**Encryptor Walkthrough** (version 1.1)

**Revision Log**

| **Ver. No.** | **Version Date** | **Summary of Change** | **Changed By** |
| --- | --- | --- | --- |
| 1.1 | 30/Apr/2016 | Fix the bug of ‘Hash Function’ panel not displaying Salt when BCrypt is selected | Samuel Huang |
| 1.0 | 25/Apr/2016 | Initial Version | Samuel Huang |

This document will serve as a quick tutorial on how the application is implemented using NetBeans 8.1 so the reader can get an idea of how to maintain or enhance the application should they decide to do so.

**In the beginning**

Back in 2013, the testers of the company I was working at needed an easy way to test a *Change Password Webservice* that used the **Twofish** algorithm to encrypt passwords. The database would store the actual password as **MD5** hashed string so the testers also needed a simple way of verifying if the right password was stored.

Hence I took it upon myself to implement a simple Swing app named **Encryptor** that did both requirements above in my own time. This is an important point because I wasn’t asked by the company to do this so I can distribute *Encryptor* now as Open Source software. ;)

Fast forward to 2016 as of this writing, the Encryptor app is now enhanced to use more algorithms (AES, BCrypt, …), log4j 2(for logging) and better UI.

**Functional Design**

Encryption/Decryption (Figure 1)

**Definition:** 'Symmetric-key Encryption'algorithms can use a predefined secret key to encrypt data and then decrypt the encrypted data using the same key that was used to encrypt the data in the first place.

The 'Symmetric-key Encryption' tab can encrypt/decrypt data using one of the two algorithms, **Twofish** and **AES**. AES is considered (or hyped) to be military grade encryption by NSA/industry (really?). If one does use AES, make sure the key size is at least 256 bits long for security. One may argue its overkill but it never hurts to be overkill when it comes to security and we want to entertain anyone who tries to break it by brute force (should take ridiculously long time).

Both the 'Key' and 'Data' text fields need to be populated before encryption can happen.

The encrypted data could also be decrypted using the key in the 'Key' text field.

The encryption key can be randomly generated by clicking the 'Generate' button.

The generated key in 'Key' text field will be Base64 encoded string from the generated binary data.

The **encryption key** can be generated in 3 different sizes, 128, 192 and 256 bits from the 'Key size' drop down list.

The longer the key length, the stronger the strength of encryption will be in theory so always use the longest key size if performance hit in the server is negligible.

Hash Function (Figure 2)

**Definition:** A hash function is basically a one-way hash of input data such that once hashed, the data is practically impossible to revert back to its original input. Hash function is not considered to be the same as encryption even though it seems to be doing the same thing.

The hashed data will look like encrypted data except there is no way to reverse it back to its original form. If one does figure out magic algorithm(s) to reverse the Hashed data of a popular hash function within a reasonable time, that someone could expect fame if not fortune.

Four hash functions are available in ‘Hash Function’ tab as seen in Figure 2. These functions are **MD5, SHA-1, SHA-256** and **BCrypt**. See https://en.wikipedia.org/wiki/Cryptographic\_hash\_function for more details of hash functions.

After selecting the desired algorithm from *Algorithm* drop down list, click the *Generate* button to generate salt as Base64 encoded string. Then fill the *Data* text field with password to be hashed before clicking the *Hash Data* button. Hashed password will then appear in the *Hashed Data* text area.

