



PL/PDF Native Digital Signature

v5.31

Setup, key files, optional signature fields, PAdES Baseline B/B-T, and examples

1. Purpose and scope

The PLPDF_DIGSIG package creates detached CMS digital signatures for PL/PDF documents entirely in native Oracle PL/SQL. It replaces the previous Java stored procedure and Bouncy Castle dependency. PL/PDF creates the PDF signature structure, ByteRange values, and reserved signature contents area; PLPDF_DIGSIG performs the cryptographic signing operation and returns the detached CMS object. A visible AcroForm signature field is optional. If no signature field ID is supplied to PLPDF.SETDIGSIG, PL/PDF automatically creates an internal hidden signature field so that the signature remains discoverable by standard PDF viewers.

PL/PDF v5.31 supports the existing legacy CMS/PKCS#7 signature profile, PAdES Baseline B (PAdES-B-B), and PAdES Baseline T (PAdES-B-T). All profiles use SHA-256 with RSA; PAdES-B-T additionally embeds an RFC 3161 signature timestamp.

2. Signing flow

- PL/PDF creates the unsigned PDF and reserves the signature contents area. If the application does not supply a signature field, PL/PDF creates an internal hidden signature field automatically.
- PL/PDF calculates the PDF ByteRange and supplies the exact bytes outside the reserved signature placeholder to PLPDF_DIGSIG.
- PLPDF_DIGSIG calculates a SHA-256 digest of the ByteRange BLOB with DBMS_CRYPTO.HASH.
- PLPDF_DIGSIG builds the CMS signed attributes required by the selected signature profile.
- DBMS_CRYPTO.SIGN signs the DER-encoded signed attributes with the RSA private key using SIGN_SHA256_RSA.
- PLPDF_DIGSIG builds the detached CMS SignedData object and includes the signer certificate.



PL/PDF OffX v5

- PL/PDF writes the CMS bytes into the reserved signature area and returns the final signed PDF BLOB.

The private key is never written into the PDF. Only the signer certificate, signed attributes, and RSA signature value are stored in the CMS object.

3. Current supported configuration

- Oracle Database 21c or later.
- RSA private key.
- SHA-256 document digest and RSA PKCS#1 v1.5 signature.
- Unencrypted PKCS#1 PEM private-key file whose first line is -----BEGIN RSA PRIVATE KEY-----.
- Binary DER-encoded X.509 signer certificate.
- One signer certificate embedded in the CMS object.
- Legacy detached CMS/PKCS#7, PAdES Baseline B (PAdES-B-B), and PAdES Baseline T (PAdES-B-T) signature profiles.
- 2048-bit or larger RSA key recommended.

Current limitations

- PKCS#8 private keys beginning with -----BEGIN PRIVATE KEY----- are rejected and must be converted to PKCS#1.
- Encrypted private keys are not supported by the current package.
- The current CMS object contains only one signer certificate; an intermediate certificate chain is not yet embedded.
- PAdES-B-LT and PAdES-B-LTA are not included. PAdES-B-T provides RFC 3161 signature timestamping, but OCSP/CRL validation data and long-term validation information are not embedded.
- The implementation currently supports RSA/SHA-256 only.

4. Required key files

The signing operation requires two separate BLOB values.



PL/PDF OffX v5

private-rsa.pem

An unencrypted RSA private key in traditional PKCS#1 PEM format. The file must begin and end with the following markers:

```
-----BEGIN RSA PRIVATE KEY-----
```

```
-----END RSA PRIVATE KEY-----
```

Do not pass a PFX/P12 file directly. Do not pass a PKCS#8 file beginning with -----BEGIN PRIVATE KEY-----. PLPDF_DIGSIG removes the PEM markers and whitespace and supplies the Base64 key body to DBMS_CRYPTOSIGN.

cert.der

The matching signer certificate in binary X.509 DER format. This is binary ASN.1 data, not a text PEM certificate. A valid DER certificate normally starts with ASN.1 SEQUENCE byte 30 in hexadecimal.

