public_key

version 0.2
Typeset in \LaTeX{} from SGML source using the DocBuilder-0.9.8.4 Document System.
Contents

1 public_key User's Guide ............................. 1
  1.1 Introduction ........................................ 1
    1.1.1 Purpose ........................................ 1
    1.1.2 Prerequisites .................................. 1
  1.2 Public key records ................................... 1
    1.2.1 RSA as defined by the PKCS-1 standard and RFC 3447. .... 2
    1.2.2 DSA as defined by Digital Signature Standard (NIST FIPS PUB 186-2) .... 2
  1.3 Certificate records .................................. 2
    1.3.1 Common Data Types ............................... 3
    1.3.2 PKIX Certificates ............................... 3
    1.3.3 Standard certificate extensions .................... 5
    1.3.4 Private Internet Extensions ...................... 8
    1.3.5 CRL and CRL Extensions Profile .................... 8

2 public_key Reference Manual ......................... 11
  2.1 public_key .......................................... 12

List of Tables ........................................... 15
Chapter 1

public_key User’s Guide

This application provides an API to public key infrastructure from RFC 3280 (X.509 certificates) and some public key formats defined by the PKCS-standard.

1.1 Introduction

1.1.1 Purpose

This application provides an API to public key infrastructure from RFC 3280 (X.509 certificates) and public key formats defined by the PKCS-standard.

1.1.2 Prerequisites

It is assumed that the reader is familiar with the Erlang programming language, concepts of OTP and has a basic understanding of the concepts of using public keys.

1.2 Public key records

This chapter briefly describes Erlang records derived from asn1 specifications used to handle public and private keys. The intent is to describe the data types and not to specify the meaning of each component for this we refer you to the relevant standards and RFCs.

Use the following include directive to get access to the records and constant macros used in the following sections.

-include_lib("public_key/include/public_key.hrl").
Chapter 1: public_key User's Guide

1.2.1 RSA as defined by the PKCS-1 standard and RFC 3447.

#'RSAPublicKey'{
    modulus, % integer()
    publicExponent % integer()
}.

#'RSAPrivateKey'{
    version, % two-prime | multi
    modulus, % integer()
    publicExponent, % integer()
    privateExponent, % integer()
    prime1, % integer()
    prime2, % integer()
    exponent1, % integer()
    exponent2, % integer()
    coefficient, % integer()
    otherPrimeInfos % [#OtherPrimeInfo{}] | asn1_NOVALUE
}.

#'OtherPrimeInfo'{
    prime, % integer()
    exponent, % integer()
    coefficient % integer()
}.

1.2.2 DSA as defined by Digital Signature Standard (NIST FIPS PUB 186-2)

#'DSAPrivateKey',{  
    version, % integer()
    p, % integer()
    q, % integer()
    g, % integer()
    y, % integer()
    x % integer()
}.

#'Dss-Parms',{  
   p, % integer()
   q, % integer()
   g % integer()
}.

1.3 Certificate records

This chapter briefly describes erlang records derived from asn1 specifications used to handle X 509 certificates. The intent is to describe the data types and not to specify the meaning of each component for this we refer you to RFC 3280.

Use the following include directive to get access to the records and constant macros described in the following sections.
1.3 Certificate records

-include_lib("public_key/include/public_key.hrl").

1.3.1 Common Data Types

Common non standard erlang data types used to described the record fields in the below sections are defined in public key reference manual [page 12] or follows here:

\[
\text{time()} = \text{uct}\_\text{time()} | \text{general}\_\text{time()} \\
\text{uct}\_\text{time()} = \{\text{utc}\_\text{Time}, "YYMMDDHHMMSSZ"\} \\
\text{general}\_\text{time()} = \{\text{general}\_\text{Time}, "YYYYMMDDHHMMSSZ"\}
\]

