/bin/perl
use Crypt::OpenSSL::RSA;
use MIME::Base64;
use File::Slurp;

# read public key
my $publicKeyFile = "/etc/pki/relais/oorii-public.key";
my $publicKey = Crypt::OpenSSL::RSA->new_public_key(scalar read_file $publicKeyFile);
$publicKey->use_pkcs1_padding;

# encrypt plaintext using public key
my $plaintext = <<"END";
I’m afraid I’m not personally qualified to confuse cats,
but I can recommend an extremely good service.
        - Vet, Monty Python’s Flying Circus
END

# the following Base64 encoded string is not URL safe
# in production, it may be necessary to use URL safe Base64 encoding
my $ciphertext = encode_base64($publicKey->encrypt($plaintext));

print "\nCiphertext:\n" . $ciphertext . "\n";

PHP

Version: PHP 5.4.17

Dependency:
phpseclib: RSA (http://phpseclib.sourceforge.net/rsa/intro.html)

Code Sample:

<?php
include('Crypt/RSA.php');

# read public key
$rsa = new Crypt_RSA();
$rsa->setEncryptionMode(CRYPT_RSA_ENCRYPTION_PKCS1);
$rsa->loadKey(file_get_contents("/etc/pki/relais/oorii-public.key"));

# encrypt plaintext using public key
$plaintext = "As sharp as a freshly groomed porcupine.";
$ciphertext = $rsa->encrypt($plaintext);

# the following Base64 encoded string is not URL safe
# in production, it may be necessary to use URL safe Base64 encoding
$ciphertext = base64_encode($ciphertext);

print "\nCiphertext:\n" . $ciphertext . "\n";
?>

Python

Version: Python 2.7.3

Dependency:
M2Crypto (https://pypi.python.org/pypi/M2Crypto/)

https://help.oclc.org/Resource_Sharing/Relais_D2D/Authentication/Encryption
Printed: Fri, 23 Sep 2022 12:00:22 GMT
Code Sample:

```
#!/usr/bin/python
import M2Crypto

# read public key
publicKey = M2Crypto.RSA.load_pub_key("/etc/pki/relais/oorii-public.key")

# encrypt plaintext using public key
plaintext = "We may not be big but we're small! - Stuart McLean"
ciphertext = publicKey.public_encrypt(plaintext, M2Crypto.RSA.pkcs1_padding)

# the following Base64 encoded string is not URL safe
# in production, it may be necessary to use URL safe Base64 encoding
# ciphertext = ciphertext.encode("base64")

print "Ciphertext:
" + ciphertext
```

Ruby

Version: Ruby 1.9.3

Dependency:
OpenSSL module (http://ruby-doc.org/stdlib-1.9.3/lib...c/OpenSSL.html)
Base64 module (http://ruby-doc.org/stdlib-1.9.3/lib...oc/Base64.html)

Code Sample:

```
#!/usr/bin/ruby
require 'openssl'
require 'base64'

# read public key
publicKeyFile = "/etc/pki/relais/oorii-public.key";
publicKey = OpenSSL::PKey::RSA.new(File.read(publicKeyFile));

# encrypt plaintext using public key
plaintext = "As nervous as a long-tailed cat in a room full of rocking chairs."

ciphertext = Base64.encode64(publicKey.public_encrypt(plaintext));

puts "Ciphertext:\n" + ciphertext + "\n";
```