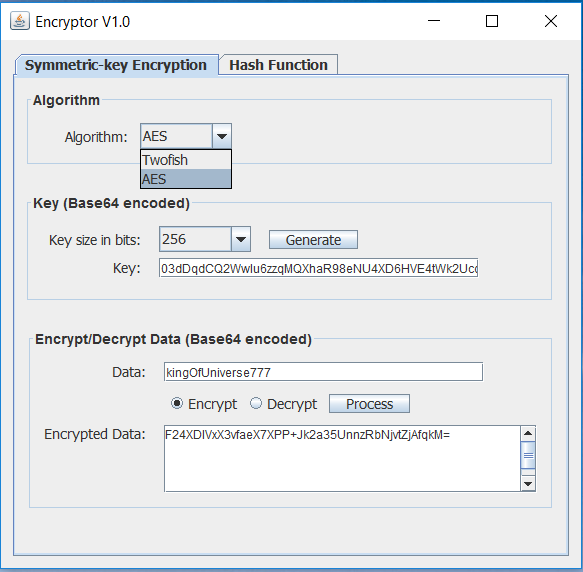
The most common application of hash function is to generate hashed string of passwords for storing in database. The salt will also be stored (or at least derivable from some algorithm at runtime) in database in this case so authentication can be done by comparing the hashed password from database against the hash of

***salt + {user supplied password}***

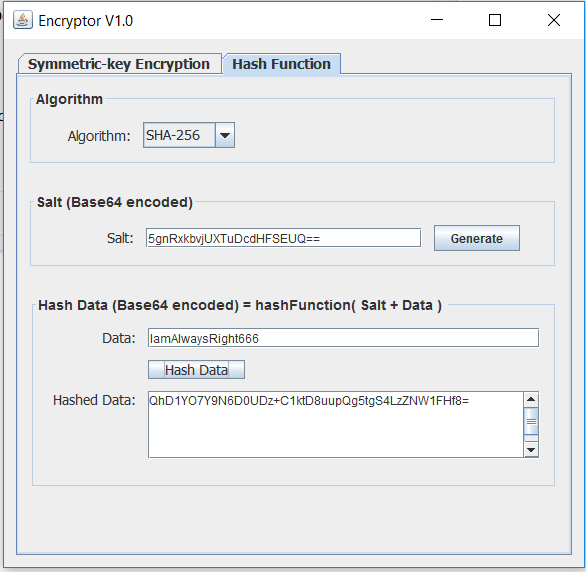
at runtime to see if they are the same.

**Do not use MD5 to hash data** for new project, **period**. It's considered insecure and has been shown to be cracked REPEATEDLY. The only reason it’s still used in the real world is legacy applications are stuck with MD5 hashed passwords stored in database and it’s infeasible to switch to a more secure hash function without burning time and kaachinnn, not to mention the potential risk of happy customers if anything goes wrong. ;)

A good alternative for MD5 would be *SHA-256* or *BCrypt* provided the salt is randomly generated.

****

**Figure 1:** Panel for encrypting/decrypting data by using a Base64 encoded string as secret key

****

**Figure 2:** Panel for hashing data by using a Base64 encoded string as salt

**Stop right there…**

First of all, if you know nothing about Swing and are not required to use it on your job then **STOP RIGHT THERE** and stop learning Swing. Why? Swing has reached End of Life (EOL) already and Oracle isn’t going to support it anymore.

This means Swing is dying and anytime spent on learning Swing will be better spent on learning **JavaFX**, which is the replacement for Swing and the next big thing for Java desktop app as planned by Oracle. Having said that, feel free to knock yourself out. ;)

**Loading the project with NeBeans IDE 8.1**

After downloading the project from GitHub, import the project (File 🡪 Open project 🡪select ‘Encryptor’ folder) into NetBeans 8.1.

*Don’t even think of using another IDE* to do this. Why?

Because the pixel perfect GUI layout was all done in NetBeans by drag and drop Swing components from *Palette* panel into *Design* tab/panel. See Figure 3. All the Swing code is generated by simply drag and drop. One doesn’t even need to be a Swing expert to achieve good GUI layout. To see Figure 3 in NetBeans, double click the *EncryptorUI.java* file from *Files* panel/tab on the left, then *Design* tab on the middle.

As a bonus, by just clicking a component in the Design, it will bring focus to the followings automatically as seen in Figure 3:

* Focus to the clicked component in *Navigator* panel on lower left area
* Focus to the *Properties* panel of the clicked component on the right.

