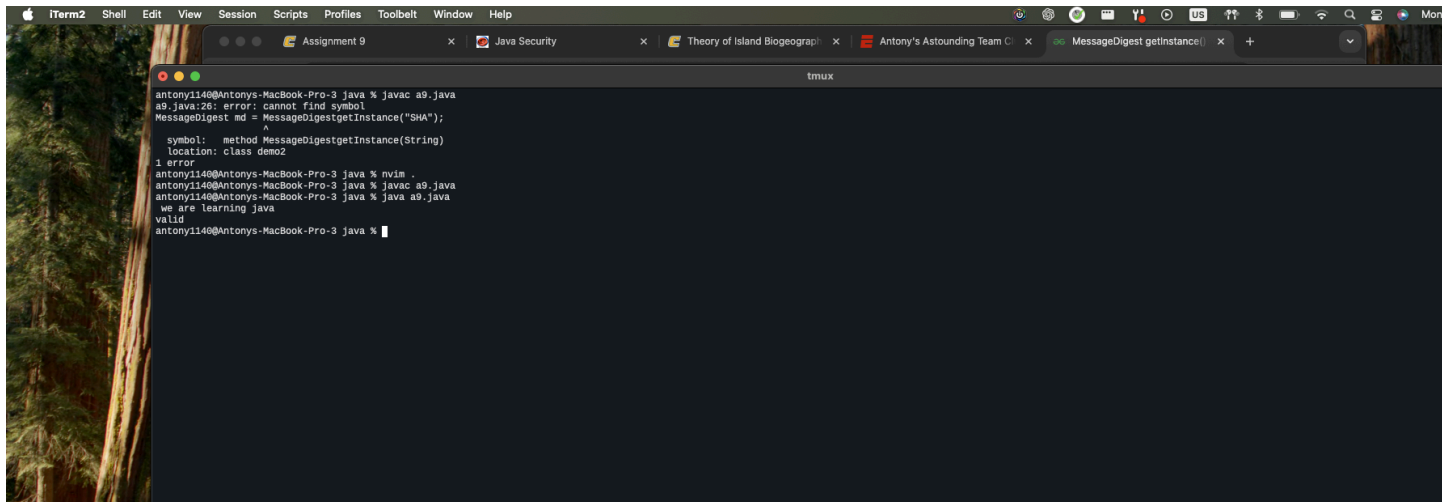


The image consists of two screenshots of a terminal window. The top screenshot shows a series of commands being executed in a tmux session. The user is in the directory `/Users/antony1140/utc/cs3600/java`. They run `zsh: permission denied: /Users/antony1140/utc/cs3600/java`, then `cd /Users/antony1140/utc/cs3600/java`, `nvim .`, `javac a9.java`, and `java a9.java`. The output shows "digest ready!" multiple times. The bottom screenshot shows the same terminal session with error messages. The first error is `a9.java:44: error: incompatible types: byte[] cannot be converted to String` at `String digest = md.digest();`. The second error is `a9.java:44: error: incompatible types: byte[] cannot be converted to byte` at `byte digest = md.digest();`. The third error is `a9.java:44: error: incompatible types: byte[] cannot be converted to String` at `String digest = md.digest();`. The output shows a long list of numbers and the final output `[80525f1e4e]` followed by `digest ready!`.

