

Danske Click

Information for Merchants

Contents

- 1. License agreement 3
- 2. Document purpose 3
- 3. What is Danske Click? 3
- 4. Danske Click information protection..... 3
- 5. How does Danske Click work? 3
- 6. Danske Click messages 4
 - Message No. 1001 5
 - Message No. 1101 6
 - Message No. 1201 7
 - Message No. 1901 8
- 7. Launching of IT system interface..... 9
- 8. Example of the PKCS#10 certificate query 9
- 9. Example of the HTML form of the payment transfer message (No. 1001) 10
- 10. Example of the Danske Click message signing and signature verification 10
 - PERL 11
 - PHP 13
 - Java 14

1. License agreement

The document is the property of **Danske Bankas** (hereinafter – the Bank). All rights reserved. No copying without a written permission from the Bank.

The document has been prepared in accordance with UAB Forbis document “IB Pay. Payments through FORPOST*Internet Banking. Information for sellers”.

2. Document purpose

This document is intended for the persons administering e-trade portals.

The persons having acknowledged with the document will understand:

- **Danske Click** operation principles;
- configuration rules of e-trade portal.

3. What is Danske Click?

Danske Click is a settlement system intended for the Bank’s business clients – managers of electronic trade portals (further referred to as the Traders).

This system will help the Bank’s and the trader’s common customers (further referred to as the Purchasers) to make direct online payments for goods and services offered on the e-trade portals.

At least three participants must participate in the system to make it operate successfully:

- the Bank;
- the trader who made an agreement on **Danske Click** Usage with the Bank;
- the purchase using the Internet Banking System **Danske eBank**.

4. Danske Click information protection

The principle of operation of the **Danske Click** system – exchange of the trader’s and the Bank’s systems in special HTML form parameter sets signed by digital signatures of the parties using RSA PKCS#1 standard.

It is obligatory to use the HTTP SSL encoding for the protection of the contact between the Bank and the trader.

5. How does Danske Click work?

The purchaser fills the shopping basket in the trader’s e-trade portal and selects the option of payment through **Danske Bank** online banking system (further referred to as the **Danske eBank**).

After the purchaser selects the payment through **Danske eBank**, the trader’s system processes a service payment order message (1001), signs it by own digital signature and then

directs the purchaser together with the signed message to the special page for the logging on to the **Danske eBank**, where overall payment information is displayed.

When the purchaser successfully logs on to the **Danske eBank** system, a filled in payment order appears right away. Here the purchaser may set only the debited account number.

After the purchaser signs the payment order and money is transferred to the trader's bank account, the **Danske Click** system processes, signs and sends a payment confirmation message (1101) into the trader's system.

If the purchaser decides to terminate the payment process, the Bank's system rejects the payment transfer or the payment transfer expires (expiration date selected by the trader), the **Danske Click** system processes, signs and sends a payment cancellation message (1901) into the trader's system.

If the purchaser decides to suspend the payment execution, the **Danske Click** system processes, signs and sends a payment suspension message (1201) into the trader's system. In such a case the **Danske Click** system will definitely notify about the further payment execution course as soon as this process is renewed (message 1101 or 1901).

6. Danske Click messages

Danske Click messages are HTTPS queries with HTML parameters which are transmitted by the HTTPS POST method. In the HTML form parameters the message fields are transmitted, one of which is the digital signature of the other fields. One line consisting of the message field values connected in a certain order is signed using the digital signature, where the three-digit length message field value (if the length is shorter than 100, leading zeroes are added) is followed by the field value itself; it is followed by the message field value length and value which is next in the sequence; it is repeated until all message field values are connected. An example of three fields: 007ABCDEFGG001A00288. The spaces existing at the beginning and the end of the field values are deleted and the empty values are not signed at all.

The Unicode encoding cannot be used in the HTML page containing the transmitted field form and in making the signature; so for the national symbols transmitted in the fields to be displayed in the **Danske eBank** pages correctly, presently it is recommended to use Windows-1257 for the Lithuanian language, Windows-1251 - for Russian, and any one-byte encoding - for English.

The RSASSA-PKCS1-v1_5 scheme with the SHA1 compression algorithm is used for the signature (for more information about the PKCS#1 standard see www.rsasecurity.com). The signature entered into the message field is encoded with the Base64 algorithm.