The *Properties* panel can be used to customize the properties of selected component directly, e.g. font, text to display, foreground colour, preferred width … And you wouldn’t even be able to change/delete the generated Swing code in *Source* panel if you try. So less chance of introducing bugs by accident.

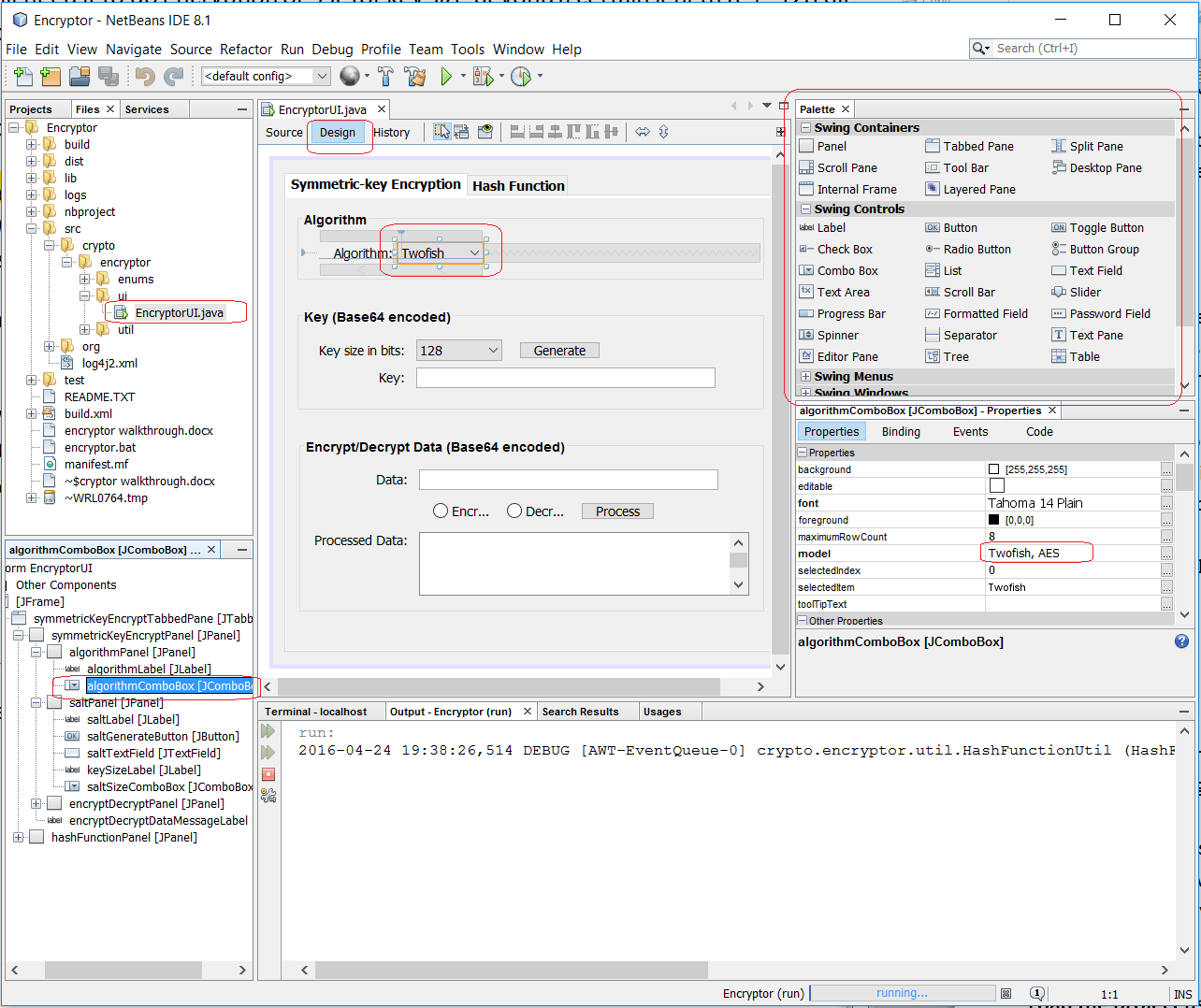
Now if a different IDE is used to edit the code, then all the above features may be broken if one tries to open the project again using NetBeans IDE. So don’t.