The certificate must contain the public key corresponding to private-rsa.pem. A self-signed certificate is suitable for functional testing, but PDF viewers will not trust it unless the certificate is explicitly trusted. Production use should use an appropriate trusted signing certificate.

5. Creating a new test key and self-signed certificate

Create a 2048-bit traditional PKCS#1 RSA private key:

```
openssl genrsa -traditional -out private-rsa-2048.pem 2048
```

Create a self-signed X.509 certificate directly in DER format:

```
openssl req -new -x509 -key private-rsa-2048.pem -outform DER -out cert-2048.der -days 3650 -sha256 -subj "/C=HU/O=PLPDF/CN=PLPDF Test"
```

The resulting files can be used directly by the current package: private-rsa-2048.pem is passed to setPEMStore and cert-2048.der is passed to setDERStore.

6. Extracting the required files from an existing PFX/P12 file

Extract the unencrypted private key from the PFX file:

```
openssl pkcs12 -in signing-certificate.pfx -nocerts -noenc -out private.pem
```

Convert the extracted key to traditional PKCS#1 RSA PEM:

```
openssl pkey -in private.pem -traditional -out private-rsa.pem
```

Extract only the end-entity signer certificate:

```
openssl pkcs12 -in signing-certificate.pfx -clcerts -nokeys -out cert.pem
```



PL/PDF OffX v5

Convert the certificate from PEM to binary DER:

```
openssl x509 -in cert.pem -outform DER -out cert.der
```

OpenSSL prompts for the PFX password. If OpenSSL 3 cannot open an older PFX because it uses legacy algorithms, retry the relevant pkcs12 command with the -legacy option.

Legacy private-key extraction example:

```
openssl pkcs12 -legacy -in signing-certificate.pfx -nocerts -noenc -out private.pem
```

7. Creating a certificate signing request for a CA

Create a CSR using the PKCS#1 private key:

```
openssl req -new -key private-rsa.pem -out signing-request.csr -sha256 -subj "/C=HU/O=Your Company/CN=PDF Signing"
```

Submit signing-request.csr to the certificate authority. If the returned signer certificate is PEM, convert it to DER:

```
openssl x509 -in signer-certificate.pem -outform DER -out cert.der
```

The returned certificate must correspond to the same private key used to create the CSR. Intermediate CA certificates are not embedded by the current package baseline.

8. Checking the key and certificate

Check the private-key structure:

```
openssl pkey -in private-rsa.pem -check -noout
```

Display the DER certificate:

```
openssl x509 -inform DER -in cert.der -text -noout
```

Calculate a digest of the RSA modulus from the private key:

```
openssl rsa -in private-rsa.pem -noout -modulus | openssl sha256
```

Calculate a digest of the RSA modulus from the certificate:

```
openssl x509 -inform DER -in cert.der -noout -modulus | openssl sha256
```

The two modulus digests must be identical. Different results mean that the certificate and private key do not belong to the same key pair.

9. Storing the files for the test procedure

The supplied test1_digsig procedure expects a table containing the two files as BLOB values:

```
create table digsigtest (name1 varchar2(255), file1 blob);
```

The test expects these exact row names:



- cert-2048.der — the binary DER certificate.
- private-rsa-2048.pem — the unencrypted PKCS#1 PEM private key.

Load the files with the existing PL/PDF test-file upload mechanism, SQL Developer, or another BLOB loader. The key and certificate are selected by name at the beginning of the test procedure.

