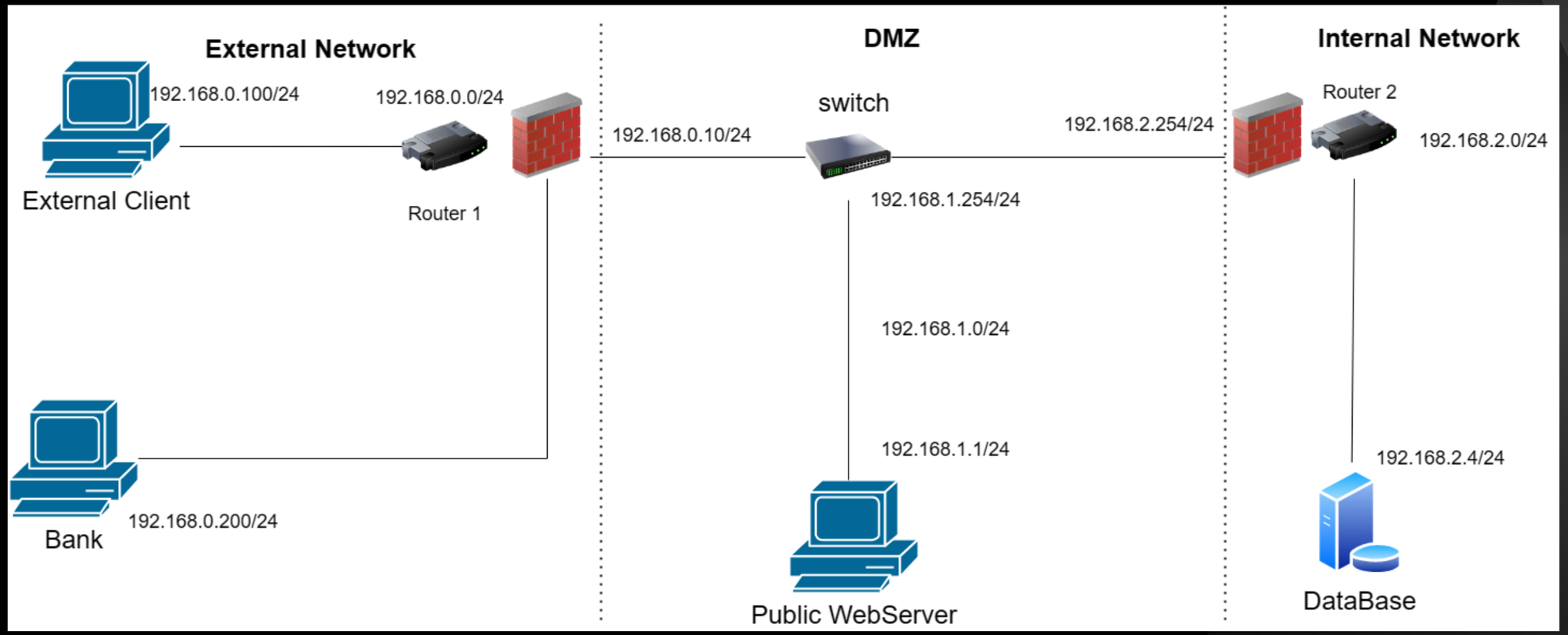
SIRS – The Cork

Group T48 – Presentation

- Bernardo Castiço ist196845
- Hugo Rita ist196870
- Pedro Pereira ist196905

Build Infrastructure



Secure Channels & Key Distribution

The secure channels are:

- Client – WebServer
- Bank – WebServer

The key distribution is the following:

- WebServer has a private/public key pair, being its public key known by the Bank.

Our security challenge

(ii) TheCork's users keep their credit card data information stored in the app in order to facilitate reservations at high–end restaurants that need you to leave a deposit. This data needs to be safe and confidential at the device level and on–the–wire when it gets sent to TheCork's servers and to the Credit Card company.

Develop Protocol (Client – WebServer)

In the communication between Client and WebServer we chose to implement the HTTPS protocol once it is one of the most known security protocols to confer privacy and security.

We generated a self signed certificate which was validated by a fictious CA.

Develop Protocol (Bank – WebServer)

The connection between Bank and WebServer is not secure so we designed a protocol inspired in TLS to protect it.

Our protocol follows the steps on the next slides.

Develop Protocol (Bank – WebServer)

I. Bank generates a pre master secret that will be send to the WebServer in a packet when establishing connection with it.

```
/*Generate pre master secret */
Long preMasterSecret = Math.round(Math.abs(Math.random()) * 1000000);
```

```
// Create request message
JsonObject requestJson = JsonParser.parseString("{}").getAsJsonObject();
{
    requestJson.addProperty("preMasterSecret", preMasterSecret);
    requestJson.addProperty("from", "Client");
    String bodyText = "Establish connection";
    requestJson.addProperty("info", bodyText);
}
```

Develop Protocol (Bank – WebServer)

II. The packet information is a Json Object encrypted with WebServer's public key using RSA algorithm. After encrypted this packet, it will be sent to WebServer by a socket.

```
try{
        key = readPublicKey(keyPath);
}catch(Exception e){
        System.out.println("Error reading server's public key");
}

//Encrypt with server's public key
try{
        cipherText = do_RSAEncryption(plainText, key);
} catch(Exception e){
        System.out.println("Error encrypting with server's public key");
}
```

```
public static byte[] do_RSAEncryption(String plainText,Key key) throws Exception
{
    Cipher cipher = Cipher.getInstance("RSA");

    cipher.init(Cipher.ENCRYPT_MODE, key);

    return cipher.doFinal(plainText.getBytes());
}
```