What if one needs to add custom event listener code for a component? This could be done easily by clicking the target component first in Design panel, right click to show drop down menu, select *Events 🡪 Action 🡪 actionPerformed*. This will jump to the Java code in Source panel directly to let you add custom code. See figures 4 and 5.

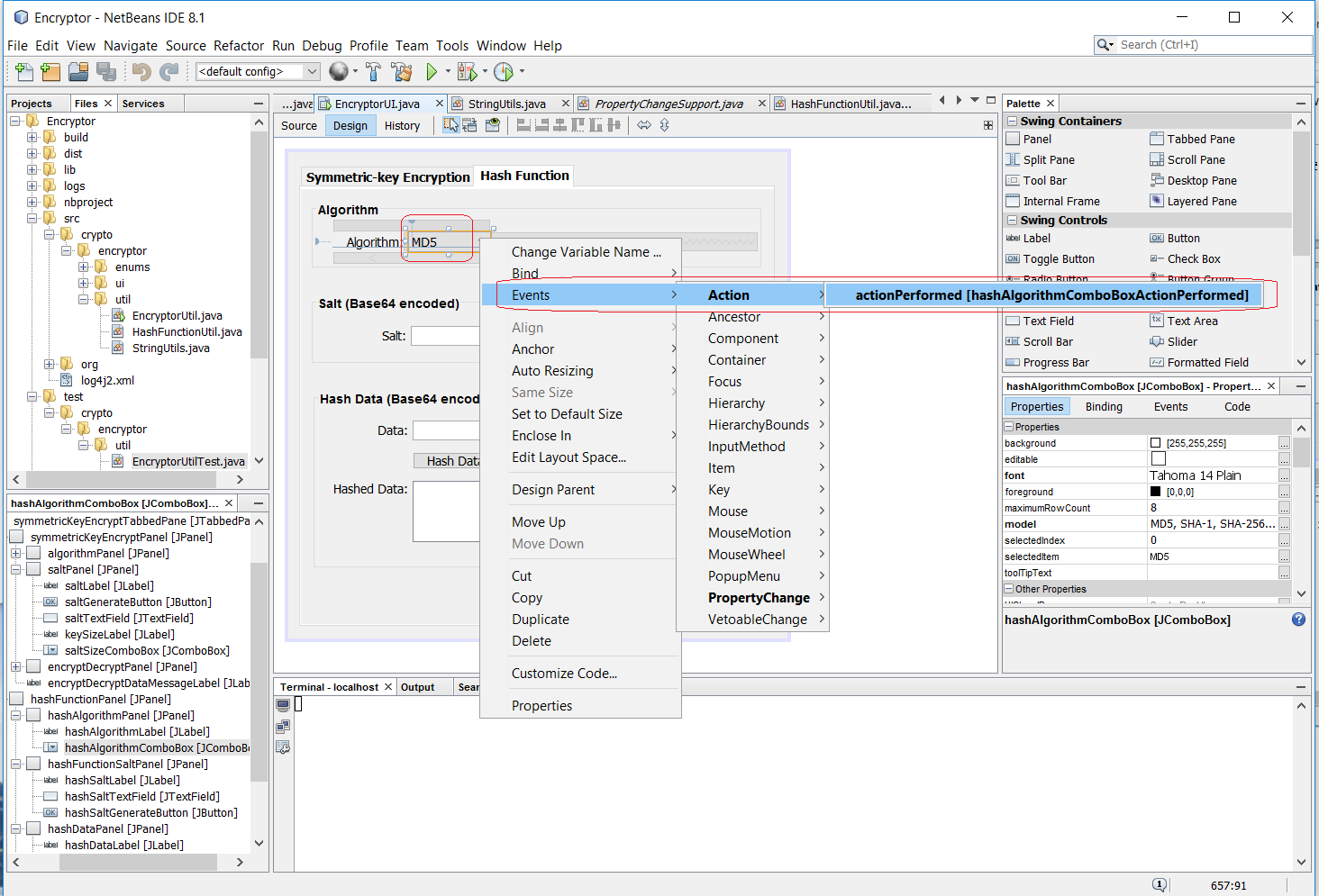
One thing I don’t like about the Swing support in NetBeans is that all the UI code is generated in the single JFrame class **EncryptorUI** (which launch the application). So a monolithic class is resulted. It would be nice to refactor EncryptorUI class into many smaller classes and retain the Swing IDE support. I will leave these for now.

