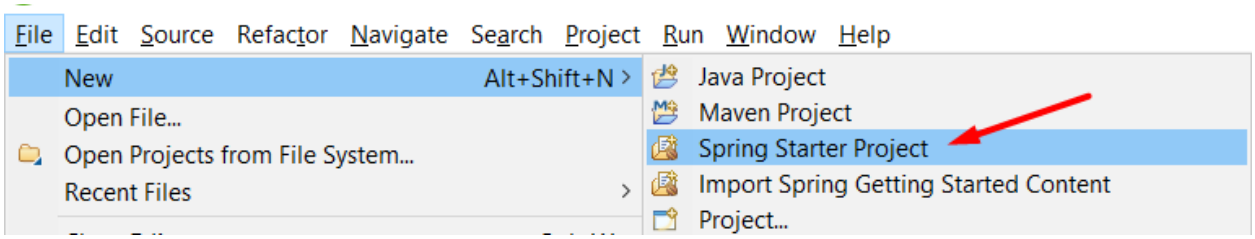
Proyecto REST de Cifrado Simétrico (AES) y Asimétrico (RSA)

Paso 1: Crear el Proyecto en STS

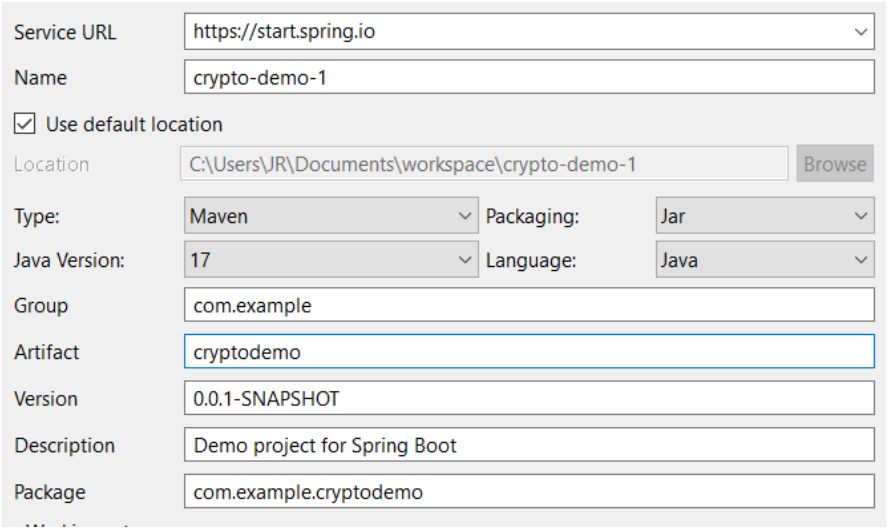
Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**



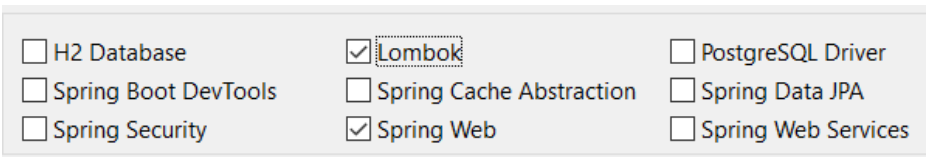
Completar los datos del proyecto:

- **Name:** crypto-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** cryptodemo
- **Package:** com.example.cryptodemo
- **Packaging:** Jar



Seleccionar las dependencias necesarias:

- **Spring Web**
- **Lombok**



Hacer clic en **Finish** para generar el proyecto.

Paso 2: Crear en el paquete Util la clase AESUtil

Creamos la clase Util de AESUtil

```
package com.example.cryptodemo.util;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AESUtil {
    private static final String SECRET_KEY = "1234567890123456"; // 16 chars = 128 bits

    public static String encrypt(String input) throws Exception {
        SecretKeySpec key = new SecretKeySpec(SECRET_KEY.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] encrypted = cipher.doFinal(input.getBytes());
        return Base64.getEncoder().encodeToString(encrypted);
    }

    public static String decrypt(String encryptedText) throws Exception {
        SecretKeySpec key = new SecretKeySpec(SECRET_KEY.getBytes(), "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] decoded = Base64.getDecoder().decode(encryptedText);
        byte[] original = cipher.doFinal(decoded);
        return new String(original);
    }
}
```

Explicación:

```
private static final String SECRET_KEY = "1234567890123456"; // 16 chars = 128 bits
```

Definimos una clave secreta fija de 16 caracteres (AES necesita 128 bits = 16 bytes).