10. test3_digsig procedure - signature without a visible field

Current test procedure:

create or replace procedure test3_digsig is

```
l_blob    blob;
l_text    varchar2(255 char) := 'Hello World!';
l_der     blob;
l_pem     blob;
```

begin

```
select t.file1 into l_der
  from digsigtest t
 where name1 = 'cert-2048.der';
```

```
select t.file1 into l_pem
  from digsigtest t
 where name1 = 'private-rsa-2048.pem';
```

```
plpdf.init();
plpdf.NewPage;
plpdf.SetPrintFont('Arial', null, 12);
plpdf.PrintCell(100, 20, l_text);
```

```
plpdf_digsig.setDERStore(l_der);
plpdf_digsig.setPEMStore(l_pem);
```

```
plpdf.setDigSig(
  p_access_perms => 2,
  p_Name         => 'Name1',
  p_Location     => 'Location1',
```



PL/PDF OffX v5

```
p_Reason    => 'Reason1',
p_ContactInfo => 'Contact1'
);

plpdf.SendDoc(l_blob);

delete from STORE_BLOB;
insert into STORE_BLOB (blob_file, created_date)
values (l_blob, sysdate);
commit;
end;
/
```

How the test works

1. The procedure reads cert-2048.der and private-rsa-2048.pem from DIGSIGTEST into BLOB variables.
2. PLPDF.INIT starts a new PDF document, NewPage creates the first page, and PrintCell writes “Hello World!”.
3. setDERStore stores the signer certificate in the session-specific PLPDF_DIGSIG package state.
4. setPEMStore stores the private key in the same package state.
5. setDigSig configures the digital signature. Because p_id is omitted, PL/PDF automatically creates an internal hidden signature field.
6. The hidden field is used for PDF viewer interoperability; no visible signature rectangle is required.
7. SendDoc finalizes the PDF, creates the ByteRange input, calls PLPDF_DIGSIG, receives the CMS signature BLOB, and inserts it into the PDF.
8. The final signed PDF BLOB is stored in STORE_BLOB.

11. PAdES Baseline B (PAdES-B-B)

PL/PDF v5.31 supports PAdES Baseline B as an explicit digital-signature profile. PAdES-B-B is the minimum PAdES Baseline level and is intended for standards-based PDF digital signatures without requiring an external timestamp authority or online revocation services.



PL/PDF OffX v5

Select the PAdES-B-B profile before SendDoc:

```
plpdf_digsig.setProfile(plpdf_digsig.c_profile_pades_bb);
```

The certificate and private-key setup is the same as for the legacy signature profile.

The existing setDERStore and setPEMStore calls are used.

PAdES-B-B uses the same optional signature-field behavior as the legacy profile. The application may pass a field ID to PLPDF.SETDIGSIG, or omit p_id and let PL/PDF create an internal hidden signature field automatically.

12. PAdES-B-B example

To create a PAdES-B-B signature, add the profile-selection call before configuring the certificate and private key. The rest of the signing flow remains unchanged:

```
plpdf_digsig.setProfile(plpdf_digsig.c_profile_pades_bb);
```

```
plpdf_digsig.setDERStore(l_der);
```

```
plpdf_digsig.setPEMStore(l_pem);
```

```
plpdf.setDigSig(
```

```
    p_access_perms => 2,
```

```
    p_Name        => 'Name1',
```

```
    p_Location    => 'Location1',
```

```
    p_Reason      => 'Reason1',
```

```
    p_ContactInfo => 'Contact1'
```

```
);
```

If setProfile is not called, the existing legacy signature profile remains the default for backward compatibility.

13. PAdES-B-B validation

The PAdES-B-B output has been validated with Adobe Acrobat Reader, Foxit Reader, the ETSI Signature Conformance Checker, and the European Commission Digital Signature Service (DSS). DSS identifies the generated signature format as PAdES-BASELINE-B.

A self-signed test certificate can produce an indeterminate trust result because no trusted certificate chain is available. This is a certificate trust issue and does not change the PAdES-B-B signature format. Production deployments should use an appropriate trusted signing certificate.



PL/PDF OffX v5

Session behavior

PLPDF_DIGSIG stores the key, certificate, original ByteRange data, and generated signature in package variables. Oracle package state is session-specific, so `setDERStore`, `setPEMStore`, and `plpdf.SendDoc` must run in the same database session. Different Oracle sessions have independent package state.