The Swing support from NetBeans IDE can do a lot more but I will leave those for reader to explore for now. A quick google should answer most of your questions.

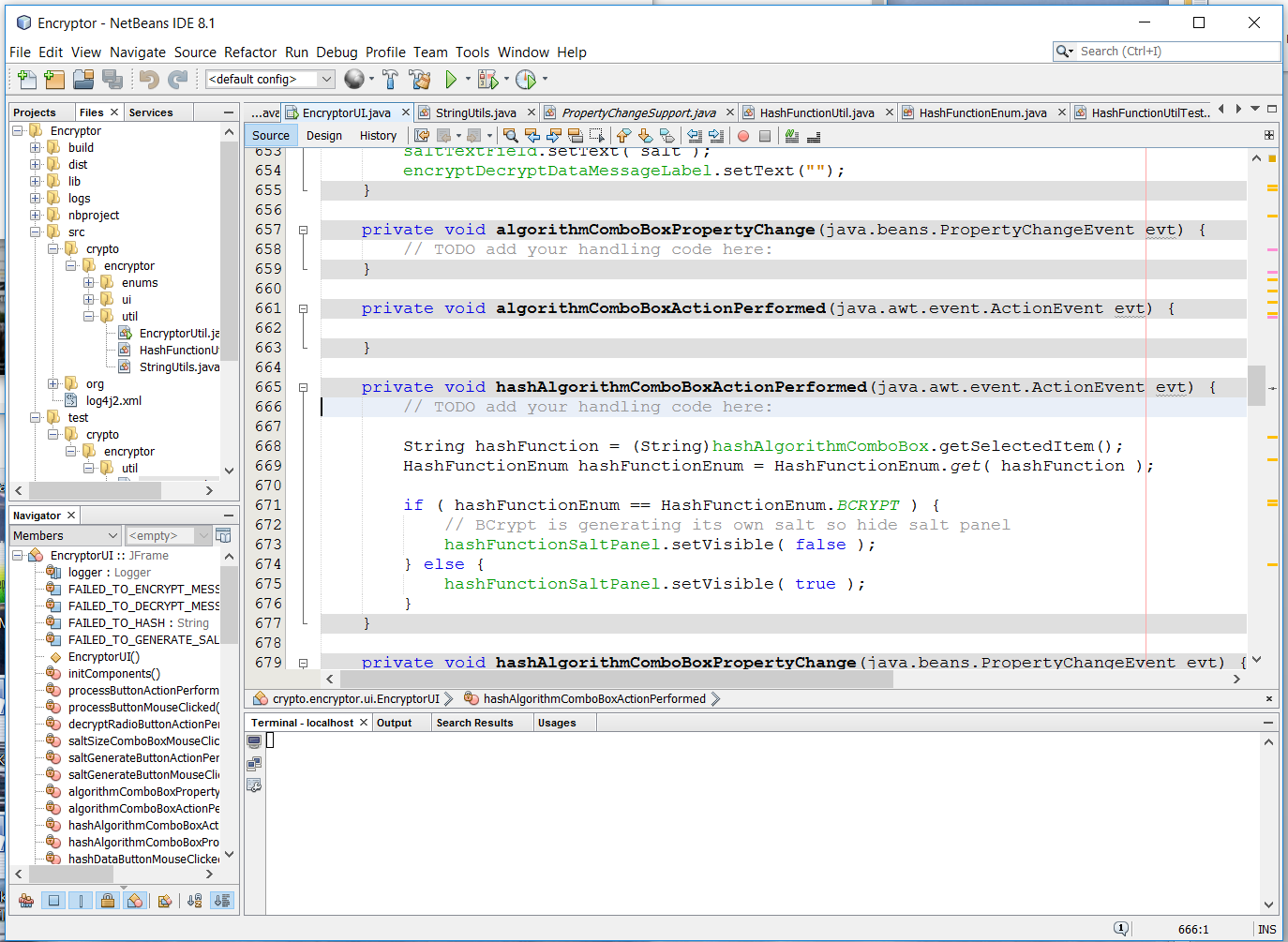
Of all the drag and drop tool for implementing a web or desktop application, this may be the best I have seen so far.