Método encrypt(String input):

```
SecretKeySpec key = new SecretKeySpec(SECRET_KEY.getBytes(), "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] encrypted = cipher.doFinal(input.getBytes());
return Base64.getEncoder().encodeToString(encrypted);
```

- **SecretKeySpec:** prepara la clave como objeto Key para AES.
- **Cipher.getInstance("AES"):** obtiene el algoritmo AES.
- **init(Cipher.ENCRYPT_MODE, key):** inicializa el cifrador para cifrar.
- **doFinal(input.getBytes()):** aplica el algoritmo al texto.
- **Base64.encodeToString(...):** convierte los bytes cifrados en texto legible.

Método decrypt(String encryptedText):

```
SecretKeySpec key = new SecretKeySpec(SECRET_KEY.getBytes(), "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.DECRYPT_MODE, key);
byte[] decoded = Base64.getDecoder().decode(encryptedText);
byte[] original = cipher.doFinal(decoded);
return new String(original);
```

- **SecretKeySpec:** Crea la misma clave que para cifrar.
- **Cipher.getInstance("AES"):** obtiene el algoritmo AES.
- **init(Cipher.DECRYPT_MODE, key):** inicializa el cifrador para descifrar.
- **byte[] decoded = Base64.getDecoder().decode(encryptedText):** Decodifica el Base64 a bytes cifrados.
- **byte[] original = cipher.doFinal(decoded) :** Descifra los bytes.
- **return new String(original):** Convierte los bytes descifrados a String.

Paso 3: Crear en el paquete Util la clase RSAUtil

Creamos la clase Util de RSAUtil

```java
package com.example.cryptodemo.util;

import java.security.*;
import javax.crypto.Cipher;
import java.util.Base64;

public class RSAUtil {

    public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
        generator.initialize(2048); // 2048 bits = buena seguridad
        return generator.generateKeyPair();
    }

    public static String encrypt(String text, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        return Base64.getEncoder().encodeToString(cipher.doFinal(text.getBytes()));
    }

    public static String decrypt(String encrypted, PrivateKey privateKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decoded = Base64.getDecoder().decode(encrypted);
        return new String(cipher.doFinal(decoded));
    }
}
```

Explicación:

Método generateKeyPair():

```java
public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
        generator.initialize(2048); // 2048 bits = buena seguridad
        return generator.generateKeyPair();
    }
```

- **public static KeyPair generateKeyPair():** Método para generar par de claves RSA.
- **KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA"):** Obtiene generador de pares de claves RSA.
- **generator.initialize(2048):** Configura tamaño de clave a 2048 bits.
- **return generator.generateKeyPair():** Genera y devuelve el par de claves.

Método String encrypt():

```
public static String encrypt(String text, PublicKey publicKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        return Base64.getEncoder().encodeToString(cipher.doFinal(text.getBytes()));
}
```

- **Cipher cipher = Cipher.getInstance("RSA"):** Obtiene instancia de cifrador RSA.
- **cipher.init(Cipher.ENCRYPT_MODE, publicKey):** Inicializa en modo ENCRYPT con clave pública.
- **return Base64.getEncoder().encodeToString(cipher.doFinal(text.getBytes())):** Cifra texto y codifica en Base64.

Método String decrypt():

```
public static String decrypt(String encrypted, PrivateKey privateKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decoded = Base64.getDecoder().decode(encrypted);
        return new String(cipher.doFinal(decoded));
    }
```

- **Cipher cipher = Cipher.getInstance("RSA"):** Obtiene instancia de cifrador RSA.
- **cipher.init(Cipher.DECRYPT_MODE, privateKey):** Inicializa en modo DECRYPT con clave privada.
- **byte[] decoded = Base64.getDecoder().decode(encrypted):** Decodifica Base64 a bytes cifrados.
- **return new String(cipher.doFinal(decoded)):** Descifra y convierte a String.