14. PAdES Baseline T (PAdES-B-T)

PAdES-B-T extends PAdES-B-B with an RFC 3161 signature timestamp token. The token is added to the CMS signature as an unsigned attribute and provides independent evidence that the signature existed at the time asserted by the timestamp authority (TSA).

PAdES-B-T does not add embedded OCSP or CRL material and does not provide the long-term validation data required by PAdES-B-LT or PAdES-B-LTA.

Select the PAdES-B-T profile, then configure exactly one timestamp mode before `PLPDF.SendDoc`: the native TSA client or an installed timestamp procedure.

```
plpdf_digsig.setProfile(plpdf_digsig.c_profile_pades_bt);
```

Network ACL for TSA URL access

An Oracle network ACL must allow the database schema that performs the HTTP request to reach the TSA host. Run the following once as SYS or another suitably privileged DBA account. Replace `PLPDF_OWNER`, `freetsa.org`, and port 443 with the actual signing schema and TSA endpoint. The ACL host value is a host name only; do not include `https://` or the `/tsr` path.

```
GRANT EXECUTE ON SYS.UTL_HTTP TO PLPDF_OWNER;
```

```
BEGIN
```

```
DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(  
  host      => 'freetsa.org',  
  lower_port => 443,  
  upper_port => 443,  
  ace       => xs$ace_type(  
    privilege_list => xs$name_list('connect'),  
    principal_name => 'PLPDF_OWNER',
```



PL/PDF OffX v5

```
principal_type => xs_acl.ptype_db
)
);

DBMS_NETWORK_ACL_ADMIN.APPEND_HOST_ACE(
  host => 'freetsa.org',
  ace => xs$ace_type(
    privilege_list => xs$name_list('resolve'),
    principal_name => 'PLPDF_OWNER',
    principal_type => xs_acl.ptype_db
  )
);
END;
/
```

The connect ACE is restricted to TCP port 443. The resolve ACE is intentionally added without a port range so that Oracle can resolve the TSA host name.

The ACL authorizes the network destination; the Oracle wallet establishes HTTPS trust.

The wallet path is resolved on the database server, not on the client workstation.

Import the TSA server's trusted certificate chain into the wallet. An auto-login wallet can use a null password; otherwise provide the wallet password through the application's secure configuration.

The same ACL rule applies to the procedure mode when the registered timestamp procedure calls the TSA through UTL_HTTP. In that case, grant the ACEs to the database user under which that HTTP call executes.

Native TSA client

In native mode, PLPDF_DIGSIG sends the RFC 3161 request directly to the configured TSA URL. Configure the URL, database-server wallet, wallet password, and timeout with `setTimestampServer` after selecting the PAdES-B-T profile.

The following complete example is taken from the supplied `test5_digsig_pades_bt_native_tsa` procedure. The Windows wallet path is an example and must be changed to a valid wallet on the database server.

create or replace procedure `test5_digsig_pades_bt_native_tsa` is
 l_blob blob;



PL/PDF OffX v5

```
l_der blob;
l_pem blob;
begin
  select file1 into l_der
    from digsigtest
   where name1 = 'cert-2048.der';

  select file1 into l_pem
    from digsigtest
   where name1 = 'private-rsa-2048.pem';

  plpdf.init();

  plpdf_digsig.setProfile(plpdf_digsig.c_profile_pades_bt);

  plpdf_digsig.setTimestampServer(
    p_url      => 'https://freetsa.org/tsr',
    p_wallet_path => 'file:C:\oracle_wallet',
    p_wallet_password => null,
    p_timeout    => 30
  );

  plpdf_digsig.setDERStore(l_der);
  plpdf_digsig.setPEMStore(l_pem);

  plpdf.NewPage;
  plpdf.SetPrintFont('Arial',null,12);
  plpdf.PrintCell(100,20,'Hello PAdES-B-T native TSA!');

  plpdf.setDigSig(
    p_access_perms => 2,
    p_Name         => 'PLPDF',
    p_Reason       => 'PAdES-B-T native TSA test'
```