**Figure 3:** Design view



**Figure 4:** Jump to Source panel to add custom Event listener code



**Figure 5:** Add custom event listener code in Source panel

**How to clean, build, run, debug the project and the miscellaneous**

So how does one clean, build, run, or debug the project? Since a picture is worth a thousand words, see Figure 6.

The /*build.xml* and /*nbproject/build-impl.xml* files are the auto-generated ant build files from NetBeans. I haven’t changed anything in there. Try not to change anything in *nbproject* folder which contains auto generated configuration unless you know what you are doing. You may ask this is 2016, why Maven is not used to do the build? Well, everything seems to work out of the box so I don’t see the need to use Maven unless I need to manage a lot of third party Jar files.

Manage third party jar files

To use any third part jar files on classpath, drop the jar files in the ‘lib’ folder then right click the *Encryptor* folder in Files panel on the left, ‘Libraries’ from Categories panel 🡪 select ‘lib’ from Compile tab 🡪 Edit button 🡪 ‘Add Jar/Folder’ button 🡪 select the Jar files dropped in lib folder. See Figure 7 for details. The *./lib/nblibraries.properties* file will contain the changes of classpath for the jar files used in project.

Logging

Any logging of level ERROR in *crypto.encryptor* package will be logged in the log file at /logs/encryptor.log. Any logging of level DEBUG or above will be output to the console. The logging configuration is controlled in **log4j2.xml**. Since Log4j 2.5 is used, JDK 7 or above will be required to run Encryptor.