Paso 4: Crear en el paquete Dto la clase CryptoRequest

Creamos la clase Dto de CryptoRequest

```java
package com.example.cryptodemo.dto;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
public class CryptoRequest {
    private String message;
    private String encrypted;
    private String publicKey;
    private String privateKey;
}
```

Explicación:

- **message:** texto plano
- **encrypted:** texto cifrado
- **publicKey:** clave pública RSA
- **privateKey:** clave privada RSA

Paso 5: Crear en el paquete Controller la clase CryptoController

Creamos la clase Controller de CryptoController

```java
package com.example.cryptodemo.controller;

import com.example.cryptodemo.dto.CryptoRequest;
import com.example.cryptodemo.util.AESUtil;
import com.example.cryptodemo.util.RSAUtil;
import org.springframework.web.bind.annotation.*;

import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;
import java.security.spec.PKCS8EncodedKeySpec;
import java.util.Base64;
import java.util.Map;

@RestController
@RequestMapping("/crypto")
public class CryptoController {

    private KeyPair rsaKeys;

    public CryptoController() throws Exception {
        rsaKeys = RSAUtil.generateKeyPair();
    }

    @PostMapping("/aes/encrypt")
    public String encryptAES(@RequestBody CryptoRequest req) throws Exception {
        return AESUtil.encrypt(req.getMessage());
    }

    @PostMapping("/aes/decrypt")
    public String decryptAES(@RequestBody CryptoRequest req) throws Exception {
        return AESUtil.decrypt(req.getEncrypted());
    }

    @GetMapping("/rsa/keys")
    public Map<String, String> getRSAKeys() {
        return Map.of(
                "publicKey", Base64.getEncoder().encodeToString(rsaKeys.getPublic().getEncoded()),
                "privateKey", Base64.getEncoder().encodeToString(rsaKeys.getPrivate().getEncoded())
        );
    }
```

```
    @PostMapping("/rsa/encrypt")
    public String encryptRSA(@RequestBody CryptoRequest req) throws Exception {
        byte[] pubBytes = Base64.getDecoder().decode(req.getPublicKey());
        PublicKey pubKey = KeyFactory.getInstance("RSA").generatePublic(new
X509EncodedKeySpec(pubBytes));
        return RSAUtil.encrypt(req.getMessage(), pubKey);
    }

    @PostMapping("/rsa/decrypt")
    public String decryptRSA(@RequestBody CryptoRequest req) throws Exception {
        byte[] privBytes = Base64.getDecoder().decode(req.getPrivateKey());
        PrivateKey privKey = KeyFactory.getInstance("RSA").generatePrivate(new
PKCS8EncodedKeySpec(privBytes));
        return RSAUtil.decrypt(req.getEncrypted(), privKey);
    }
}
```

Explicación:

```
private KeyPair rsaKeys;
```

- Almacena el par de claves RSA.

```
public CryptoController() throws Exception {
        rsaKeys = RSAUtil.generateKeyPair();
    }
```

- Constructor: genera par de claves al iniciar.

```
@PostMapping("/aes/encrypt")
    public String encryptAES(@RequestBody CryptoRequest req) throws Exception {
        return AESUtil.encrypt(req.getMessage());
    }
```

- Endpoint POST para cifrado AES.

```
@PostMapping("/aes/decrypt")
    public String decryptAES(@RequestBody CryptoRequest req) throws Exception {
        return AESUtil.decrypt(req.getEncrypted());
    }
```

- Endpoint POST para descifrado AES.

```
@GetMapping("/rsa/keys")
    public Map<String, String> getRSAKeys() {
        return Map.of(
                "publicKey", Base64.getEncoder().encodeToString(rsaKeys.getPublic().getEncoded()),
                "privateKey", Base64.getEncoder().encodeToString(rsaKeys.getPrivate().getEncoded())
        );
    }
```

- Endpoint GET que devuelve las claves RSA en Base64.

```
@PostMapping("/rsa/encrypt")
    public String encryptRSA(@RequestBody CryptoRequest req) throws Exception {
        byte[] pubBytes = Base64.getDecoder().decode(req.getPublicKey());
        PublicKey pubKey = KeyFactory.getInstance("RSA").generatePublic(new
X509EncodedKeySpec(pubBytes));
        return RSAUtil.encrypt(req.getMessage(), pubKey);
    }
```

