

# Secure Communication in Java

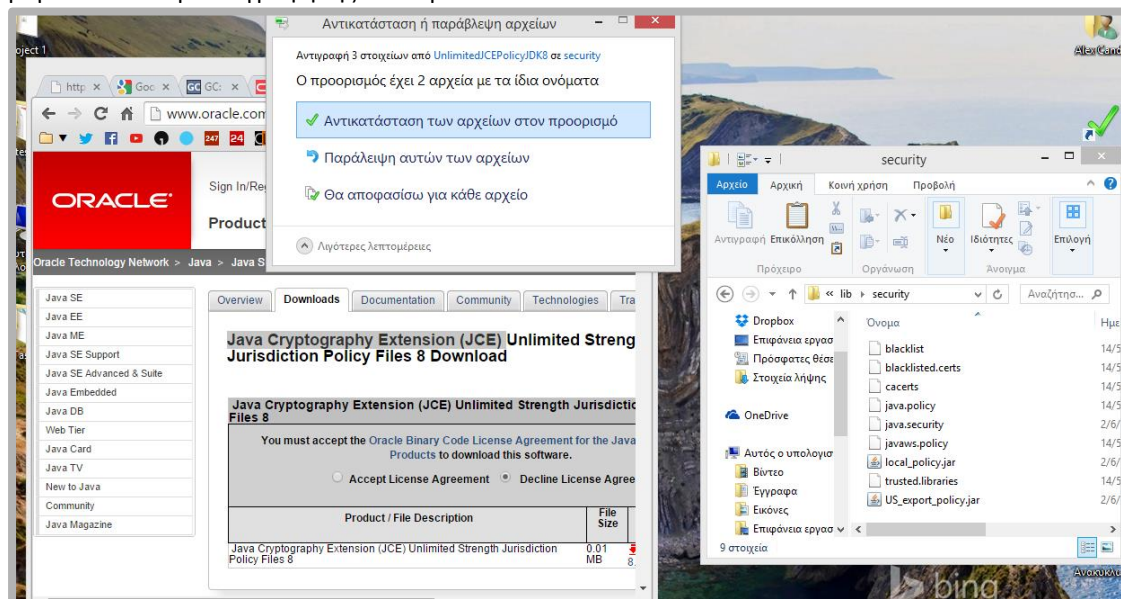
Developed by Alexandros Kantas

## Περιεχόμενα

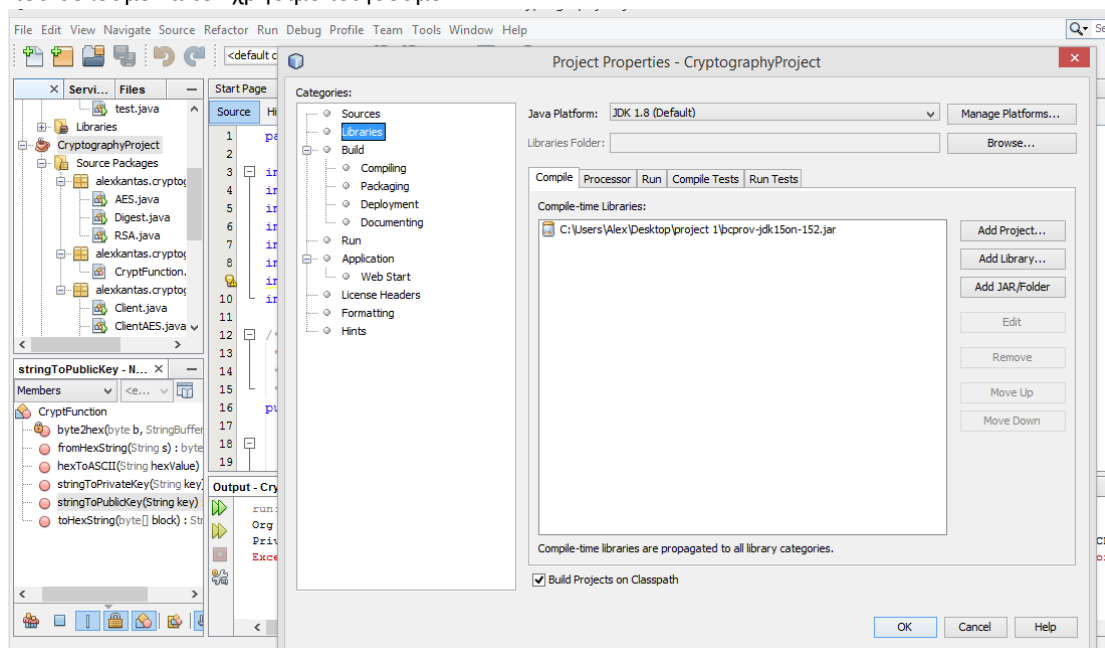
<b>Πληροφορίες Εγκατάστασης Βιβλιοθηκών και αρχείων.....</b>	<b>2</b>
<b>Digest.....</b>	<b>3</b>
Code key points .....	3
More info.....	3
<b>Συμμετρική κρυπτογραφία .....</b>	<b>4</b>
<b>Ασύμμετρη κρυπτογραφία .....</b>	<b>5</b>
Sample code .....	5
More code .....	5
<b>Ασφαλής επικοινωνία με AES.....</b>	<b>6</b>
Επικοινωνία χωρίς κρυπτογράφηση .....	6
Κώδικας Server .....	6
Κώδικας Client .....	7
Επικοινωνία με κρυπτογράφηση .....	8
Κώδικας .....	8
<b>Cool experiments.....</b>	<b>9</b>
Chat χωρίς κρυπτογράφηση .....	9
Chat με κρυπτογράφηση.....	10
<b>Ασφαλής επικοινωνία με RSA.....</b>	<b>12</b>