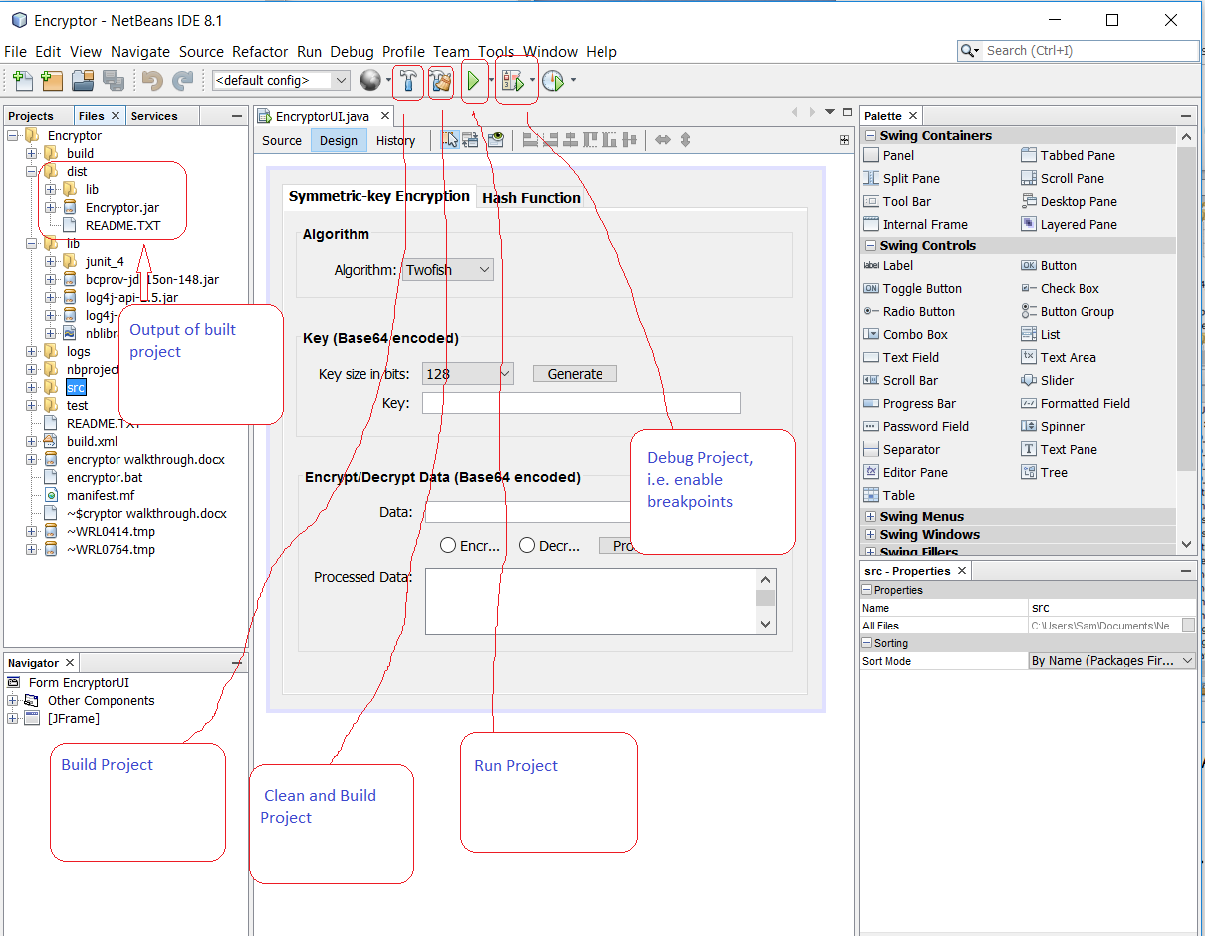


Figure 6: How to clean, build, run, or debug the project.