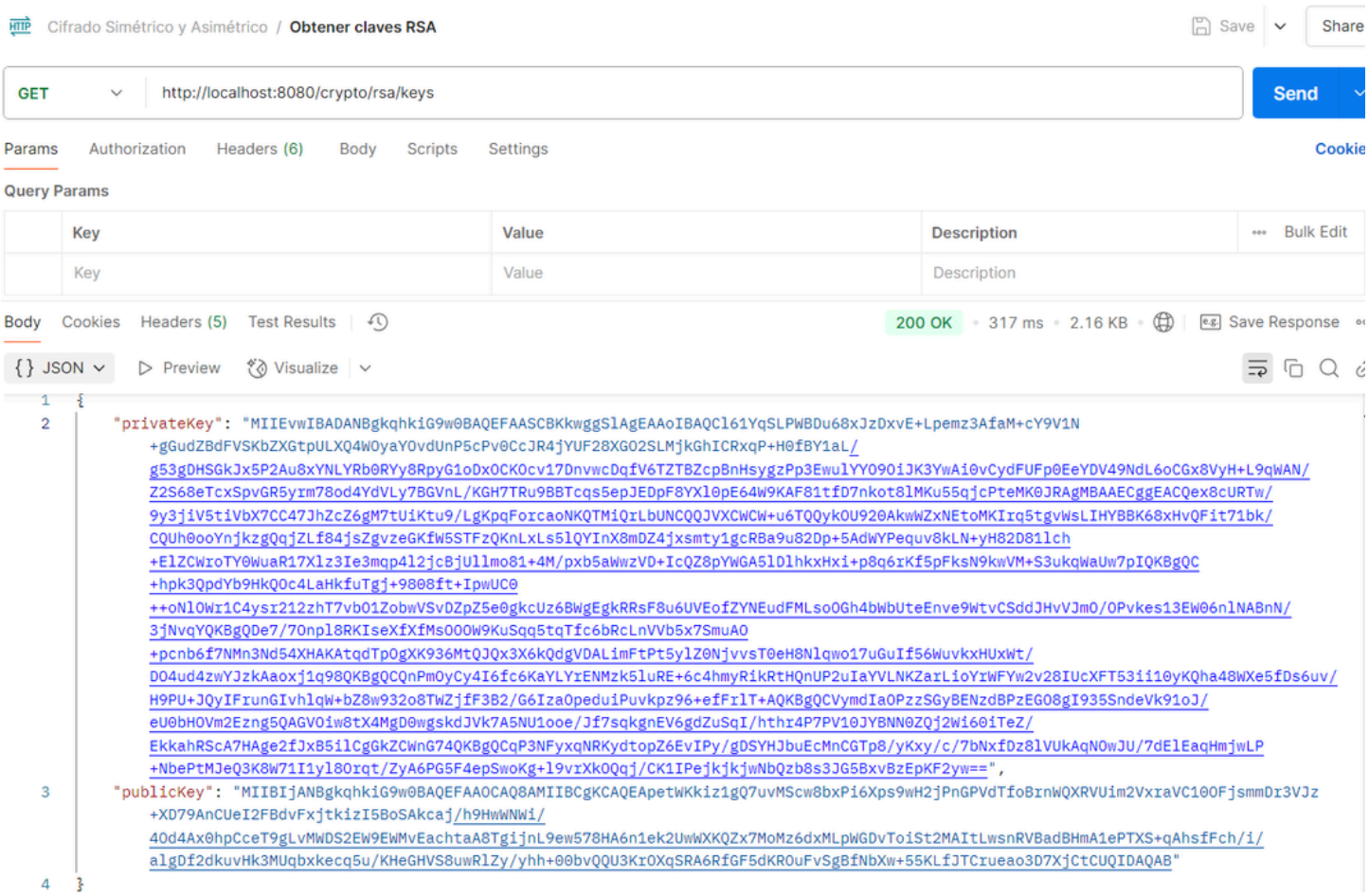
- Endpoint POST para cifrado RSA:
  1. Decodifica clave pública de Base64
  2. Reconstruye objeto PublicKey
  3. Cifra mensaje

```
@PostMapping("/rsa/decrypt")
    public String decryptRSA(@RequestBody CryptoRequest req) throws Exception {
        byte[] privBytes = Base64.getDecoder().decode(req.getPrivateKey());
        PrivateKey privKey = KeyFactory.getInstance("RSA").generatePrivate(new
PKCS8EncodedKeySpec(privBytes));
        return RSAUtil.decrypt(req.getEncrypted(), privKey);
    }
```