## Πληροφορίες Εγκατάστασης Βιβλιοθηκών και αρχείων

Αντικαθιστούμε τα *Security Policy Files* στο με τα *Java Cryptography Extension* ώστε να έχουμε όσο μεγάλο κλειδί κρυπτογράφησης θέλουμε.



Επιλέγουμε να εγκαταστήσουμε δυναμικά το Bouncy Castle provider. Για να γίνει αυτό ενημερώνουμε τις βιβλιοθήκες του NetBeans και δηλώνουμε ένα νέο αντικείμενο *BouncyCastleProvider()* κάθε φορά που θέλουμε να τον χρησιμοποιήσουμε.



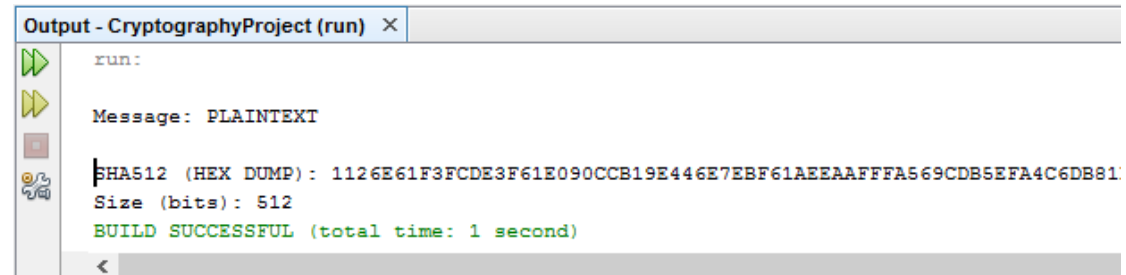
## Digest

### Code key points

Δημιουργήσαμε ένα νέο αντικείμενο `Digest` και με την μέθοδο `Calc` υπολογίζεται η σύνοψη της συμβολοσειράς.

```
Digest digest = new Digest();  
byte[] randomDataDigest = digest.Calc("Alex");
```

Παρατηρούμε ότι η συμβολοσειρά "Alex" έχει το παρακάτω *SHA512 digest*



### More info

Η κλάση `Digest` που δημιουργήσαμε περιέχει ένα `MessageDigest` και στον constructor ορίζεται ότι default θα χρησιμοποιείτε `SHA512` αλγόριθμος

```
public class Digest {  
  
    private MessageDigest message;  
  
    public Digest() {  
        setMessageAlgorithm("SHA512");  
    }  
}
```

Στη κλάση έχουν δημιουργηθεί οι κατάλληλες μέθοδοι για την αλλαγή του default αλγορίθμου καθώς και για τον υπολογισμό του digest από δοθέν string ή byte array. Κάθε μέθοδος περιέχει Javadoc τεκμηρίωση.