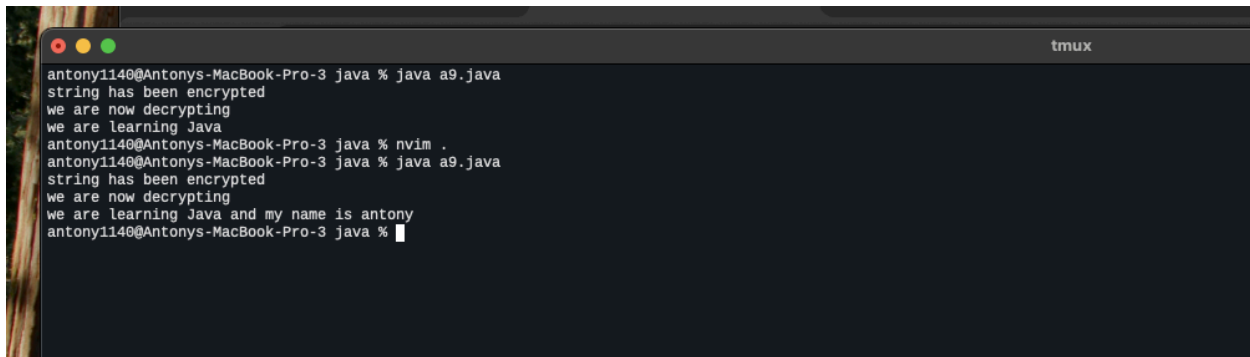
```
antony1140@Antony's-MacBook-Pro-3 ~ % /Users/antony1140/utc/cs3600/java
zsh: permission denied: /Users/antony1140/utc/cs3600/java
antony1140@Antony's-MacBook-Pro-3 ~ % cd /Users/antony1140/utc/cs3600/java
antony1140@Antony's-MacBook-Pro-3 java % nvim .
antony1140@Antony's-MacBook-Pro-3 java % javac a9.java
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
digest ready!
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
digest ready!
antony1140@Antony's-MacBook-Pro-3 java % nvim .
antony1140@Antony's-MacBook-Pro-3 java % javac a9.java
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
digest ready!
antony1140@Antony's-MacBook-Pro-3 java % nvim .
antony1140@Antony's-MacBook-Pro-3 java % javac a9.java
a9.java:44: error: incompatible types: byte[] cannot be converted to String
String digest = md.digest();
^
1 error
antony1140@Antony's-MacBook-Pro-3 java % nvim .
antony1140@Antony's-MacBook-Pro-3 java % javac a9.java
a9.java:44: error: incompatible types: byte[] cannot be converted to byte
byte digest = md.digest();
^
1 error
antony1140@Antony's-MacBook-Pro-3 java % nvim .
antony1140@Antony's-MacBook-Pro-3 java % javac a9.java
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
-38
57
-93
-18
94
107
75
13
50
85
-65
-17
-107
96
24
-112
-81
-40
7
9
[80525f1e4e
digest ready!
antony1140@Antony's-MacBook-Pro-3 java %
```

In demo 1, the program uses a SHA algorithm to encrypt the message. It writes the object to a file called `demo1test`. You can see in the first few runs, the output is “digest ready” to signal the write went as planned. The file is written with “”I. Im not sure what that means exactly, but I managed to modify the program to print out the byte array created by the digest method. And then it finally prints out the digest to see what it was doing under the hood. I changed the `s1` string to see if the file output was any different but it was still “”I.



```
antony1140@Antonys-MacBook-Pro-3 java % javac a9.java
a9.java:26: error: cannot find symbol
MessageDigest md = MessageDigestgetInstance("SHA");
                        ^
symbol:   method MessageDigestgetInstance(String)
location: class demo2
1 error
antony1140@Antonys-MacBook-Pro-3 java % nvim .
antony1140@Antonys-MacBook-Pro-3 java % javac a9.java
antony1140@Antonys-MacBook-Pro-3 java % java a9.java
we are learning java
valid
antony1140@Antonys-MacBook-Pro-3 java %
```