The merchant is given the possibility of testing the message sending. To view the information about the errors in an incorrectly formed message it is necessary to transmit the additional HTML form parameter "DEBUG", the value of which must not be empty.

Descriptions of the message field sets are provided below (all the signed fields have a sequence number).

Message No. 1001

The merchant message containing the service payment transfer data.

Sequence No.	Name of HTML form parameter	Maximum length of the field value	Obligatory	Description
1	VK_SERVICE	4	Yes	Message number 1001.
2	VK_VERSION	3	Yes	Signature algorithm 008.
3	VK_SND_ID	100	Yes	Merchant ID.
4	VK_STAMP	100	No	Query identifier. Usually corresponds to the customer basket identifier.
5	VK_AMOUNT	16	Yes	Payment amount with cents delimited by the dot symbol.
6	VK_CURR	3	Yes	Currency abbreviation (ISO 4217).
7	VK_TERM	19	No	Payment expiry date. Format: DD.MM.YYYY HH24:MI:SS
8	VK_ACC	35	No	Beneficiary Danske Click account number.
9	VK_PCODE	50	No	Payment code.
10	VK_PANK	35	No	Beneficiary bank code.
11	VK_NAME	200	No	Beneficiary name.
12	VK_REF	10	No	Payment document number. If not indicated - selected by the payer (customer) him or herself.
13	VK_MSG	300	Yes	Payment purpose.
	VK_MAC	400	Yes	Digital RSASSA-PKCS1-v1_5 signature with the SHA1 compression algorithm.
	VK_RETURN	256	No	The Internet address for sending the settlement process messages (URL of the merchant).
	VK_LANG	3	No	Customer language (ISO-639 three-letter or two-letter code).

Message No. 1101

The bank message of the successful fulfilment of the service payment transaction.

Sequence No.	Name of HTML form parameter	Maximum length of the field value	Description
1	VK_SERVICE	4	Message number is 1101.
2	VK_VERSION	3	Signature algorithm number is 008.
3	VK_SND_ID	100	Bank identifier.
4	VK_REC_ID	100	Merchant identifier.
5	VK_STAMP	100	Query identifier.
6	VK_AMOUNT	16	Payment amount with cents delimited by the dot symbol.
7	VK_CURR	3	Currency abbreviation (ISO 4217).
8	VK_REC_ACC	35	Beneficiary account number.
9	VK_REC_NAME	200	Beneficiary name.
10	VK_SND_ACC	35	Payer account number.
11	VK_SND_NAME	200	Payer name.
12	VK_REF	10	Payment document number.
13	VK_MSG	300	Payment purpose.
14	VK_T_DATE	10	Payment date. Format: DD.MM.YYYY
	VK_T_NO	12	Message identifier.
	VK_PANK	35	Beneficiary bank code.
	VK_MAC	400	Digital RSASSA-PKCS1-v1_5 signature with the SHA1 compression algorithm.
	VK_LANG	3	Customer language (ISO-639: two-letter or three-letter code).
	VK_AUTO	1	The value is „Y“ if Danske Click system sends the message automatically. The value is „N“ if the message is sent by directing the user to the merchant's page.

Message No. 1201

The bank message of the successful service payment transaction, which was received but is not confirmed yet.

Sequence No.	Name of HTML form parameter	Maximum length of the field value	Description
1	VK_SERVICE	4	Message number is 1201.
2	VK_VERSION	3	Signature algorithm number is 008.
3	VK_SND_ID	100	Bank identifier.
4	VK_REC_ID	100	Merchant identifier.
5	VK_STAMP	100	Query identifier.
6	VK_AMOUNT	16	Payment amount with cents delimited by the dot symbol.
7	VK_CURR	3	Currency abbreviation (ISO 4217).
8	VK_REC_ACC	35	Beneficiary account number.
9	VK_REC_NAME	200	Beneficiary name.
10	VK_SND_ACC	35	Payer account number.
11	VK_SND_NAME	200	Payer name.
12	VK_REF	10	Payment document number.
13	VK_MSG	300	Payment purpose.
	VK_T_NO	12	Message identifier.
	VK_PANK	35	Beneficiary bank code.
	VK_MAC	400	Digital RSASSA-PKCS1-v1_5 signature with the SHA1 compression algorithm.
	VK_LANG	3	Customer language (ISO-639 two-letter or three-letter code)
	VK_AUTO	1	The value is „N“

Message No. 1901

The bank message of the terminated service payment.