## Συμμετρική κρυπτογραφία

Δημιουργούμε ένα αντικείμενο AES και καλούμε την μέθοδο generate key

```
AES aes = new AES();  
aes.generateKey();
```

την οποία έχουμε ορίσει να δημιουργεί ένα AES 256-bits κλειδί

```
public void generateKey() throws NoSuchAlgorithmException {  
    KeyGenerator AES_keygen = KeyGenerator.getInstance("AES");  
    AES_keygen.init(256, new SecureRandom());  
    SecretKey AES_key = AES_keygen.generateKey();  
    this.AES_key = AES_key;  
}
```

Στον constructor της AES δηλώνουμε τον Bouncycastle provider (αφού χρησιμοποιούμε δυναμικό input) και ορίζουμε το Cipher text να χρησιμοποιεί ECB mode με PKCS5 Padding

```
public AES() throws Exception {  
    Security.addProvider(bcp); // using the Bouncycastle provider  
    AES_Cipher = Cipher.getInstance("AES/ECB/PKCS5Padding", "BC"); }
```

Η AES επίσης περιλαμβάνει μεθόδους encrypt και decrypt οι οποίες δέχονται ορίσματα string ή byte arrays για κρυπτογράφηση και αποκρυπτογράφηση αντίστοιχα.

Παρακάτω ένας κώδικας πειραματισμού της χρήσης του AES. Δηλώνουμε το plaintext, το μέγεθος του plaintext και το digest του κλειδιού κρυπτογράφησης

```
String plaintext = "Hello Alex !!!"; //Plaintext  
int plaintextSize = plaintext.getBytes().length; //Plaintext real size  
byte[] keydigest = digest.Calc(aeskey.getEncoded());
```

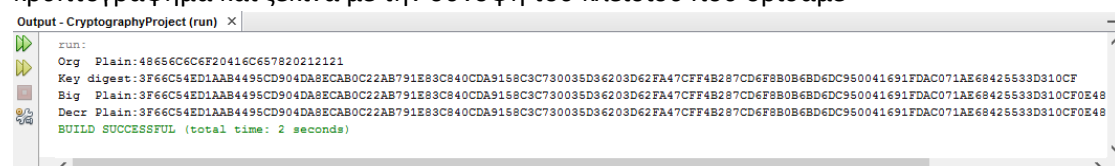
Όλα αυτά τα αποθηκεύουμε σε ένα byte array με τον παρακάτω κώδικα

```
outputStream.write(keydigest);  
outputStream.write(plaintextSize);  
outputStream.write(plaintext.getBytes());  
  
byte[] bigplain = outputStream.toByteArray(); //Plaintext with  
key digest and length in front
```

Κρυπτογραφούμε το νέο μας μεγάλο plaintext με την μέθοδο *aes.encrypt* και το αποκρυπτογραφούμε με την *aes.decrypt*.

```
byte[] ciphertext = aes.encrypt(bigplain);  
byte[] dplain = aes.decrypt(ciphertext); //Decrypted plain text
```

Εκτελούμε το πρόγραμμα και παρατηρούμε ότι το αποκρυπτογράφημα είναι ίδιο με το κρυπτογράφημα και ξεκινά με την σύνοψη του κλειδιού που ορίσαμε



```
run:  
Org Plain: 48656C6CF20416C657820212121  
Key digest: 3F66C54ED1AAB4495CD904DA8ECAB0C22AB791E83C840CDA9158C3C730035D36203D62FA47CFF4B287CD6F8B0B6BD6DC950041691FDAC071AE68425533D310CF  
Big Plain: 3F66C54ED1AAB4495CD904DA8ECAB0C22AB791E83C840CDA9158C3C730035D36203D62FA47CFF4B287CD6F8B0B6BD6DC950041691FDAC071AE68425533D310CF0E48  
Decr Plain: 3F66C54ED1AAB4495CD904DA8ECAB0C22AB791E83C840CDA9158C3C730035D36203D62FA47CFF4B287CD6F8B0B6BD6DC950041691FDAC071AE68425533D310CF0E48  
BUILD SUCCESSFUL (total time: 2 seconds)
```

Για λόγους συντομίας τα println παραλήφθηκαν παραπάνω αλλά υπάρχουν κανονικά στο AES.java που συνοδεύει την παρούσα αναφορά

## Ασύμμετρη κρυπτογραφία

### Sample code

Με παρόμοιο τρόπο υλοποιούμε τον παραπάνω πειραματικό κώδικα με RSA με την χρήση του αντικειμένου που RSA δημιουργήσαμε.