Demo 2 is a test of the validity or integrity of the algorithm and its keys. Its reading the object from the file we wrote to earlier and then printing it out as a string. It then creates another object identical to the one we got in the first step and encrypts it again with SHA. The comparison of the hashes then prompts the valid or not to be printed out. There is a lot of magic going on here making it slightly difficult to really tell what exactly is going on. This is a good thing of course if you know how to use the library making it easier to implement without critical mistakes that could easily be made rolling your own encryption and decryption.



```
antony1140@Antonys-MacBook-Pro-3 java % java a9.java
string has been encrypted
we are now decrypting
we are learning Java
antony1140@Antonys-MacBook-Pro-3 java % nvim .
antony1140@Antonys-MacBook-Pro-3 java % java a9.java
string has been encrypted
we are now decrypting
we are learning Java and my name is antony
antony1140@Antonys-MacBook-Pro-3 java %
```

```
antony1140@Antony's-MacBook-Pro-3 java % javac a9.java
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
string has been encrypted
60
35
93
100
-88
47
-44
8
-54
47
-48
-97
93
48
60
5
37
76
23
66
-41
-85
113
106
-15
-62
-89
-41
-77
-44
38
-112
-51
85
-5
-13
-101
-21
-61
-125
-69
-12
-96
-38
-118
8
-68
12
[B@60704c
we are now decrypting
we are learning Java and my name is antony
```

```
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
string has been encrypted
-113
-116
-124
-107
-22
-32
-17
-80
-57
-27
100
-76
102
121
-24
89
-127
-32
-127
44
-104
92
-18
-108
-13
-121
74
11
36
26
-112
69
-57
-18
110
67
-124
-27
-83
89
79
78
-25
107
-67
48
-80
115
[B@23f7d05d
we are now decrypting
we are learning Java and my name is antony
antony1140@Antony's-MacBook-Pro-3 java %
```

In Demo 3, we are using a different algorithm. But to make it more secure, an initialization vector is used for the process of encrypting the text. This initvector makes the cipher more secure as I understand it. It adds a layer of complexity if it's a different random value each time making the same plain text encrypt to a different cipher text each time hiding potential patterns that could be used against the user. I ran the program several times to make sure that it works with different values. I then modified the program to print out the encrypted array of bytes and running it twice with the same plain text shows different values as expected with an IV.

```
antony1140@Antony's-MacBook-Pro-3 java % java a9.java
SUN version 17
Signature.SHA3-384withDSA ImplementedIn
Alg.Alias.MessageDigest.SHA512/224
Signature.SHA384withDSA KeySize
Alg.Alias.KeyPairGenerator.1.2.840.10040.4.1
CertPathValidator.PKIX
```

Demo 4 just prints out a long list of security providers available in the java security library. I have java 17 on my computer so my list is current as of the latest version of that.

```

antony1140@Antonys-MacBook-Pro-3 java % javac a9.java
antony1140@Antonys-MacBook-Pro-3 java % java a9.java
public & private keys ready!
the data signed by private key
now verifying with public key
authentic..ok
antony1140@Antonys-MacBook-Pro-3 java % █

```

Demo 5 is the first program to show the key generation. Up till now, the encryption and decryption was done without the need to explicitly create keys. This involves more steps than the others but it removes a layer or two of abstraction away revealing just a little more to the user. Of course, the inner workings are all still very much hidden and a lot of magic is involved for obvious reasons. This demo also demonstrated the signing of data to make sure it is authentic by the receiver.

```

antony1140@Antonys-MacBook-Pro-3 java % javac a9.java
Note: a9.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
antony1140@Antonys-MacBook-Pro-3 java % java a9.java
Note: a9.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
enter the password
antony1140password
enter the datastring
antony1140string
encryption over
now decrypting
enter the password again!
antony1140password
antony1140string
antony1140@Antonys-MacBook-Pro-3 java % java a9.java
Note: a9.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
enter the password
antony1140password
enter the datastring
antony1140string
encryption over
now decrypting
enter the password again!
anotherpassword
javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
antony1140@Antonys-MacBook-Pro-3 java % █

```

Demo 6 uses a password based system to create keys. It appears to be a symmetrical system which isn't the most secure form of cryptography. Needing a password helps the security however in this context. I ran the program twice. Once with the proper password both times and the decryption was a success. The second time, I entered mismatched passwords. This time, as expected, the decryption didn't work out as well.