

# 소프트웨어 시스템보안 4차과제

## ~RSA 전자서명~

전북대학교 소프트웨어공학과

유현진  
이수아  
임원용  
임종묵

# 1. 개발환경

- 클라이언트 : JDK 14
- 서버 : node.js v12.20.1
- 의존성도구 : npm v6.14.0
- 사용 라이브러리
  - java.security\* : 전자서명 RSA키 생성 및 검증(Key, Signature)
  - java.net.\* : 클라이언트 → 서버로 값 전달(HttpURLConnection)
  - child\_process : javascript 서버코드에서 Java프로그램 호출

## 2. 구현

### 2-1 RSA키 생성

```
/**
 * 1. publicKey / privateKey 쌍을 생성해주는 객체 2. Strongly random number 생성 3. 키 쌍 생성
 */
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA"); // RSA 암호화 알고리즘으로 키 쌍 생성
SecureRandom random = new SecureRandom(); // 난수생성
kpg.initialize(1024, random); // 초기화
KeyPair keyPair = kpg.genKeyPair(); // 생성한 공개키/비밀키 쌍 가져오기
```

<Client.java>

## 2. 구현

### 2-2 Reference타입, Byte array타입, 문자열타입에 해당하는 각각의 키 생성

```
PublicKey publicKey = keyPair.getPublic(); // 공개 키 가져오기
byte[] bufPublicKey = publicKey.getEncoded(); // PrivateKey객체 -> byte[]형태로 비밀 키 변환
String strPublicKey = Base64.getEncoder().encodeToString(bufPublicKey);
System.out.println(strPublicKey);

PrivateKey privateKey = keyPair.getPrivate(); // 비밀 키 가져오기
byte[] bufPrivateKey = privateKey.getEncoded(); // PublicKey객체 -> byte[]형태로 공개 키 변환
String strPrivateKey = Base64.getEncoder().encodeToString(bufPrivateKey);
```

<Client.java>

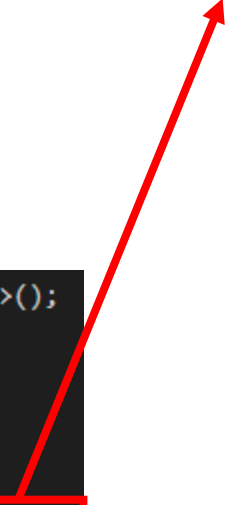
- Key : Signature에 사용되는 API에 사용
- Byte[] : POST요청 시 req.body에 담길 String으로 변환하기 위한 byte array
- String : POST요청 시 req.body에 담길 Parameter. 서버측에서 decode

## 2. 구현

### 2-3 평문, 전자서명, 공개 키를 req.body에 담아 POST request

```
Map<String, String> postRequestParams = new HashMap<>();
postRequestParams.put("plainText", plainText);
postRequestParams.put("signature", strSig);
postRequestParams.put("publicKey", strPublicKey);

// request POST
postRequest("http://127.0.0.1:3000/", postRequestParams);
```



```
// HTTP POST request
private static void postRequest(String targetUrl, Map<String, String> params) throws Exception {
    StringBuilder postData = new StringBuilder();
    for (Entry<String, String> param : params.entrySet()) { ...

    URL url = new URL(targetUrl); // URL 설정
    try {
        // URL 설정 하고 접속하기

        HttpURLConnection http = (HttpURLConnection) url.openConnection(); // 접속
        // System.out.println("http : " + http);
        // -----
        // 전송 모드 설정 - 기본적인 설정
        // -----
        http.setRequestMethod("POST"); // 전송 방식은 POST
        http.setRequestProperty("content-type", "application/x-www-form-urlencoded");
        http.setRequestProperty("Content-Length", String.valueOf(postData.length()));
        http.setRequestProperty("charset", "utf-8");
        http.setUseCaches( false );
        http.setDoOutput(true); // 서버로 쓰기 모드 지정

        // -----
        // 헤더 세팅
        // -----
        // 서버에게 웹에서 <Form>으로 값이 넘어온 것과 같은 방식으로 처리하라는 걸 알려준다

        // -----
        // 서버로 값 전송
        // -----
        OutputStreamWriter outputStream = new OutputStreamWriter(http.getOutputStream(), "UTF-8");
        PrintWriter writer = new PrintWriter(outputStream);
        writer.write(postData.toString());
        writer.flush();

        int code = http.getResponseCode();
        System.out.println("응답코드 : " + code);
        System.out.println("request");
    }
}
```

## 2. 구현

### 2-4 서버의 Router에서 req.body에 담긴 parameter를 parse

```
router.post('/', function (req, res, next) {  
  console.log('post request from ', req.headers['x-forwarded-for'] || req.connection.remoteAddress)  
  
  const strPublicKey = req.body.publicKey  
  const strPlainText = req.body.plainText  
  const strSignature = req.body.signature;  
}
```

<index.js>

## 2. 구현

### 2-5 Decryption을 수행하는 Java Program(sub procedure) 호출

```
var exec = require('child_process').exec, child;
child = exec(`java -jar "C:\\Users\\IOUIOU\\Desktop\\univ\\2021-1\\보안\\과제\\4차과제\\RSA-in-node.js\\routes\\RSADecryption.jar" ${strPublicKey} ${strPlainText} ${strSignature}`,
  function (error, stdout, stderr) {
    console.log('검증 결과: ' + stdout);
    if (error !== null) {
      console.log('exec error: ' + error);
    }
  });
```

<index.js>

## 2. 구현

### 2-7 전자서명 검증

```
public static void main(String[] args) throws InvalidKeyException {
    String strPublicKey = args[0];
    String strPlainText = args[1];
    String strSignature = args[2];

    init(strPublicKey);

    System.out.println(verify(strPlainText, strSignature));
}
```

<RSADecryption.java>

```
post request from ::ffff:127.0.0.1
POST / 200 10.324 ms - 6
검증 결과: true
```

서버 측 POST결과 : 검증성공



## 2. 구현

2-6 publicKey, plaintext, signature를 통해 전자서명 검증

```
public static void init(String strPublicKey)
    throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidKeySpecException {
    byte[] bufPublicKey = Base64.getDecoder().decode(strPublicKey);
    publicKey = KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(bufPublicKey));
    sig = Signature.getInstance("SHA1WithRSA");

    public static boolean verify(String plainText, String strSignature) throws InvalidKeyException, SignatureException {
        byte[] bufPlaninText = plainText.getBytes();
        byte[] bufSignature = Base64.getDecoder().decode(strSignature);

        sig.initVerify(publicKey);
        sig.update(bufPlaninText);

        return sig.verify(bufSignature);
    }
}
```

<RSADecryption.java>