Κρυπτογραφούμε το τελικό plaintext με την μέθοδο *rsa.encrypt* χρησιμοποιώντας το ιδιωτικό κλειδί .

```
byte[] bigplain = outputStream.toByteArray();  
byte[] ciphertext = rsa.encrypt(bigplain, prkey);
```

Αποκρυπτογραφούμε το ciphertext με την μέθοδο *rsa.decrypt* χρησιμοποιώντας το δημόσιο κλειδί

```
byte[] dplain = rsa.decrypt(ciphertext, pukey);
```

Η κλάση AES πάντα δημιουργεί ένα δικό της ζεύγος κλειδιών και οι μέθοδοι encrypt-decrypt δουλεύουν χωρίς την απαίτηση για input κλειδιών. Αυτό γίνεται καθαρά για λόγους πειραματισμού. Το προτεινόμενο συντακτικό είναι το παρακάτω

```
byte[] ciphertext = rsa.encrypt(bigplain);  
byte[] dplain = rsa.decrypt(ciphertext);
```

### More code

Ο constructor της RSA δημιουργεί μια γεννήτρια κλειδιών η οποία αρχικοποιήσεις για κλειδί 2048 bits. Το ζεύγος κλειδιών που δημιουργεί αποθηκεύεται στην *RSA\_KeyPair* και χρησιμοποιείτε στις μεθόδους του *RSA()*.

```
public RSA() throws Exception {  
    kpGen = KeyPairGenerator.getInstance("RSA");  
    kpGen.initialize(2048, new SecureRandom());  
    RSA_KeyPair = kpGen.genKeyPair();  
    c = Cipher.getInstance("RSA/None/PKCS1Padding");}
```

Ενδεικτικά η μέθοδος encrypt που παίρνει όρισμα ένα byte array. Το cipher ορίζεται ότι θα λειτουργεί σε λειτουργία κρυπτογράφησης και γίνεται χρήση του ιδιωτικού κλειδιού. Βάσει αυτού με την χρήση της doFinal παίρνουμε το κρυπτογραφημένο μήνυμα.

```
public byte[] encrypt(byte[] plaintext) throws Exception {  
    c.init(Cipher.ENCRYPT_MODE, RSA_KeyPair.getPrivate());  
    byte[] ciphertext = c.doFinal(plaintext);  
    return ciphertext;  
}
```

# Ασφαλής επικοινωνία με AES

## Επικοινωνία χωρίς κρυπτογράφηση

### Κώδικας Server

Για την επικοινωνία του Server μας με έναν client ξεκινάμε δημιουργώντας ένα `ServerSocket` και να ορίζοντας σε ποια θύρα τρέχει. Αποδεχόμαστε την αίτηση του client να συνδεθεί μαζί μας. Ορίζουμε `PrintWriter` και `BufferedReader` για την διαχείριση των input και output streams της γραμμής αλλά και τον input του πληκτρολογίου.

```
try ( //Using try-with-resources statement
    ServerSocket serverSocket = new ServerSocket(portNumber);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
    true);
    BufferedReader in = new BufferedReader(new
    InputStreamReader(clientSocket.getInputStream()));
    BufferedReader stdIn = new BufferedReader(new
    InputStreamReader(System.in)); )
```

Το εν λόγω συντακτικό λέγεται try με resources και μας απαλλάσσει από την ανάγκη να καλούμε την `.close` μέθοδο από κάθε resource καθώς πλέον γίνεται αυτόματα με το τέλος της try.