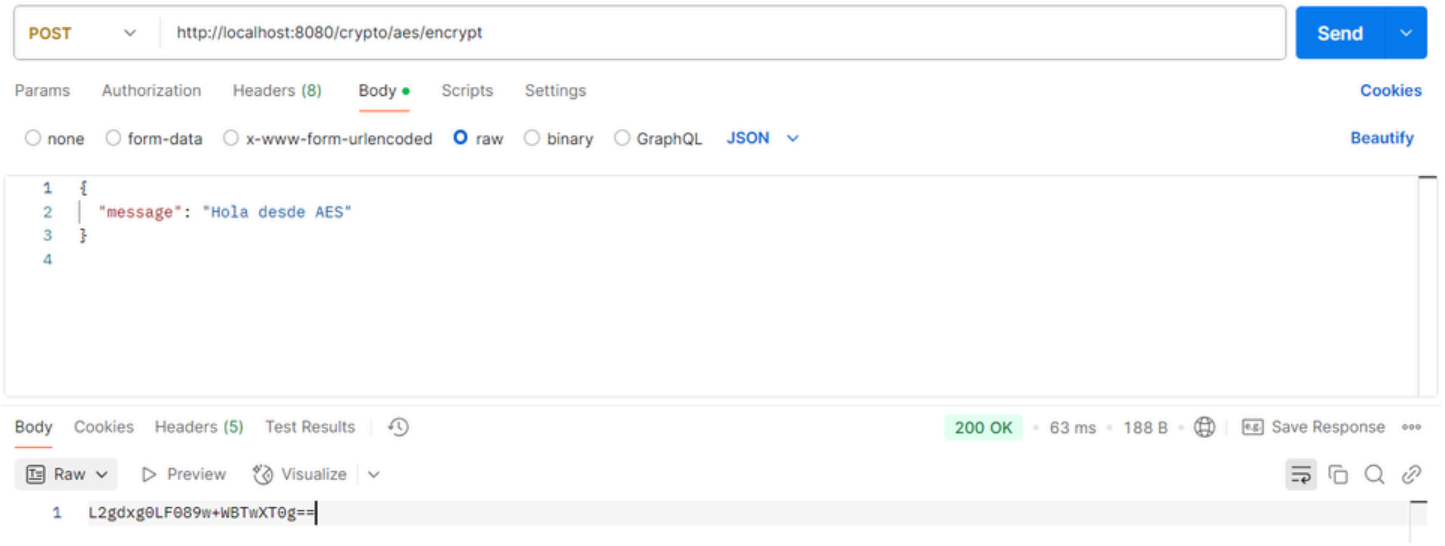
- Endpoint POST para descifrado RSA:
  1. Decodifica clave privada de Base64
  2. Reconstruye objeto PrivateKey
  3. Descifra mensaje

Paso 6: Ejecutar los endpoints en Postman

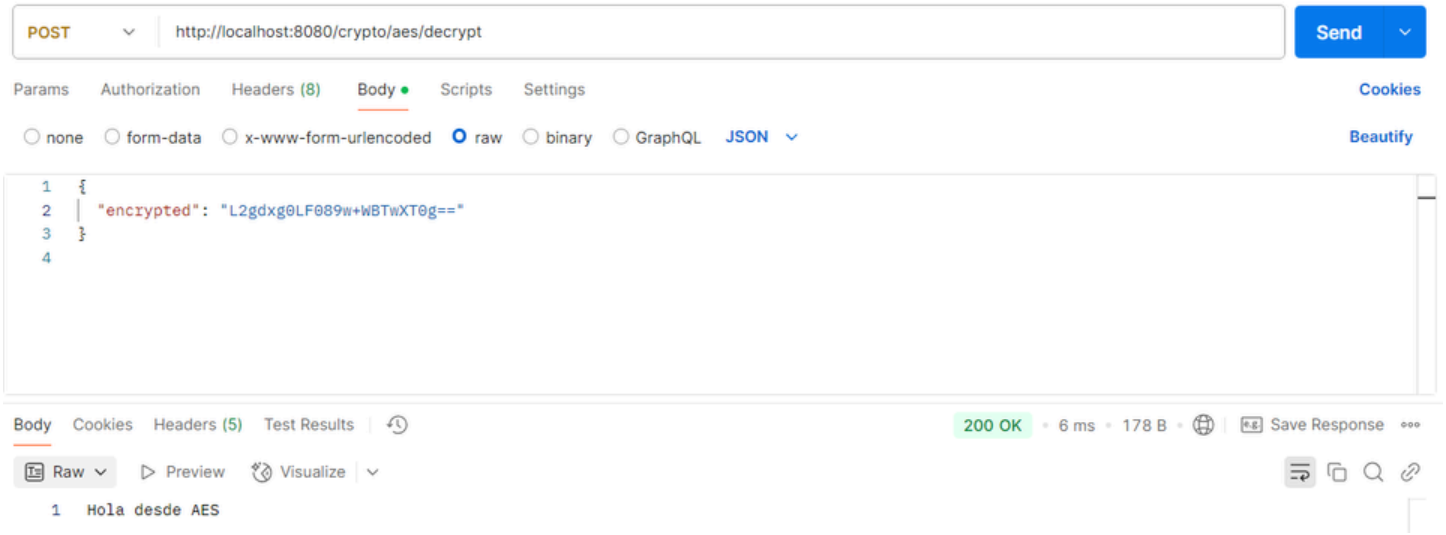
GET http://localhost:8080/crypto/rsa/keys