Sequence No.	Name of HTML form parameter	Maximum length of the field value	Description
1	VK_SERVICE	4	Message number is 1901.
2	VK_VERSION	3	Signature algorithm number is 008.
3	VK_SND_ID	100	Bank identifier.
4	VK_REC_ID	100	Merchant identifier.
5	VK_STAMP	100	Query identifier.
6	VK_REF	10	Payment document number.
7	VK_MSG	300	Payment purpose.
	VK_MAC	400	Digital RSASSA-PKCS1-v1_5 signature with the SHA1 compression algorithm.
	VK_LANG	3	Customer language (ISO-639 two-letter or three-letter code).
	VK_AUTO	1	The value is „Y“ if the Danske Click system sends the message automatically. The value is „N“ if the message is sent by directing the user to the merchant's page.

7. Launching of IT system interface

Both parties seeking to relate the trader's and the Bank's IT systems must exchange certain data and parameters.

The trader must submit to the Bank:

- request file of his RSA public key PKCS#10 certificate in PEM format (it is not recommended to use the key length shorter than 1024 bits, maximum key length – 2048 bits);
- commercial service name;
- Internet address to receive the Bank's messages about the course of settlements. The given address must be safe (HTTP SSL);
- his decision on fixing of the settlement term (it is possible to set a fixed term in days for the transaction, and if each message on service payment transfer is submitted – to specify a special transfer expiration term – date and time).

The Bank must submit to the trader:

- its public key certificate X.509 file;
- the Bank's public key that is announced on the Bank's website <http://www.danskebankas.lt/files/Danske.Click.crt>;
- trader's ID;
- logging on to the **Danske eBank** Internet address whereby the service payment transfer message will be transmitted and the paying user directed:
<https://ebankas.danskebankas.lt/ib/site/ibpay/login>.

8. Example of the PKCS#10 certificate query

To make, check and change the format of the RSA keys it is possible to use the free program OpenSSL (<http://www.openssl.org>). It is distributed for Linux and Windows operating systems (<http://www.slproweb.com/products/Win32OpenSSL.html>).

Below is an example of the contents of the settings file openssl.cnf:

```
[ req ]
default_bits = 1024
default_keyfile = pk8.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca
dirstring_type = nobmp
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = LT
countryName_min = 2
countryName_max = 2
localityName = Locality Name (eg, city)
organizationalUnitName = Organizational Unit Name (eg, section)
commonName = Common Name (eg, YOUR name)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 40
[ req_attributes ]
challengePassword = A challenge password
```

```
challengePassword_min = 4
challengePassword_max = 20
[ v3_ca ]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = CA:true
```

The key_rsa.pem file with the RSA key pair information is generated.

```
openssl genrsa -nodes -out key_rsa.pem 1024
```

The key_rsa.pem file is saved in the special OpenSSL format; it is recommended to change it into the PKCS#8 format:

```
openssl pkcs8 -topk8 -inform PEM -in key_rsa.pem -nocrypt -out pk8.pem
```

In order to get a certificate from a partner the certificate query is obligatory. This method is applied when a partner requires our public key (for example, to verify our signatures); we transmit to the partner the certificate query with our public key and they must return our public key certificate, i.e. our public key signed by the partner. Our private key file in the PKCS#8 format is required to make this query. In the command example below the (PKCS#8) pk8.pem file is indicated and the (PKCS#10) req.pem file is received:

```
openssl req -new -key pk8.pem -out req.pem -config openssl.cnf
```