Με το που γίνει η σύνδεση αποστέλνεται το string "ON." . Στην συνέχεια κάθε φορά που υπάρχει input stream εμφανίζεται στην κονσόλα. Μετά έρχεται η σειρά μας να στείλουμε ένα output stream. Έχουμε γράψει το πρόγραμμα έτσι ώστε να τερματίζει όταν δεχτεί input το "OFF."

```
String inputLine, outputLine;
System.out.println("Server Running"); //Run server logic
out.println("ON.");
while ((inputLine = in.readLine()) != null) {
    if (inputLine.equals("OFF.")) {
        break;
    }
    System.out.println("Server receive : " + inputLine);
    System.out.println("type message :");
    outputLine = stdIn.readLine();
    out.println(outputLine);
}
}
```

## Κώδικας Client

Η κύρια διαφορά στον κώδικά του client είναι ότι επιλέγουμε που σε ποια ip να συνδεθούμε κατά την δημιουργία του Socket

```
String ip = "192.168.1.68";
int portNumber = 4433;

try (
    Socket mySocket = new Socket(ip, portNumber);
    PrintWriter out = new
PrintWriter(mySocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new
InputStreamReader(mySocket.getInputStream()));) {

    BufferedReader stdIn =
newBufferedReader(newInputStreamReader(System.in));

    String fromServer;
    String fromUser;

    System.out.println("Client run");
    while ((fromServer = in.readLine()) != null) {
        System.out.println("Server: " + fromServer);
        if (fromServer.equals("OFF.")) {
            break;
        }

        fromUser = stdIn.readLine();
        if (fromUser != null) {
            System.out.println("Client: " + fromUser);
            out.println(fromUser);
        }
    } catch (IOException ex) {

        System.err.println("Couldn't get I/O for the connection to " + ip);
        System.exit(1);
    }
}
```

Στον παραπάνω κώδικα δοκιμάσαμε να συνδεθούμε με μια τοπική ip ενός άλλου pc που τρέχει το Server. Παρακάτω στην αναφορά υπάρχουν Screenshots από τα μηχανήματα να τρέχουν.



## Επικοινωνία με κρυπτογράφηση

### Κώδικας

Για την κρυπτογραφημένη επικοινωνία χρησιμοποιούμε την κλάση AES() που φτιάξαμε. Δίνουμε με String ένα συγκεκριμένο κλειδί στον Server ( το οποίο θα είναι ίδιο με του client ).

```
int portNumber = 4433;
AES aes = new AES();
String key = "414C4558414E44524F53204B414E54415320416C6578204B616E";
```

Στην συνέχεια ορίζουμε ότι θα γίνεται κρυπτογράφηση με αυτό το κλειδί καλώντας την μέθοδο aes.setkey. Πριν από κάθε output κρυπτογραφούμε το stream με την aes.encrypt ενώ σε κάθε input το αποκρυπτογραφούμε με την aes.decrypt

```
String inputLine, outputLine, decryptedinput, encryptedoutput;
aes.setkey(key);
System.out.println("Server Running"); //Run server logic
out.println(aes.encrypt("ON."));
while ((inputLine = in.readLine()) != null) {
    decryptedinput = aes.decrypt(inputLine);
    decryptedinput = alg.hexToASCII(decryptedinput);
    if (decryptedinput.equals("OFF.")) {
        break;
    }
    System.out.println("Server receive : " + decryptedinput);
    System.out.println("type message :");
    outputLine = stdIn.readLine();
    encryptedoutput = aes.encrypt(outputLine);
    out.println(encryptedoutput);
}
```

Με παρόμοια λογική έχει υλοποιηθεί και ο Client. Επόμενο βήμα είναι να κάνουμε chat να δούμε αν δουλεύει η κρυπτογράφηση.

## Cool experiments

### Chat χωρίς κρυπτογράφηση

Στήσαμε στο ένα μηχανήμα τον server και στο άλλο τον client και δοκιμάσαμε να μιλήσουμε χωρίς κρυπτογράφηση. Ανοίξαμε το wireshark και κάναμε ανάλυση των πακέτων που μεταδίδονται στο δίκτυο. Βλέπουμε ότι καταφέραμε επιτυχώς να πάρουμε την συνομιλία μας.

