# Java Security – Encryption

By Ronald Brown
CS 2340 – Spring 2011

## I. What is encryption in Java mean and why do I need it?

### 0. Introduction to Crypto

The first thing many people thing of when they hear the world encryption normally relates to taking some form of data, transforming it into another non-human readable set of data commonly known as encrypted data, and presenting it to a known party with the hopes that said data will not fall into mischievous hands. In order to achieve this task, complex mathematical algorithms are used to take data and transform it into what is known as cipher text. There are many ways in java to do encryption, some of which are very much non trivial, however the Java community does a fairly good job of providing out of the box cryptography libraries for use with your application. Some of these libraries include BouncyCastle, JCA/JCE, and the javax.crypto package that comes bundled with Java.

### 1. Designing with Crypto in mind

In order to successfully integrate Encryption into your program, it is important to design the system with this in mind from the beginning. Some of the more CPU intensive mathematical calculations may need to be implemented before the first line of actual general code implementation is created. Things to keep in mind are things like:

- Who are the users?
- How will parts of the application communicate with one another?
- What encryption scheme will I use?
- Is this going to be a networked application, if so how will we implement key exchange (depending on the schema implemented)?
- Are there passwords involved?
- Is there a database involved to store those passwords?
- What type of hashing algorithm will I use to encrypt these passwords?
- What type of general cryptographic scheme will I need to implement (DES, RSA, AES, etc)?

These are all things to keep in mind when designing your system. For my 2340 Spring 2011 project, I choose to implement a combination of different encryption schemes along with my networked application. Having a networked client/server application, I decided to use Diffie-Helman( http://en.wikipedia.org/wiki/Diffie–Hellman_key_exchange ) key exchange to exchange the private keys of both the client and server applications. I then used DES(http://en.wikipedia.org/wiki/DES) type encryption to encrypt all traffic with each client's public/private key (using non-blocking sockets and threading this was simple to implement) and transmit

data between an instance of the client and server application. Finally, we used SHA1 hashing to hash all passwords and store said hashes in our database. The reason for storing the hash rather then the password supplies a safety net for us. If for some reason our MySQL database was compromised, the attacker would now have a list of usernames but hashed passwords. This means we had to have our server portion of the application compute the hash value each time a user logged in, and checked it agains the saved hash password in the database.

## II. How do I use encryption?

### 1. Implementation

As stated earlier, implementing encryption in Java is fairly easy. Here is an example from the Diffie-Hellman class I wrote for my project this semester:

```
SecretKey CreateFullKey(byte[] publicKeyBytes)
        {
                try {
                    // makes the key bytes into a public key object
                    X509EncodedKeySpec x509KeySpec = new
X509EncodedKeySpec(publicKeyBytes);
                    KeyFactory keyFact = KeyFactory.getInstance("DH");
                    PublicKey publicKey = keyFact.generatePublic(x509KeySpec);

                    // Prepare to generate the secret key object from our
privatekey and publicKey
                    KeyAgreement ka = KeyAgreement.getInstance("DH");
                    ka.init(privateKey);
                    ka.doPhase(publicKey, true);

                    // Our algorithm we are using for encryption
                    String algorithm = "DES";

                    // Generate the secret key and return to user
                    return ka.generateSecret(algorithm);

                }

                //catching some common errors(none of them occur tho)
                catch (java.security.InvalidKeyException e) {

                }
                catch (java.security.spec.InvalidKeySpecException e) {

                }
                catch (java.security.NoSuchAlgorithmException e) {

                }
                //default return value
                return null;

                }
```

As you may have noticed, this method is used to generate a SecretKey that can be used in the deciphering of our encrypted data. The reason we are generating a key here is because of the way Diffie-Hellman works. (**Note: if you are confused and would like to know more, please visit the Wikipedia page on Diffie-Hellman to get a better understanding, or take CS 4235!**).

As stated earlier, we also implemented hashing, here is an exert from that function of our project:

```
private static String hash(String str){
	try{
		MessageDigest sha1 = MessageDigest.getInstance("SHA1");
		byte[] digest = sha1.digest((str).getBytes());
		return byteToHex(digest);
	}
	catch(Exception e){
		System.out.println(e);
		return "";
	}

	}
```

The entire purpose of this function is simply to take the input string, and hash it using the SHA1 hashing method. As you can see, this is super simple to do and requires virtually no prior knowledge other then Google ☺.

**2. OMG THIS IS SO COOL, I HAS KEYS!**

So you have keys and you're ready to encrypt. You've implemented diffie-helman, you have some sockets, you're PUMPED. Now let's do some actual encryption using something like DES. Without going into the actual implementation of these methods, here are the actual methods used to encrypt and decrypt traffic in our application(**NOTE, I do not show HOW encipher and dcipher are actually setup, this can be left to the implementer)** :

```
Cipher ecipher;
Cipher dcipher;
 public String encrypt(String str) {
        try {
            // Encode the string into bytes using utf-8
            byte[] utf8 = str.getBytes("UTF8");
// Encrypt
            byte[] enc = ecipher.doFinal(utf8);
// Encode bytes to base64 to get a string
            return Base64Coder.encodeLines(enc);

        } catch (Exception e) {
           e.printStackTrace();
        }
        return null;
    }

    //decrypts the string using the specified secretkey
    public String decrypt(String str) {
        try {
            // Decode base64 to get bytes
            byte[] dec = Base64Coder.decodeLines(str);
```

```
            // Decode using utf-8
            byte[] utf8 = dcipher.doFinal(dec);
            return new String(utf8, "UTF8");
        } catch (Exception e) {
           System.out.println("Decrypt Fail: " + str);
           e.printStackTrace();
        }
        return null;
         }
```

Without going into too much detail, we used our already known AND saved public/private keys to encrypt/decrypt all traffic. We knew these keys because at this point, we have already done the DH key exchange and (because we used threading with non-blocking sockets) we also had ALL the private keys of all the clients without sending them using clear text (again, thanks to DH).

## III. Cool so I can encrypt stuff, now what?

### 1. Try it out!

Pick an encryption scheme and try it out. Here are a few links to check out for some examples of encryption in Java along with some libraries. Also, just a word of wisdom, DON'T WRITE YOUR OWN CRYPTO. DON'T WRITE YOUR OWN CRYPTO, DO NOT WRITE YOUR OWN CRYPTO.

http://www.java2s.com/Code/Java/Security/Encryption.htm
http://www.bouncycastle.org/
http://java.sun.com/developer/technicalArticles/Security/AES/AES_v1.html
http://download.oracle.com/javase/6/docs/api/javax/crypto/package-summary.html
http://www.source-code.biz/base64coder/java/ (This is an AWESOME Base64 Encoder, avoid the sun.java Base64 encoder at all costs).