\[
\text{general}\_\text{name()} = \{\text{rfc822Name}, \text{string()}\} | \{\text{dNSName}, \text{string()}\} | \{\text{x400Address}, \text{string()}\} | \{\text{directoryName}, \{\text{rdnSequence}, \#\text{AttributeTypeAndValue}'\}\}\} | \{\text{eidPartyName}, \text{special}\_\text{string()}\} | \{\text{eidPartyName}, \text{special}\_\text{string()}, \text{special}\_\text{string()}\} | \{\text{uniformResourceIdentifier}, \text{string()}\} | \{\text{ipAddress}, \text{string()}\} | \{\text{registeredId}, \text{oid()}\} | \{\text{otherName}, \text{term()}\}
\]

\[
\text{special}\_\text{string()} = \{\text{teletexString}, \text{string()}\} | \{\text{printableString}, \text{string()}\} | \{\text{universalString}, \text{string()}\} | \{\text{utf8String}, \text{string()}\} | \{\text{bmpString}, \text{string()}\}
\]

\[
\text{dist}\_\text{reason()} = \text{unused} | \text{keyCompromise} | \text{cACompromise} | \text{affiliationChanged} | \text{superseded} | \text{cessationOfOperation} | \text{certificateHold} | \text{privilegeWithdrawn} | \text{aACompromise}
\]

1.3.2 PKIX Certificates

\[
\#'\text{Certificate}'
\]

\[
\begin{align*}
\text{tbsCertificate}, & \quad \#'\text{TBSCertificate}' \\
\text{signatureAlgorithm}, & \quad \#'\text{AlgorithmIdentifier}' \\
\text{signature} & \quad \{0, \text{binary()}\} - \text{asn1 compact bitstring}
\end{align*}
\]

\[
\#'\text{TBSCertificate}'
\]

\[
\begin{align*}
\text{version}, & \quad \text{v1} | \text{v2} | \text{v3} \\
\text{serialNumber}, & \quad \text{integer()} \\
\text{signature}, & \quad \#'\text{AlgorithmIdentifier}' \\
\text{issuer}, & \quad \{\text{rdnSequence}, \#\text{AttributeTypeAndValue}'\}\} \\
\text{validity}, & \quad \#'\text{Validity}' \\
\text{subject}, & \quad \{\text{rdnSequence}, \#\text{AttributeTypeAndValue}'\}\} \\
\text{subjectPublicKeyInfo}, & \quad \#'\text{SubjectPublicKeyInfo}' \\
\text{issuerUniqueID}, & \quad \text{binary()} | \text{asn1_no_value} \\
\text{subjectUniqueID}, & \quad \text{binary()} | \text{asn1_no_value} \\
\text{extensions}, & \quad \{\#'\text{Extension}'\}\}
\end{align*}
\]

\[
\#'\text{AlgorithmIdentifier}'
\]

\[
\begin{align*}
\text{algorithm}, & \quad \text{oid()} \\
\text{parameters} & \quad \text{asn1_der_encoded()}
\end{align*}
\]

\[
\#'\text{SignatureAlgorithm}'
\]

\[
\begin{align*}
\text{algorithm}, & \quad \text{id}\_\text{signature}\_\text{algorithm()} \\
\text{parameters} & \quad \text{public_key_params()}
\end{align*}
\]

public_key
id_signature_algorithm() = ?oid_name_as_erlang_atom for available oid names see table below.
Ex: ?'id-dsa-with-shal'

<table>
<thead>
<tr>
<th>OID name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-dsa-with-shal</td>
</tr>
<tr>
<td>md2WithRSAEncryption</td>
</tr>
<tr>
<td>md5WithRSAEncryption</td>
</tr>
<tr>
<td>sha1WithRSAEncryption</td>
</tr>
<tr>
<td>ecdsa-with-SHA1</td>
</tr>
</tbody>
</table>

Table 1.1: Signature algorithm oids

#'AttributeTypeAndValue'{
  type, % id_attributes()
  value % term()
}.

id_attributes() = ?oid_name_as_erlang_atom for available oid names see table below. Ex: ?'id-at-name'