The screenshot displays the NetBeans IDE 8.0 environment with a project named 'CryptographyProject'. The 'main - Navigator' on the left shows a 'Server' class with a 'main(String[] args)' method. The 'Output - CryptographyProject (run)' window at the bottom shows the server's execution log:

```
run:
Server Running
Server receive : 00 00 00 00'00 00 00 Hello Giorgos
type message :
hello alex
Server receive : Where is Giannis ?
type message :
our am are 3141 3147 and 3025
OFF.
Server receive :
```

The 'Follow TCP Stream (tcp.stream eq 24)' window shows the stream content:

```
.....ON.
Hello Giorgos
hello alex
where is Giannis ?
our am are 3141 3147 and 3025
OFF.
OFF.
```

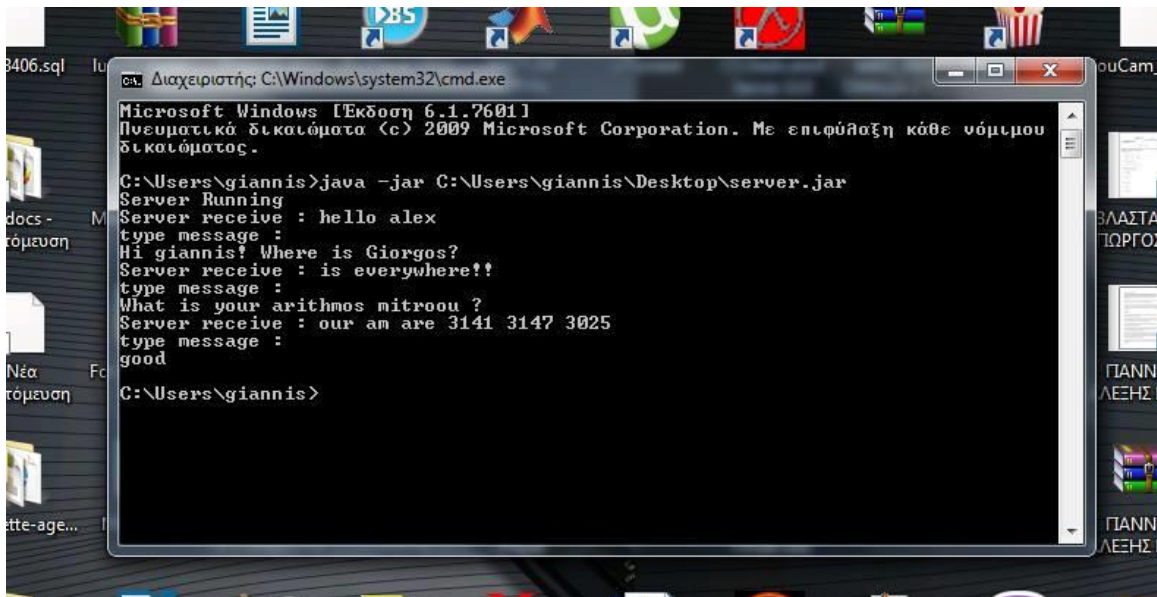
The 'Wireshark 1.12.5' window shows a list of captured packets on the 'Wi-Fi' interface. The filter is set to 'tcp.stream eq 24'. The packet list shows the following data:

No.	Time	Source	Destination	Protocol	Length	Info
615	18.747570000	192.168.1.68	192.168.1.65	TCP	66	60440→4433 [SYN
619	19.050601000	192.168.1.65	192.168.1.68	TCP	66	4433→60440 [SYN
620	19.055491000	192.168.1.68	192.168.1.65	TCP	54	60440→4433 [ACK
621	19.055914000	192.168.1.68	192.168.1.65	TCP	75	60440→4433 [PSH
622	19.059565000	192.168.1.65	192.168.1.68	TCP	59	4433→60440 [PSH
623	19.263089000	192.168.1.68	192.168.1.65	TCP	54	60440→4433 [ACK
737	45.008723000	192.168.1.68	192.168.1.65	TCP	67	60440→4433 [PSH
738	45.008937000	192.168.1.68	192.168.1.65	TCP	56	60440→4433 [PSH
739	45.009001000	192.168.1.65	192.168.1.68	TCP	54	4433→60440 [ACK
750	55.858321000	192.168.1.65	192.168.1.68	TCP	66	4433→60440 [PSH
751	56.060317000	192.168.1.68	192.168.1.65	TCP	54	60440→4433 [ACK
802	78.911380000	192.168.1.68	192.168.1.65	TCP	72	60440→4433 [PSH
803	78.911800000	192.168.1.68	192.168.1.65	TCP	56	60440→4433 [PSH

The status bar at the bottom indicates 'Packets: 979 · Displayed: 29 (3,0%) · Dropped: 0 (0,0%)'. The system clock shows 4:34 πμ on 12/6/2015.