9. Example of the HTML form of the payment transfer message (No. 1001)

```
<form action="https://ebankas.danskebankas.lt/ib/site/ibpay/login" method="POST">
  <input type="hidden" name="VK_MSG" value="Užsakymas patalpintas: 05/27/04 16:52:20">
  <input type="hidden" name="VK_LANG" value="LIT">
  <input type="hidden" name="VK_NAME" value="UAB &quot;CITY Shop&quot; ">
  <input type="hidden" name="VK_AMOUNT" value="64.99">
  <input type="hidden" name="VK_SERVICE" value="1001">
  <input type="hidden" name="VK_PANK" value="40100">
  <input type="hidden" name="VK_REF" value="L14760740">
  <input type="hidden" name="VK_PCODE" value="1234">
  <input type="hidden" name="VK_VERSION" value="008">
  <input type="hidden" name="VK_MAC"
  value="ElceYj7426sj6QGknVeObnQm75JzsazOvSicfC8uWfFRMPrnMyusc7HqRAohP/Dua42JWgZ5eqzkdHm
  WLC8Q6/A3I6w+CpsM4jHs7pQzOqNDyA00dox1QwJWBguyKz+slUtIDB6PTZEWp85m4wOKpFtNXNMk7p
  1Ye61YpoNUI=">
  <input type="hidden" name="VK_CURR" value="LTL">
  <input type="hidden" name="VK_STAMP" value="L14760740">
  <input type="hidden" name="VK_RETURN" value="https://www.merchant.com/cgi-
  bin/store/store.cgi?&amp;cart=62456482x12711&amp;session=40b5ef5a35550337&amp;L=lit&amp;Subm
  itOrderButton=1&amp;PaymentMode=DanskeClick&amp;TransactionID=53098402x13714&amp;Total=64.9
  9&amp;OrderID=L14760740">
  <input type="hidden" name="VK_SND_ID" value="IBPAY74233">
  <input type="hidden" name="VK_ACC" value="LT554010049500039227">
  <input type="submit" name="Submit" value="Pay with Danske Click">
  <!--input type="submit" name="DEBUG" value="1"-->
</form>
```

10. Example of the Danske Click message signing and signature verification

The public key derivable from the X.509 certificate which is entered into the file in the PEM format is used in the example below:

```
-----BEGIN CERTIFICATE-----
MIICxjCCAi+gAwIBAgIBADANBgkqhkiG9w0BAQQFADCBqDEPMAOGA1UEAxMGRk9S
QkITMQswCQYDVQQGEwJMMDVDEQMA4GA1UEBxMHVkiMTkivUzEQMA4GA1UECBMHVkiM
TkIVUzEoMCYGA1UEChMfVVpEQVJPSkkgQUtDSU5FIEJFTkRST1ZFIEZPUkJKUzEU
MBIGA1UECxMlSU5URUdSQVRJTO4xJDAiBgkqhkiG9w0BCQEFUwuwSIVaSUtFTkFT
QEZPUkJKUy5MVDAeFwOwNDAzMjxNTE2MDNaFwOwNDAOMjExNTE2MDNaMIGoMQ8w
DQYDVQQDEwZGT1JCSVMxMzA1JG9wBAYTAkxUMRAwDgYDVQQHEwdWSUxOSVVTMRAw
DgYDVQQIEwdWSUxOSVVTMSgwJgYDVQQKEw9VWwRBUk9KSSBBSONJTkUgQkVORFJP
VkUgRk9SQkITMRQwEgYDVQQLEwtJTRFR1JBVEIPTjEkMCIGCSqGSIb3DQEJARYV
TC5KVVpJSOVoQVNARk9SQkITLkxUMIGfMAOGCSqGSIb3DQEBAAQAA4GNADCBiQKB
gQCrbyCkLdo1gfT3d5JjwrLYC8WAXNI50aftGx9+ncjfnONGtScsbwlQ5Qw55neH
TUe1Tb1/QJc8KZ7PU5/sJAVNpuJW9JEIOy1xX6egfVSWkDGv/GgSb2JebnD1+Nw2
fw8IU0v4F6/ljHU9FofSTBAIN58g5FqTweAZg5BU7uN9XwIDAQABMAOGCSqGSIb3
DQEBBAUAA4GBACH8eFBJ/8p8f1t8TWlh6IX4hpGpyej0h+OBW45icxkpDplfGbFx
470CIHjzgROo6zFz7Axn5JC1IWPtiyyIbSbahpyCpcasuQchErcXJ72ctq8nBXqV
s7sPhlunemdfpFuZLBNFbw5xaUs+lt9tAZFi6EHnhjCFGih4u5aRcta
-----END CERTIFICATE-----
```