<table>
<thead>
<tr>
<th>OID name</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-at-name</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-surname</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-givenName</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-initials</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-generationQualifier</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-commonName</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-localityName</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-stateOrProvinceName</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-organizationName</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-title</td>
<td>special_string()</td>
</tr>
<tr>
<td>id-at-dnQualifier</td>
<td>{printableString, string()}</td>
</tr>
<tr>
<td>id-at-countryName</td>
<td>{printableString, string()}</td>
</tr>
<tr>
<td>id-at-serialNumber</td>
<td>{printableString, string()}</td>
</tr>
<tr>
<td>id-at-pseudonym</td>
<td>special_string()</td>
</tr>
</tbody>
</table>

Table 1.2: Attribute oids

#'Validity'{
  notBefore, % time()
  notAfter   % time()
}.

#'SubjectPublicKeyInfo'{
  algorithm, % AlgorithmIdentifier{}
  subjectPublicKey % binary()
1.3: Certificate records

id_public_key_algorithm() = ?oid_name_as_erlang_atom for available oid names see table below.
Ex: ?'id-dsa'

<table>
<thead>
<tr>
<th>OID name</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsaEncryption</td>
<td></td>
</tr>
<tr>
<td>id-dsa</td>
<td></td>
</tr>
<tr>
<td>dhpublicnumber</td>
<td></td>
</tr>
<tr>
<td>ecdsa-with-SHA1</td>
<td></td>
</tr>
<tr>
<td>id-keyExchangeAlgorithm</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.3: Public key algorithm oids

id_extensions() = ?oid_name_as_erlang_atom for available oid names see tables. Ex: ?'id-ce-authorityKeyIdentifier'

1.3.3 Standard certificate extensions

<table>
<thead>
<tr>
<th>OID name</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-ce-authorityKeyIdentifier</td>
<td>#AuthorityKeyIdentifier{}</td>
</tr>
<tr>
<td>id-ce-subjectKeyIdentifier</td>
<td>oid()</td>
</tr>
<tr>
<td>id-ce-keyUsage</td>
<td>[key_usage()]</td>
</tr>
<tr>
<td>id-ce-privateKeyUsagePeriod</td>
<td>#PrivateKeyUsagePeriod{}</td>
</tr>
<tr>
<td>id-ce-certificatePolicies</td>
<td>#PolicyInformation{}</td>
</tr>
<tr>
<td>id-ce-policyMappings</td>
<td>#PolicyMappings_SEQOF{}</td>
</tr>
<tr>
<td>id-ce-subjectAltName</td>
<td>general_name()</td>
</tr>
<tr>
<td>id-ce-issuerAltName</td>
<td>general_name()</td>
</tr>
<tr>
<td>id-ce-subjectDirectoryAttributes</td>
<td>[#Attribute{}]</td>
</tr>
<tr>
<td>id-ce-basicConstraints</td>
<td>#BasicConstraints{}</td>
</tr>
<tr>
<td>id-ce-nameConstraints</td>
<td>#NameConstraints{}</td>
</tr>
</tbody>
</table>

continued ...
Chapter 1: PublicKey User's Guide

... continued

<table>
<thead>
<tr>
<th>Certificate Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-ce-policyConstraints</td>
<td>PolicyConstraints{}</td>
</tr>
<tr>
<td>id-ce-extKeyUsage</td>
<td>[id_key_purpose()]</td>
</tr>
<tr>
<td>id-ce-cRLDistributionPoints</td>
<td>DistributionPoint{}</td>
</tr>
<tr>
<td>id-ce-inhibitAnyPolicy</td>
<td>integer()</td>
</tr>
<tr>
<td>id-ce-freshestCRL</td>
<td>[#DistributionPoint{}]</td>
</tr>
</tbody>
</table>

Table 1.4: Standard Certificate Extensions

key_usage() = digitalSignature | nonRepudiation | keyEncipherment | dataEncipherment | keyAgreement | keyCertSign | cRLSign | encipherOnly | decipherOnly

id_key_purpose() = ?oid_name_as_erlang_atom for available oid names see table below. Ex: ?’id-kp-serverAuth’