## Chat με κρυπτογράφηση

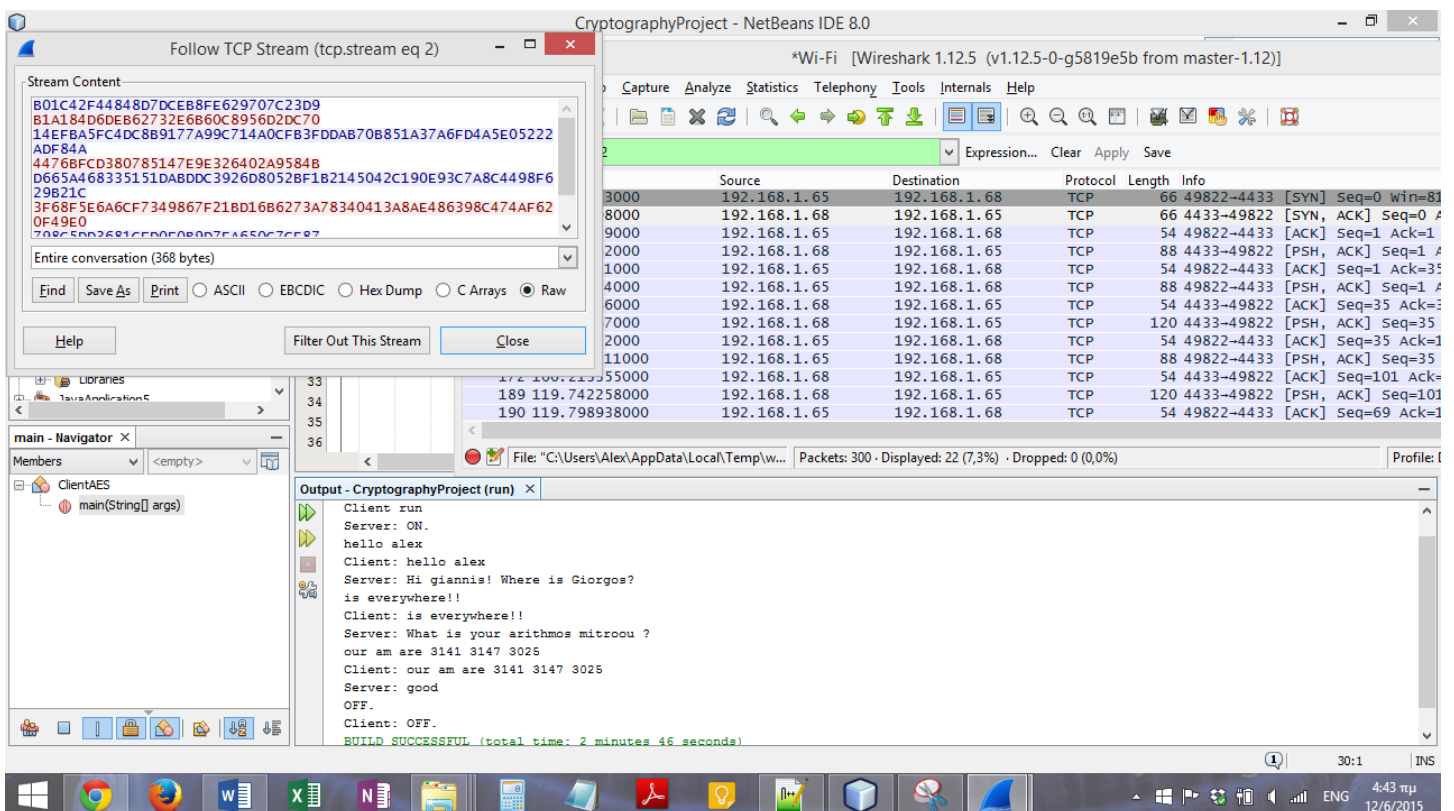
Στη συνέχεια κάναμε το ίδιο με την χρήση των κλάσεων για κρυπτογραφημένη επικοινωνία μεταξύ client και Server



```
Microsoft Windows [Έκδοση 6.1.7601]
Πνευματικά δικαιώματα (c) 2009 Microsoft Corporation. Με επιφύλαξη κάθε νόμιμου
δικαιώματος.

C:\Users\giannis>java -jar C:\Users\giannis\Desktop\server.jar
Server Running
Server receive : hello alex
type message :
Hi giannis! Where is Giorgos?
Server receive : is everywhere!!
type message :
What is your arithmos mitroou ?
Server receive : our am are 3141 3147 3025
type message :
good
C:\Users\giannis>
```