PL/PDF OffX v5

```
);

plpdf.SendDoc(l_blob);

delete from STORE_BLOB;
insert into STORE_BLOB(blob_file,created_date)
values(l_blob,sysdate);
commit;

dbms_output.put_line('PDF_BYTES'||dbms_lob.getlength(l_blob));
dbms_output.put_line('PLPDF_PADES_BT_NATIVE_TSA=PASS');
end;
/
```

Timestamp procedure

In procedure mode, PLPDF_DIGSIG delegates timestamp acquisition to an installed PL/SQL timestamp procedure. Register its name with `setTimestampProc` after selecting the PAdES-B-T profile. This is useful when an existing wrapper handles the HTTP call, authentication, proxy settings, auditing, or TSA-specific processing.

The example below expects PLPDF_FREETSA_TIMESTAMP to be installed and to implement the timestamp-procedure contract required by the current PLPDF_DIGSIG package. If that procedure performs an outbound HTTP request, configure its executing schema with the ACL shown above and configure its own HTTPS wallet handling.

create or replace procedure test5_digsig_pades_bt_proc is

```
l_blob blob;
l_der blob;
l_pem blob;
begin
select file1
into l_der
from digsigtest
where name1 = 'cert-2048.der';

select file1
```



PL/PDF OffX v5

```
into l_pem
from digsigtest
where name1 = 'private-rsa-2048.pem';
```

```
plpdf.init();
```

```
plpdf_digsig.setProfile(
  plpdf_digsig.c_profile_pades_bt
);
```

```
plpdf_digsig.setTimestampProc(
  'PLPDF_FREETSA_TIMESTAMP'
);
```

```
plpdf_digsig.setDERStore(l_der);
plpdf_digsig.setPEMStore(l_pem);
```

```
plpdf.NewPage;
plpdf.SetPrintFont('Arial', null, 12);
plpdf.PrintCell(
  100,
  20,
  'Hello PAdES-B-T!'
);
```

```
plpdf.setDigSig(
  p_access_perms => 2,
  p_Name         => 'PLPDF',
  p_Reason       => 'PAdES-B-T test'
);
```

```
plpdf.SendDoc(l_blob);
```



PL/PDF OffX v5

```
delete from STORE_BLOB;

insert into STORE_BLOB(
  blob_file,
  created_date
)
values(
  l_blob,
  sysdate
);

commit;

dbms_output.put_line(
  'PDF_BYTES=' || dbms_lob.getlength(l_blob)
);

dbms_output.put_line(
  'PLPDF_PADES_BT_UNSIGNED_ATTRS=PASS'
);
end;
/
```

Choosing and validating the timestamp mode

- Native mode: use `setTimestampServer` when `PLPDF_DIGSIG` should call the TSA directly and the URL, wallet, and timeout can be configured in the signing flow.
- Procedure mode: use `setTimestampProc` when a separate PL/SQL integration should obtain the timestamp token.
- Use one timestamp mode for a signing operation. Select `c_profile_pades_bt` and configure the timestamp provider before `PLPDF.SendDoc` finalizes the PDF.
- Validate the result with a PAdES-aware validator. The expected level is PAdES-BASELINE-T; also verify the TSA certificate chain and the RFC 3161 token status.



PL/PDF OffX v5

The PASS messages in the examples are test markers, not a substitute for independent conformance validation.