<table>
<thead>
<tr>
<th>OID Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-kp-serverAuth</td>
</tr>
<tr>
<td>id-kp-clientAuth</td>
</tr>
<tr>
<td>id-kp-codeSigning</td>
</tr>
<tr>
<td>id-kp-emailProtection</td>
</tr>
<tr>
<td>id-kp-timeStamping</td>
</tr>
<tr>
<td>id-kp-OCSPSigning</td>
</tr>
</tbody>
</table>

Table 1.5: Key purpose oids

#'AuthorityKeyIdentifier'{
    keyIdentifier, % oid()
    authorityCertIssuer, % general_name()
    authorityCertSerialNumber % integer()
}.

#'PrivateKeyUsagePeriod'{
    notBefore, % general_time()
    notAfter  % general_time()
}.

#'PolicyInformation'{
    policyIdentifier, % oid()
    policyQualifiers % [#PolicyQualifierInfo{}]
}.

#'PolicyQualifierInfo'{
    policyQualifierId, % oid()
    qualifier % string() | #'UserNotice'{}
}.

#'UserNotice'{
    noticeRef, % #'NoticeReference'{}
    explicitText % string()
1.3: Certificate records

### NoticeReference

- **organization**: string()
- **noticeNumbers**: [integer()]

### PolicyMappings_SEQOF

- **issuerDomainPolicy**: oid()
- **subjectDomainPolicy**: oid()

### Attribute

- **type**: oid()
- **values**: [asn1_der_encoded()]

### BasicConstraints

- **cA**: boolean()
- **pathLenConstraint**: integer()

### NameConstraints

- **permittedSubtrees**: [#'GeneralSubtree'{}
- **excludedSubtrees**: [#'GeneralSubtree'{}

### GeneralSubtree

- **base**: general_name()
- **minimum**: integer()
- **maximum**: integer()

### PolicyConstraints

- **requireExplicitPolicy**: integer()
- **inhibitPolicyMapping**: integer()

### DistributionPoint

- **distributionPoint**: general_name() | [#AttributeTypeAndValue{}
- **reasons**: [dist_reason()]
- **cRLIssuer**: general_name()
1.3.4 Private Internet Extensions

<table>
<thead>
<tr>
<th>OID name</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-pe-authorityInfoAccess</td>
<td>[#'AccessDescription'{}]</td>
</tr>
<tr>
<td>id-pe-subjectInfoAccess</td>
<td>[#'AccessDescription'{}]</td>
</tr>
</tbody>
</table>

Table 1.6: Private Internet Extensions

1.3.5 CRL and CRL Extensions Profile

#'CertificateList'{
  tbsCertList, % #'TBSCertList{}
  signatureAlgorithm, % #'AlgorithmIdentifier'{}
  signature % {0, binary()} - asn1 compact bitstring
}).

#'TBSCertList'{
  version, % v2 (if defined)
  signature, % #'AlgorithmIdentifier{}
  issuer, % {rdnSequence, [#AttributeTypeAndValue'{}]}%
  thisUpdate, % time()
  nextUpdate, % time()
  revokedCertificates, % [#'TBSCertList_revokedCertificates_SEQOF'{}]%
  crlExtensions % [#'Extension'{}]
}).

#'TBSCertList_revokedCertificates_SEQOF'{
  userCertificate, % integer()
  revocationDate, % timer()
  crlEntryExtensions % [#'Extension'{}]
}).
CRL Extensions

<table>
<thead>
<tr>
<th>OID name</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-ce-authorityKeyIdIdentifier</td>
<td>AuthorityKeyIdIdentifier{}</td>
</tr>
<tr>
<td>id-ce-issuerAltName</td>
<td>{rdnSequence, {#AttributeTypeAndValue'{}]}</td>
</tr>
<tr>
<td>id-ce-cRLNumber</td>
<td>integer()</td>
</tr>
<tr>
<td>id-ce-deltaCRLIndicator</td>
<td>integer()</td>
</tr>
<tr>
<td>id-ce-issuingDistributionPoint</td>
<td>IssuingDistributionPoint'{}</td>
</tr>
<tr>
<td>id-ce-freshestCRL</td>
<td>Distributionpoint'{}</td>
</tr>
</tbody>
</table>