The private key derivable from PKCS#8 which is entered into the file in the PEM format is used in the example below:

```
-----BEGIN PRIVATE KEY-----
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAKtvIKQt2jWB9Pd3
kmPCstgLxYBc0jnRp9MbH36dyN+c40a1JyxvCVDIDDnmd4dNR7VNs9Aizwpns9T
n+wkBU2m4lb0kQjTLXfFp6B9VJaQMa/8aBJvY15ucPX43DZ/DyVTS/gXr8iMdTOU
59JMEAg3nyDkWPpB4BmDkFTu431fAgMBAAEcGyAKXIATjDXpD/63SkHNx8G5bxSz
ymhmWDAveskvhOfUJA5UgrRoahmdCwrVlO/OfKw92AFS81twpm9TxpEe25p6Wpev
ujL99yOzxAMnMwqfi7BvSF1TPp3eu7LkFN8FWnXO1XkXb++WeVLEQ6WeGcj5Kv38
3TnUAsaRi9T1NVU4sQJBA00ZbTD6nQPnmDfrNgLNcsy+JFN7MzEOTNqsfLuxynLj
9VoTxNvFsbhGfW1G675jBIJaTOoYTou+Khy5rmYRB6cCQQDA04YtJZD2p3InOWWe
aLm/9mxxUDvcur5L/NhjtOeWgZ2ekIIVoMXDyVrPj/PVdMKfFhoQUkj6Am+s/EI
wxOJAkEAg4GECNfVX4sydaTvSUFQrDetmmqE38qwwMFA0JgMnAqtMhVZ5Lb9Biu
ojRnRFNdNL0/tBVfUVGw7XYQlaNXYwJAJWbiF8+5lp5UHhecBmX54apCzBJkCiS0
1N5ueq2bjpOXvVNpi4R8WPSwYzHVkTnsfZQw8cf4D+bqPPtaPYDZQQJBANHbiUNH
R+78c4QiT26+/bjBwzJ413eozzW1fHgvfZzcT8wyd9PlieZgRmuN4RqFvkMrRByJ
gXHBV5rI5cq2qiA=
-----END PRIVATE KEY-----
```

PERL

In order to use the RSA functions in the PERL programming environment it is necessary to install two PERL modules:

- Crypt::OpenSSL::RSA (<http://search.cpan.org/~iroberts/Crypt-OpenSSL-RSA-0.21/RSA.pm>)
- Crypt::OpenSSL::X509 (<http://search.cpan.org/~daniel/Crypt-OpenSSL-X509-0.2/X509.pm>)

Signing

```
#!/usr/bin/perl
use Crypt::OpenSSL::RSA;
use MIME::Base64;

# -- reading the private key file
open (FH, "priv.pem");
my $key_string;
while (<FH> {
    $key_string .= $_;
```

```

}
close FH;

# -- line to be signed
my $mac =
"0041001003008005DIENA0060123450049.99003LTLO111000000123400573000
017UAB SAULĖTA DIENA026Apmokėjimas pagal užsakymą";

# -- creating a private key object
my $rsa_priv = Crypt::OpenSSL::RSA->new_private_key($key_string);

# -- signing the line
my $signature = $rsa_priv->sign($mac);

# -- encoding the signature with Base64 algorithm,
# -- deleting new line symbols from the encoded signature
(my $vk_mac = encode_base64($signature)) =~ s/[\r\n]//g;
print $vk_mac;