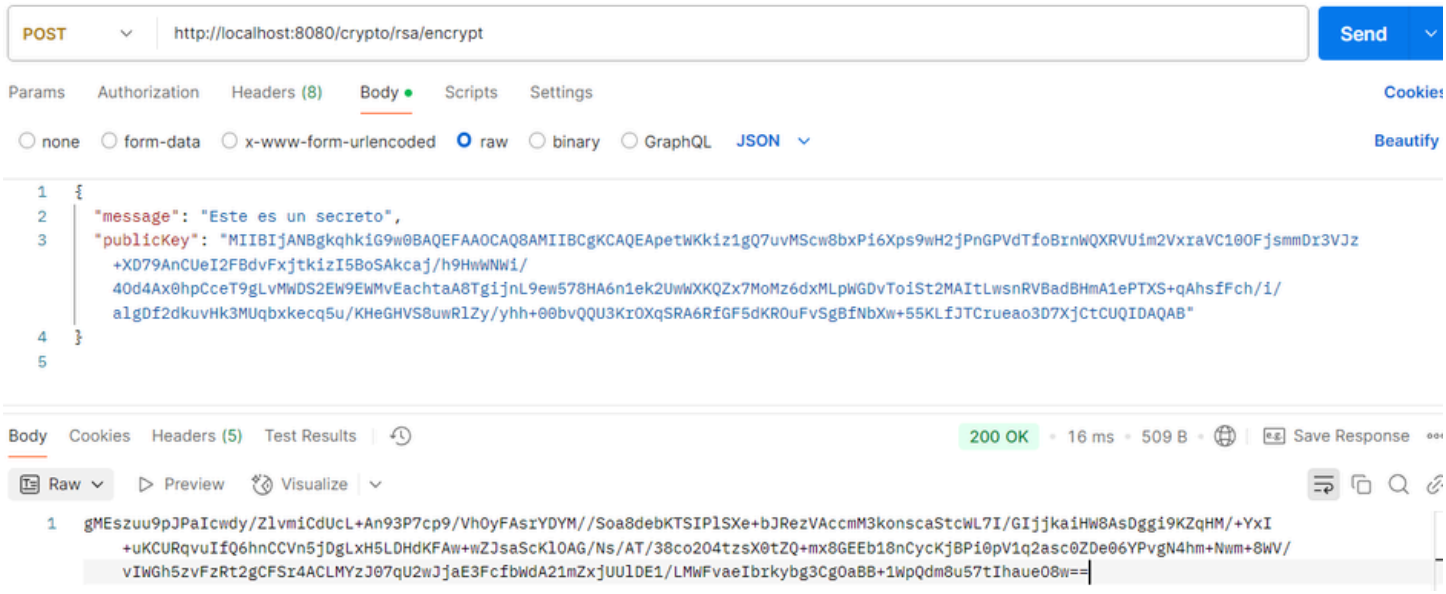
POST http://localhost:8080/crypto/aes/encrypt