Develop Protocol (Bank – WebServer)

III. Once the WebServer receives the packet, it will decrypt it with its private key and RSA Algorithm to have access to the pre master secret.

```
public static String do_RSADecryption(byte[] cipherText, Key key)
{
    Cipher cipher = Cipher.getInstance("RSA");

    cipher.init(Cipher.DECRYPT_MODE, key);

    byte[] result = cipher.doFinal(cipherText);

    return new String(result);
}
```

Develop Protocol (Bank – WebServer)

IV. Then both Bank and WebServer will generate a session key by first hashing the pre master secret with the function SHA3–256 and then using the AES algorithm.

```
try{
        secretKeyinByte = digest(preMasterSecret.toString().getBytes(UTF_8), "SHA3-256");
} catch(Exception e){
```

```
/*Create secret key with AES algorithm */
SecretKey secretKey = new SecretKeySpec(secretKeyinByte, 0, secretKeyinByte.length, "AES");
```

Develop Protocol (Bank – WebServer)

V. From now on, the information sent between Bank and WebServer will be encrypted with the session key. This ensures confidentiality.

```
try{
        serverData = do_Encryption(responseJsonWhile.toString(), secretKey);
} catch(Exception e){
        System.out.println("Error encrypting with secret key");
}
```

Develop Protocol (Bank – WebServer)

VI. To ensure freshness, when the connection is established, the WebServer generates a token that is a random integer using the function Math.random(). This token will be present in the packets changed between Bank and WebServer to guarantee the freshness of the message.

```
static double tokenDouble = Math.round(Math.abs(Math.random()) * 1000000);
static Integer token = (int)tokenDouble;
```

```
JsonObject responseJson = JsonParser.parseString("{}").getAsJsonObject();
{
                JsonObject infoJson = JsonParser.parseString("{}").getAsJsonObject();
                infoJson.addProperty("token", token.toString());
```

Develop Protocol (Bank – WebServer)

VII. When a message is received, the receiver checks if the token is incremented by one in relation to the last token sent by the receiver.

```
//Check fressness of the message
if((token + 1) == Integer.parseInt(tokenRcvd)){
        token = Integer.parseInt(tokenRcvd);
        break;
}
else{
        returnValue = false;
        System.out.println("Not fresh request");
}
```

Develop Protocol (Bank – WebServer)

VIII. From now on the payload from WebServer to Bank contains the following information: Token, Credit Card number encrypted with the session key, 3digitCode encrypted with the session key and the validity date also encrypted with the session key.

```
// Create response message
JsonObject responseJsonWhile = JsonParser.parseString("{}").getAsJsonObject();
{
    JsonObject infoJson = JsonParser.parseString("{}").getAsJsonObject();
    infoJson.addProperty("token", token.toString());
    responseJsonWhile.add("info", infoJson);
    responseJsonWhile.addProperty("name", name);
    responseJsonWhile.addProperty("cardNumber", cardNumber64);
    responseJsonWhile.addProperty("threeDigits", threeDigits64);
    responseJsonWhile.addProperty("validityDate", validityDate64);
}
```

Develop Protocol (Bank – WebServer)

IX. To ensure integrity, we hash the payload with the SHA3–256 function to obtain the hmac that will be encrypted with the session key and sent alonside the payload. When the message arrives to the other side of the communication, the receiver must decrypt the hmac and then decrypt the payload and hash it.

If the two values match, the integrity is ensured in this message.

```
toSendMessage = { <payload> , <hmac_payload> }
<payload> = {token_freshness , encrypted info}
<hmac_payload> = cypher(hash(payload, SHA3-256), session key)
```

Develop Protocol (WebServer – Database)

In the communication between WebServer and Database, we prevent SQL injection by using prepared statements.

```
String query = "SELECT * FROM user_profile";
p = conn.prepareStatement(query);
```

Develop Protocol (WebServer – Database)

When sending the information from the WebServer to the Database we encrypt the sensible information from our security challenge with the session key and AES algorithm in order to it to be protected in the Database.

```
try{
        cardNumberByte = do_Encryption(cardNumber, secretKey);
        validityDateByte = do_Encryption(validityDate, secretKey);
        threeDigitsByte = do_Encryption(threedigits, secretKey);
}catch(Exception e){
        System.out.println(e);
}
String threeDigits64 = Base64.getEncoder().encodeToString(threeDigitsByte);
String cardNumber64 = Base64.getEncoder().encodeToString(cardNumberByte);
String validityDate64 = Base64.getEncoder().encodeToString(validityDateByte);


try{
        p = conn.prepareStatement("INSERT INTO user_profile values ('" + name + "','" + cardNumber64 + "','" + threeDigits64 + "','" + validityDate64 +

        rs = p.executeQuery();
```

Known issues – Part I

- Vulnerable to interception attacks due to our protocol implementation between Bank and WebServer.
- Vulnerable to impersonation attacks once the information sent by the bank is not encrypted with bank's private key.
- Vulnerable to DoS attacks due to the slowness of the Virtual Machines.

Known issues – Part II

To make our protocol better:

- In the communication between WebServer to Bank, we encrypt the sensible information with the session key. In an ideal scenario, we would encrypt this information with a secret key and alongside the information we needed to send this secret key encrypted with the bank's public key.

- This secret key would change each time every time we send a packet.