```

Signature verification

```

#!/usr/bin/perl
use Crypt::OpenSSL::RSA;
use Crypt::OpenSSL::X509;
use MIME::Base64;

# -- reading the certificate file
my $x509 = Crypt::OpenSSL::X509->new_from_file('cert.pem');

# -- obtaining the public key
my $KeyString = $x509->pubkey();

# -- signature line
my $signature =
"ZV12AHE8WX9R5XifPjUZ2utUEEOZ4a3fK3RFBRTI1mdEwrnEA0tLRtDbIhrYU1DTzdruLW
p9i5uASII//WdkRvoHesPqyzVjI3wFqbIVUNsp796p2Uh44TrBbG3oVBODP3qwD210Zp8Sg
9ar50c3251LNImZc3Jts5vGtYu6IMM=";

# -- creating a public key object
my $RsaPub = Crypt::OpenSSL::RSA->new_public_key($KeyString);

# -- forming a signed line
my $vk_mac =
"0041001003008005DIENA0060123450049.99003LTLO111000000123400573000
017UAB SAULĖTA DIENA026Apmokėjimas pagal užsakymą";

# -- verifying the signature
if ($RsaPub->verify($vk_mac, decode_base64($signature))) {
    print "Good signature";
}
else {

```

```
print "Bad signature";
}
```

PHP

In order to use the RSA functions in the PERL programming environment it is necessary to compile PHP with the OpenSSL support.

Signing

```
<?php
// -- reading the private key file
$fp = fopen("priv.pem", "r");
$priv_key = fread($fp, 8192);
fclose($fp);

// -- creating a private key resource
$pkeyid = openssl_get_privatekey($priv_key);

// -- line to be signed
$vk_mac =
"0041001003008005DIENA0060123450049.99003LTL011100000123400573000
017UAB SAULÉTA DIENA026Apmokėjimas pagal užsakymą";

// -- signing the line
openssl_sign($data, $signature, $pkeyid);

// -- encoding the signature with the Base64 algorithm,
$signature = base64_encode($signature);

// -- deleting new line symbols from the encoded signature
$signature = preg_replace("/[\r|\n]/g", "", $signature);

// -- clearing memory
openssl_free_key($pkeyid);
?>
```

Signature verification

```
<?php
// -- reading the certificate file
$fp = fopen("cert.pem", "r");
$cert = fread($fp, 8192);
fclose($fp);

// -- obtaining the public key
$pubkeyid = openssl_get_publickey($cert);

// -- signature line
$signature =
"ZV12AHE8WX9R5XifPjUZ2utUEEOZ4a3fK3RFBRTI1mdEwrnEA0tLRtDbIhrYU1DTzdruLW"
```

```
p9i5uASII//WdkRvoHesPqyzVjl3wFqbIVUNsp796p2Uh44TrBbG3oVBODP3qwD210Zp8Sg
9ar50c3251LNlmZc3Jts5vGtYu6IMM=";

// -- forming the line to be signed
$vk_mac =
"0041001003008005DIENA0060123450049.99003LTL0111000000123400573000
017UAB SAULĖTA DIENA026Apmokėjimas pagal užsakymą";

// -- checking the signature
$ok = openssl_verify($vk_mac, base64_decode($signature), $pubkeyid);
if($ok == 1) {
    echo "Good signature";
} elseif($ok == 0) {
    echo "Bad signature";
} else {
    echo "Error checking signature";
}
openssl_free_key($pubkeyid);
?>
```

Java

In order to use the RSA functions in the Java programming environment, a package with RSA cryptographic functions is required, for example, the open code project “Bouncy Castle” (org.bouncycastle.jce.provider.BouncyCastleProvider) – <http://www.bouncycastle.org>.

Signing

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;

import java.security.Security;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.InvalidKeySpecException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.Signature;
import java.security.InvalidKeyException;
import java.security.SignatureException;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.*;

public class ExampleSign {

public static void main(String[] args) {
    // -- signing the line
    // -- [here in the class file the line is kept in the utf-8 encoding,
```

```

// -- and has to be checked as a sequence of bytes corresponding to the encoding used in the
HTML form;
// -- in the example it is assumed that the HTTPS query with the HTML form parameters is
sent in the Windows-1257 encoding)
String vk_mac =
    "0041001003008005DIENA0060123450049.99003LTLO1110000001234005730
00017UAB SAULĖTA DIENA026Apmokėjimas pagal užsakymą";