POST http://localhost:8080/crypto/aes/decrypt



POST http://localhost:8080/crypto/rsa/encrypt



POST http://localhost:8080/crypto/rsa/decrypt

Cifrado Simétrico y Asimétrico / **Descifrar mensaje con RSA**

Save | Share

POST http://localhost:8080/crypto/rsa/decrypt **Send**

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify


```
{
"encrypted": "gMEszuu9pJPaIcwdy/ZlvmiCdUcL+An93P7cp9/VhOyFAsrYDYM//Soa8debKTSIPlSXe+bJRezVAccmM3konscaStcWL7I/GIjjkaiHW8AsDggi9KZqHM/+YxI
+uKCURqvuIfQ6hnCCVn5jDgLxH5LDHdKFAw+wZJsaScKlOAG/Ns/AT/38co2O4tzsX0tZQ+mx8GEEb18nCycKjBPi0pV1q2asc0ZDe06YPvgN4hm+Nwm+8WV/
vIWGh5zvFzRt2gCFSr4ACLMYzJ07qU2wJjaE3FcfbWdA21mZxjUUlDE1/LMWFvaeIbrkybg3CgOaBB+1WpQdm8u57tIhaueO8w==",
"privateKey": "MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQCl61YqSLPWBDu68xJzDxvE+Lpemz3AfaM+cY9V1N
+gGudZBdFVSKbZXGtpULXQ4WOyaYOvdUnP5cPv0CcJR4jYUF28XGO2SLMjkGhICRxqP+H0fBY1aL/
g53gDHSGkJx5P2Au8xYNLYRb0RYy8RpyG1oDxOCKOcv17DnvwcDqfV6TZTBZcpBnHsygzPp3EwulYYO9OiJK3YwAi0vCydFUFp0EeYDV49NdL6oCGx8VyH+L9qWAN/
Z2S68eTcxSpvGR5yrm78od4YdVLy7BGVnL/KGH7TRu9BBTcqs5epJEDpF8YXl0pE64W9KAF81tfD7nkot8lMKu55qjcPteMK0JRAgMBAAECggEACQex8cURTw/
9y3jiV5tiVbX7CC47JhZcZ6gM7tUiKtu9/LgKpqForcaoNKQTMiQrLbUNCQQJVXCWCW+u6TQQykOU920AkwWZxNEtoMKIrq5tgvWsLIHYBBK68xHvQFit71bk/
CQUh0ooYnjkzgQqjZLf84jsZgvzeGKfW5STFzQKnLxLs5lQYInX8mDZ4jxsmty1gcRBa9u82Dp+5AdWYPequv8kLN+yH82D81lch
+ElZCWroTY0WuaR17Xlz3Ie3mqp4l2jcBjUllmo81+4M/pxb5aWwzVD+IcQZ8pYWGA5lDlhkxHxi+p8q6rKf5pFksN9kwVM+S3ukqWaUw7pIQKBgQC
+hpk3QpdYb9HkQOc4LaHkfuTgj+9808ft+IpwUC0
++oNlOWr1C4ysr212zhT7vbO1ZobwVSvDZpZ5e0gkcUz6BWgEgkRRsF8u6UVEofZYNEudFMLsoOGh4bWbUteEnve9WtvCSddJHvVJmO/OPvkes13EW06nlNABnN/
3jNvqYQKBgQDe7/7Onpl8RKIseXfXfMsOOOW9KuSqq5tqTfc6bRcLnVVb5x7SmuAO
+pcnb6f7NMn3Nd54XHAKAtqdTpOgXK936MtQJQx3X6kQdgVDALimFtPt5ylZ0NjvvsT0eH8Nlqwo17uGuIf56WuvkxHUxWt/
DO4ud4zwYJzkAaoxj1q98QKBgQCQnPmOyCy4I6fc6KaYLYrENMzk5luRE+6c4hmyRikRtHQnUP2uIaYVLNKZarLioYrWFYw2v28IUcXFT53ii10yKQha48WXe5fDs6uv/H9PU
+JQyIFrunGIvhlqW+bZ8w932o8TWZjfF3B2/G6IzaOpeduiPuvkpz96+efFrlT+AQKBgQCVymdIaOPzzSGyBENzdBPzEGO8gI935SndeVk91oJ/
```


Body Cookies Headers (5) Test Results 200 OK 23 ms 182 B Save Response

Raw Preview Visualize

```
Este es un secreto
```