Δοκιμάσαμε πάλι να καταγράψουμε τα πακέτα που κινούνται στο δίκτυο με το wireshark. Αυτή την φορά πήραμε μόνο κρυπτογραφημένα strings.



Stream Content

```
B01C42F44848D7DCEB8FE629707C23D9
B1A184D6DEB62732E6860C8956D2DC70
14EFBA5FC4DC8B9177A99C714A0CFB3FDDAB70B851A37A6FD4A5E05222
ADF84A
4476BFC3D380785147E9E326402A9584B
D665A468335151DABDDC3926D8052BF1B2145042C190E93C7A8C4498F6
29821C
3F68F5E6A6CF7349867F21BD16B6273A78340413A8AE486398C474AF62
0F49E0
709C5D03681CF00C080D7E4650C7C687
```

Entire conversation (368 bytes)

Find Save As Print ASCII EBCDIC Hex Dump C Arrays Raw

Filter Out This Stream Close

Output - CryptographyProject (run)

```
Client run
Server: ON.
hello alex
Client: hello alex
Server: Hi giannis! Where is Giorgos?
is everywhere!!
Client: is everywhere!!
Server: What is your arithmos mitroou ?
our am are 3141 3147 3025
Client: our am are 3141 3147 3025
Server: good
OFF.
Client: OFF.
BUILD SUCCESSFUL (total time: 2 minutes 46 seconds)
```

Κάναμε copy τα κρυπτογραφημένα strings και τα δοκιμάσαμε σε ένα online AES decrypter χρησιμοποιώντας το κλειδί που ξέρουμε για να επαληθεύσουμε ότι η κρυπτογράφηση δουλεύει σωστά.

The screenshot shows a web browser window with the URL `aes.online-domain-tools.com`. The tool interface includes the following elements:

- Input type:** A dropdown menu set to "Text".
- Input text (hex):** A text area containing a long hexadecimal string: `801C42F44848D7DCEB8FE629707C23D981A184D6DE862732E6860C8956D2DC7014EFBA5FC4DC8B9177A99C714A0CFB3FDDA870B851A37A6FD4A5E0522ADF84A4476BFC380785147E9E326402A9584BD665A468335151DABDDC3926D8052BF1B2145042C190E93C7A8C4498F629B21C3F68F5E6A6CF7349867F21BD16B6273A78340413A8AE486398C474AF620F49E0798C5DD3681CFD0F089D7EA650C7CE87184B489228F8782A87FF8D24A88CEA8B`.
- Autodetect:** A toggle switch set to "OFF".
- Function:** A dropdown menu set to "AES".
- Mode:** A dropdown menu set to "ECB (electronic codebook)".
- Key (hex):** A text input field containing the key: `414C4558414E44524F53204B414E54415320416C6578204B616E746173202041`.
- Buttons:** Two green buttons labeled "> Encrypt!" and "> Decrypt!".
- Decrypted text:** A section showing the result of the decryption. It includes a hex-to-ASCII conversion table and the resulting text: `ON...hello alex... Hi giannis! Where is Giorgos? ... is everywhere!! What is your arithmos mitroou? our amare 3141 3147 3025... good...`.

A large black arrow points from the right side of the image towards the "Decrypted text" section.

Βλέπουμε ότι η αποκρυπτογράφηση έγινε σωστά

Ασφαλής επικοινωνία με RSA