// -- reading the PEM file of the private key (PKCS#8)
byte[] prkb = null;
StringBuffer sbuf = new StringBuffer();
try {
    BufferedReader in = new BufferedReader(new FileReader("priv.pem"));
    String line;
    while ((line = in.readLine()) != null) {
        if (line.equals("-----BEGIN PRIVATE KEY-----")) {
            break;
        }
    }
    while ((line = in.readLine()) != null) {
        if (line.equals("-----END PRIVATE KEY-----")) {
            break;
        }
        sbuf.append(line);
    }
    prkb = Base64.decode(sbuf.toString());
} catch (FileNotFoundException e) { //FileReader
    e.printStackTrace();
} catch (IOException e) { //readLine
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}

try {
    // -- setting realisation of the cryptographic algorithms
    Security.addProvider(new BouncyCastleProvider());

    // -- interpreting private key data
    PKCS8EncodedKeySpec prks = new PKCS8EncodedKeySpec(prkb);
    KeyFactory kf = KeyFactory.getInstance("RSA");
    RSAPrivateKey prk = (RSAPrivateKey) kf.generatePrivate(prks);

    // -- signing the line
    Signature s = Signature.getInstance("SHA1withRSA");
    s.initSign(prk);
    s.update(vk_mac.getBytes("windows-1257"));
    byte[] sig = s.sign();

    // -- encoding the signature with the Base64 algorithm
    System.out.println(new String(Base64.encode(sig)));
} catch (SecurityException e) { //addProvider

```

```

    e.printStackTrace();
} catch (NoSuchAlgorithmException e) { //getInstance
    e.printStackTrace();
} catch (InvalidKeySpecException e) { //generatePrivate
    e.printStackTrace();
} catch (InvalidKeyException e) { //initSign
    e.printStackTrace();
} catch (SignatureException e) { //update, sign
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
}
}
}

```

Signature verification

```

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.interfaces.RSAPublicKey;

import java.security.Security;
import java.security.Signature;
import java.security.NoSuchAlgorithmException;
import java.security.InvalidKeyException;
import java.security.SignatureException;

import org.bouncycastle.jce.provider.BouncyCastleProvider;
import org.bouncycastle.util.encoders.*;

public class ExampleVerify {

    public static void main(String[] args) {

        // -- signed line
        // -- (here in the class file the line is kept in the utf-8 encoding,
        // -- and has to be checked as a sequence of bytes corresponding to the encoding used in the
        // -- HTML form;
        // -- in the example it is assumed that the HTTPS query with the HTML form parameters is
        // -- sent in the Windows-1257 encoding)
        String vk_mac =
            "0041001003008005DIENA0060123450049.99003LTL01110000001234005730
            00017UAB SAULÉTA DIENA026Apmokėjimas pagal užsakymą";

        // -- signature line
        String sigb64 =
            "ZV12AHE8WX9R5XifPjUZ2utUEEOZ4a3fK3RFBRT11mdEwrnEA0tLRtDbIhrYU1DTzdru
            LWp9i5uASII//WdkRvoHesPqyzVjl3wFqbIVUNsp796p2Uh44TrBbG3oVBODP3qwD210Zp8
            Sg9ar50c3251LNlMzc3Jts5vGtYu6IMM=";
    }
}

```

```

// -- reading the PEM file of the certificate (X.509) and obtaining the public key
RSAPublicKey pbk = null;
try {
    FileInputStream fis = new FileInputStream("cert.pem");
    CertificateFactory cf = CertificateFactory.getInstance("X.509");
    X509Certificate c = (X509Certificate) cf.generateCertificate(fis);
    pbk = (RSAPublicKey) c.getPublicKey();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (CertificateException e) {
    e.printStackTrace();
}
try {
    // -- setting realisation of the cryptographic algorithms
    Security.addProvider(new BouncyCastleProvider());

    // -- decoding the signature line
    byte[] sig = Base64.decode(sigb64);

    // -- checking the signature
    Signature s = Signature.getInstance("SHA1withRSA");
    s.initVerify(pbk);
    s.update(vk_mac.getBytes("windows-1257"));
    if (s.verify(sig))
        System.out.println("Good signature");
    else
        System.out.println("Bad signature");

} catch (SecurityException e) { //addProvider
    e.printStackTrace();
} catch (NoSuchAlgorithmException e) { //getInstance
    e.printStackTrace();
} catch (InvalidKeyException e) { //initVerify
    e.printStackTrace();
} catch (SignatureException e) { //update; verify
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```