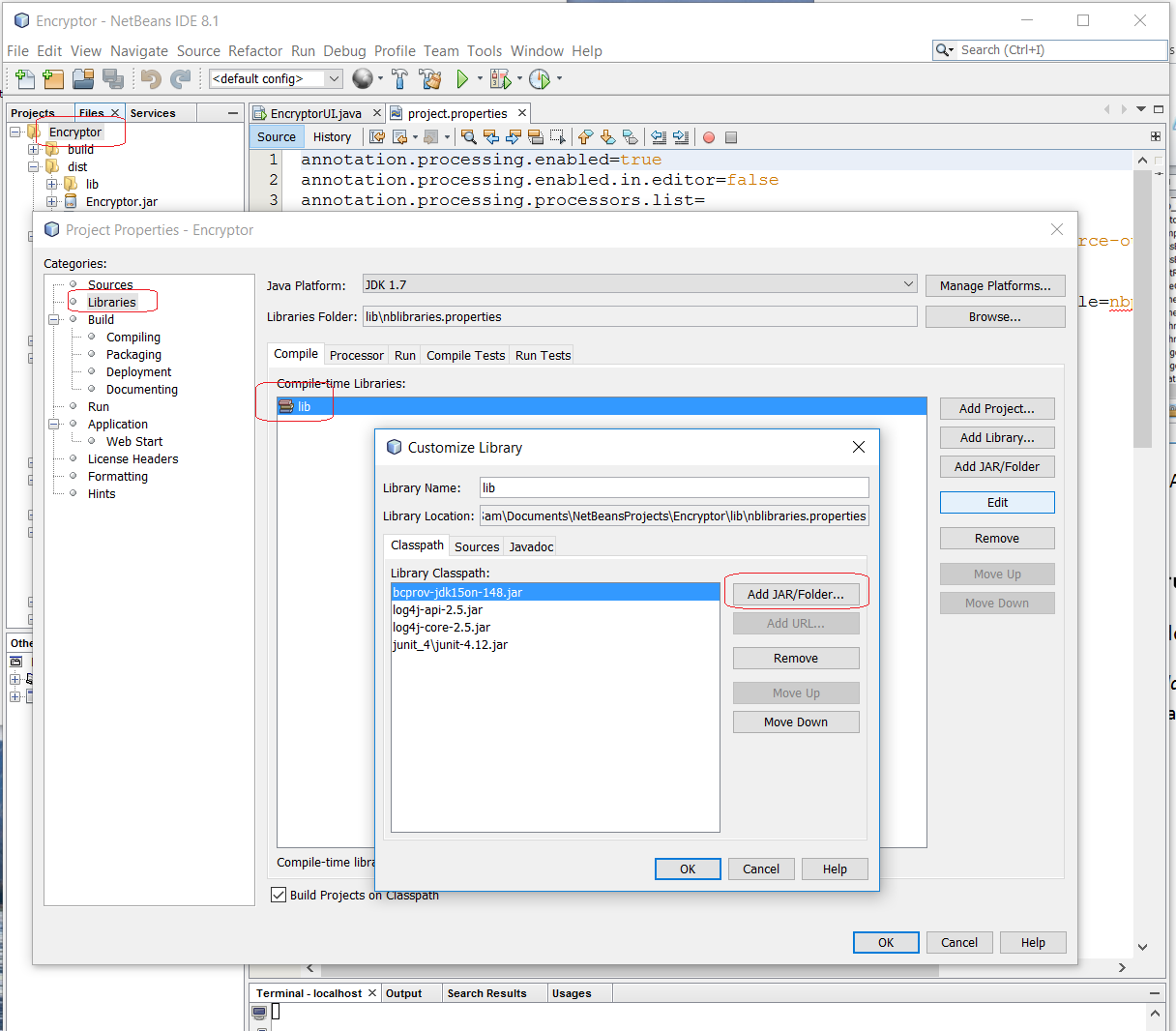


Figure 7: Add jar files from lib folder into classpath

**Encrypt, Decrypt and Hash-Hash**

The implementation of encrypting and decrypting data using AES and Twofish algorithms are greatly simplified in **EncryptorUtil** class by using the excellent open source **BouncyCastle** API from ***bcprov-jdk15on-148.jar***.

Hashing is implemented in the Util class **HashFunctionUtil** which also uses **BCrypt** class from [here](http://www.mindrot.org/projects/jBCrypt/#download) to implement the BCrypt function. See the links from ‘*Hashing food for thoughts’* comment in HashFunctionUtil class to complete one’s Jedi training in Hash function.

Note BCrypt hash function requires the generation of its own dedicated salt to work. This means BCrypt hash would only work with salt generated by the Generate button when the Algorithm ‘drop down list’ selects BCrypt, else error message will be shown at the bottom of panel.

All the generated salt, encryption key, encrypted, decrypted or hashed data are Base64 encoded string. Hex encoded string is not used since Base64 is more compact and efficient.

Why? Base64 encodes 3 bytes into 4 characters while Hex encodes 1 byte into 2 characters => Hex requires 2 more extra characters than Base64 to encode 3 bytes. Imagine the space saved for encrypting a few MB of data using Base64 encoding as opposed to Hex encoding.

***That’s the end of walkthrough. Hope you find it interesting and learned a few tricks!***