Mach 2024

**API FOR CORPORATE SERVICES**

SIGNATURE DOCUMENT

Content

[1. Introduction 3](#_Toc159242850)

[2. Java example 3](#_Toc159242851)

[2.1. Hash of the body 3](#_Toc159242852)

[2.2. Signature generation 4](#_Toc159242853)

[3. Node.js example 5](#_Toc159242854)

[3.1. Hash du body 5](#_Toc159242855)

[3.2. Signature generation 5](#_Toc159242856)

[4. Example 6](#_Toc159242857)

# Introduction

The purpose of this document is to explain you how to hash the body and how to generate the payment signature. This is applicable to all our payment APIs.

# Java example

## Hash of the body

public String hashRequest(String stringToHash) {  
  
  
 byte[] encodeHash;  
 1) encodeHash = stringToHash.getBytes(StandardCharsets.*UTF\_8*);  
  
 byte[] returnHash;  
  
 try {  
 2) MessageDigest digest = MessageDigest.*getInstance*("SHA-256");  
 3) returnHash = digest.digest(encodeHash);  
 } catch (NoSuchAlgorithmException e) {  
 throw new Exception(e);  
 }  
  
 4) return Base64.*getEncoder*().encodeToString(returnHash);  
}

The library used for the signature hash is *java.security.*

This function takes as a parameter the body to hash in String and returns the hash of the body in base 64.

1. The body is transformed into an array of bytes with the *String.getBytes()* method. The encoding is done in *UTF\_8*.
2. We instantiate the *java.security.MessageDigest* class by giving it the algorithm we use *SHA-256* as a parameter. The hash is done with this algorithm, if another algorithm is used the transaction will be rejected.
3. We use the *digest()* method of the *java.security.MessageDigest* class by passing the byte array from step 1) as a parameter.
4. We return the base 64 hash String with the *Base64.getEncoder().encodeToString()* method of the *java.util.base64* class, the parameter is the byte array generated in step 3).

It is now necessary to enrich the Digest for corresponding it to the spec.

Example the method previously returned the following value: Wj0Fqc7oN+VNcSrZhAapWaWSyUQW5WTr7U6wiQ7erfw=

We must add "Digest: SHA-256 =" so that the String that we will pass in step 2:

"Digest: SHA-256=Wj0Fqc7oN+VNcSrZhAapWaWSyUQW5WTr7U6wiQ7erfw="

## Signature generation

*/\*\*  
 \* Generate signature for a plain-text message using a private key  
 \*  
 \** ***@param*** *message message that will be signed  
 \** ***@param*** *privateKey private key that will sign the message  
 \** ***@return*** *the signature byte array  
 \*/*static String sign(String message, PrivateKey privateKey) {  
 Signature rsa = null;  
  
  
 try {  
 1) rsa = Signature.*getInstance*("SHA256withRSA");  
 2) rsa.initSign(privateKey);  
 } catch (NoSuchAlgorithmException e) {  
 throw new IllegalStateException(e);  
 } catch (InvalidKeyException e) {  
 throw new IllegalArgumentException(e);  
 }  
  
 3) try (ByteArrayInputStream inputStream = new ByteArrayInputStream(message.getBytes(StandardCharsets.*UTF\_8*))) {  
 4) byte[] buffer = new byte[1024];  
 int len;  
  
 5) while ((len = inputStream.read(buffer)) >= 0) {  
 6) rsa.update(buffer, 0, len);  
 }  
 } catch (IOException e) {  
   
 throw new Exception(e);  
 } catch (SignatureException e) {  
   
 throw new Exception(e);  
 }  
 byte[] signature;  
 try {  
 7) signature = rsa.sign();  
 } catch (SignatureException e) {  
 throw new Exception(e);  
 }  
 8) return Base64.*getEncoder*().encodeToString(signature);  
}

In this example, the certificate, public and private key generation is done previously, which is why the *sign()* method takes as a parameter the private key of the *java.security.PrivateKey* class. This method takes the body hash as its first parameter.

1. We initialize the signature of the *java.security.Signature* class by adding the algorithm we will use to sign: SHA256.
2. We add to the signature the private key that we will use with the *initSign()* method of the *java.Security.signature* class.
3. Then add the body hash to the signature. For that we create a *ByteArrayInputStream* of the *java.io.ByteArrayInputStream* class. We then convert the body hash into an array of byte with the *getBytes()* method of the String class. The encoding must be done in *UTF\_8*.
4. We initialize the size of the buffer to 1024.
5. We browse the buffer until the end.
6. We add the digest hash to the signature with the *update()* method of the *java.security.Signature class*.
7. We then sign with the *sign()* method of the *java.security.Signature class*. The result of this method is stored in an array of bytes.
8. We return the byte array converted to String in base64.

# Node.js example

## Hash of the body

1) const postBuffer = Buffer.from(postData, 'utf8');

2) const digest = 'SHA-256=' + crypto.createHash('sha256').update(Buffer.from(postBuffer)).digest("base64");

1. We add the JSON to the buffer with the node.js Buffer.from method. The encoding is done in *UTF\_8.*
2. We then create the digest by concatenating “SHA-256 =” and the hashed digest. To hash the digest we use *crypto.createHash()* by choosing the algorithm used (here sha256). We add the buffer previously initialized with the function *update(Buffer.from(postBuffer).* We then indicate that the encoding will be done in base64 with the *digest()* function.

## Signature generation

1) const toSign = 'Digest: ' + digest;  
  
2) const sign = crypto.createSign('RSA-SHA256');  
3) sign.update(toSign);  
4) sign.end();  
5) const signature = sign.sign(  
 { key: readFileSync(SIGNATURE\_PRIVATE\_KEY\_PATH, 'utf-8'), format: 'pem' },  
 'base64'  
);

1. We concatenate 'Digest: ' with the digest hash previously..
2. We initialize the signature with the *crypto.createSign()* function by specifying the algorithm we will use (RSA-SHA256 here).
3. We add the signature to the signature constant initialized in step 2) with the *sign.update()* function.
4. We close the signature buffer with the *end()* function.
5. We then sign with the *sign()* function specifying the private key used. The private key in this case is read from a file synchronously (readFileSync) specifying the encoding (utf-8) and the format of the key (pem). We then convert to base64.

# Example

Full Body: use the file **bodyV2.txt**

Certificate : use the file **doc\_signature2.cer**

Private key : use the file **doc\_signature2\_forJava**

Private key : use the file **doc\_signature2\_forNode**

We hash the body here is the result of the hash: oDGEYHEZ8owOBNn52DLZV0QACl6TVLVGTjCloAdfk7U=

We then enrich the Hash: Digest: SHA-256=oDGEYHEZ8owOBNn52DLZV0QACl6TVLVGTjCloAdfk7U=

We then sign the enriched Hash: HA/2HA5e7l8MbGBTNf7C+QKtV32w0nEKDLfAIOXE+ifHWF0MfgkwB6l3Mf7dPZ1Y8wFXTLx1IgiZUBv2mIDGeyeNHnllS6X8SuLdzZMUdSqlc4c/lhJrEBVCt9w7dfDd9v6PxbPViJDUYZ8cPLtpy3xiYMsMnbvMY6Lvl21jPS6r5mL1S4y8zzUdzQcsT3UcKnUyg9E7dxPNrtzwZbxnOKKYYVxj2rBwxRW0QFyeRSep1Bg0niFQaQg24EF+3UZw3/4h0fuUYuqQCfNR0jOJZ4OLFCazAHxB8GA30R1MKqmx2dqsLRKbA8CVVSYoopaocckpPUEUy2IlEcCnwajc0A==