15. Package API used by PL/PDF

- `setDERStore(p_digsig_derstore BLOB)`: stores the binary DER signer certificate.
- `setPEMStore(p_digsig_pemstore BLOB)`: stores the unencrypted PKCS#1 PEM private key.
- `setProfile(p_profile)`: selects the legacy, PAdES-B-B, or PAdES-B-T signature profile. The legacy profile is the default.
- `setTimestampServer(p_url, p_wallet_path, p_wallet_password, p_timeout)`: configures the native RFC 3161 TSA client, including its HTTPS wallet and timeout.
- `setTimestampProc('PROCEDURE_NAME')`: selects an installed timestamp procedure that obtains and returns the RFC 3161 timestamp token.
- `setOriginaldata(p_original_data BLOB)`: receives the exact detached PDF ByteRange content from PL/PDF.
- `PKCS7Sign`: creates the detached CMS signature using SHA-256 and RSA according to the selected signature profile.
- `getSigneddata RETURN BLOB`: returns the CMS signature BLOB to PL/PDF.

The spelling `setOriginaldata` is retained for compatibility with the current PL/PDF integration.

16. Troubleshooting

ORA-28817 from DBMS_CRYPTO.SIGN

- Confirm that the private key begins with -----BEGIN RSA PRIVATE KEY-----.
- Do not pass the complete PFX/P12 file to `setPEMStore`.
- Convert PKCS#8 to PKCS#1 with: `openssl pkey -in private.pem -traditional -out private-rsa.pem`
- Confirm that the private key is not encrypted.
- Confirm that PEM markers and whitespace are removed by `private_key_pem_to_raw` before `DBMS_CRYPTO.SIGN`.



PL/PDF OffX v5

- Use a 2048-bit or larger RSA key.

PDF signature is cryptographically invalid

- Verify that cert.der and private-rsa.pem belong to the same key pair.
- Verify that PL/PDF signs the exact bytes identified by /ByteRange.
- Verify that the generated CMS object fits into the /Contents reservation.
- Do not modify the PDF after SendDoc unless an incremental update is intentionally applied.

Signature is valid but not trusted

This is expected with a self-signed test certificate. Use a certificate trusted by the PDF viewer or explicitly add the test certificate to the viewer's trust store. A trusted production deployment may also require embedding intermediate certificates in a future package version.

ORA-24247: network access denied by access control list (ACL)

- Grant connect on the TSA host and port to the database user that performs the UTL_HTTP call.
- Grant resolve on the TSA host without a port range.
- Use only the host name in APPEND_HOST_ACE; do not include the URL scheme or path.
- For procedure mode, confirm which schema executes the outbound HTTP request and grant the ACL to that user.

HTTPS or TSA request fails

- Confirm that the database server can resolve and reach the TSA host on the configured port.
- Confirm that p_wallet_path points to a wallet on the database server and that the wallet trusts the TSA server certificate chain.
- Confirm the wallet password, timeout, proxy, and TSA URL settings required by the environment.



PL/PDF OffX v5

- Check the database error stack and TSA HTTP response before retrying; do not silently produce a PAdES-B-B signature when PAdES-B-T was requested.

Timestamp procedure cannot be called

- Confirm that the procedure named in `setTimestampProc` is installed and matches the timestamp-procedure contract required by the current `PLPDF_DIGSIG` package.
- Confirm that the signing schema can resolve and execute the procedure and that the procedure's dependencies are valid.
- If the procedure calls a TSA URL, verify its ACL and wallet configuration separately from the PL/PDF signing flow.

17. Security considerations

- The current private key is unencrypted because the PL/SQL package does not support encrypted key files.
- Restrict `SELECT` access to the table or secure store containing the private-key BLOB.
- Do not print the private key with `DBMS_OUTPUT`, write it to logs, or include it in error messages.
- Use a dedicated signing schema or tightly controlled package API rather than granting direct table access.
- Use a separate test key for development and never reuse a production signing key in test environments.
- Protect backups containing the private-key BLOB.
- Plan key rotation and certificate expiration handling before production deployment.
- Restrict the network ACL to the exact TSA host and port required by the deployment.
- Protect the Oracle wallet and wallet password with the same care as other signing-service credentials.



PL/PDF OffX v5