Table 1.7: CRL Extensions

#'IssuingDistributionPoint'{
    distributionPoint, % general_name() | [#AttributeTypeAndValue'{}]
    onlyContainsUserCerts, % boolean()
    onlyContainsCACerts, % boolean()
    onlySomeReasons, % [dist_reason]
    indirectCRL, % boolean()
    onlyContainsAttributeCerts % boolean()
}.

CRL Entry Extensions

<table>
<thead>
<tr>
<th>OID name</th>
<th>Value type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id-ce-cRLReason</td>
<td>crl_reason()</td>
</tr>
<tr>
<td>id-ce-holdInstructionCode</td>
<td>oid()</td>
</tr>
<tr>
<td>id-ce-invalidityDate</td>
<td>general_time()</td>
</tr>
<tr>
<td>id-ce-certificateIssuer</td>
<td>general_name()</td>
</tr>
</tbody>
</table>

Table 1.8: CRL Entry Extensions

crl_reason() = unspecified | keyCompromise | cACompromise | affiliationChanged | superseded | cessationOfOperation | certificateHold | removeFromCRL | privilegeWithdrawn | aACompromise

Short Summaries

- Erlang Module `public_key` (page 12) – API module for public key infrastructure.

public_key

The following functions are exported:

- `decode_private_key(KeyInfo) ->` (page 13) Decodes an asn1 der encoded private key.
- `decode_private_key(KeyInfo, Password) -> {ok, PrivateKey} | {error, Reason}` (page 13) Decodes an asn1 der encoded private key.
- `pem_to_der(File) -> {ok, [Entry]}` (page 13) Reads a PEM file and translates it into its asn1 der encoded parts.
- `pkix_decode_cert(Cert, Type) -> {ok, DecodedCert} | {error, Reason}` (page 13) Decodes an asn1 der encoded pkix certificate.
This module provides functions to handle public key infrastructure from RFC 3280 - X.509 certificates (will later be upgraded to RFC 5280) and some parts of the PKCS-standard. Currently this application is mainly used by the new ssl implementation. The API is yet under construction and only a few of the functions are currently documented and thereby supported.

COMMON DATA TYPES

**Note:** All records used in this manual are generated from asn1 specifications and are documented in the User’s Guide. See Public key records [page 1] and X.509 Certificate records [page 2].

Use the following include directive to get access to the records and constant macros described here and in the User’s Guide.

```
-include_lib("public_key/include/public_key.hrl").
```

**Data Types**

- `boolean()` = true | false
- `string` = [bytes()]
- `asn1:der_encoded()` = binary() | [bytes()]
- `der_bin()` = binary()
- `oid()` - a tuple of integers as generated by the asn1 compiler.
- `public_key()` = rsa_public_key() | dsa_public_key()
- `rsa_public_key()` = #'RSAPublicKey'
- `rsa_private_key()` = #'RSAPrivateKey'
- `dsa_public_key()` = integer()
- `public_key.params()` = dsa_key_params()
- `dsa_key_params()` = #'Dss-Parms'
- `private_key()` = rsa_private_key() | dsa_private_key()
- `rsa_private_key()` = #'RSAPrivateKey'
- `dsa_private_key()` = #'DSAPrivateKey'
- `x509_certificate()` = #'Certificate"

```haskell
x509_tbs_certificate() = '#TBSCertificate'{}
```

**Exports**

```haskell
decode_private_key(KeyInfo) ->
decode_private_key(KeyInfo, Password) -> \{ok, PrivateKey\} | \{error, Reason\}
```

Types:
- `KeyInfo = \{KeyType, der_bin(), CipherInfo\}`
  - `KeyType = rsa_private_key \| dsa_private_key`  
  - `CipherInfo = opaque() \| no_encryption`
    - CipherInfo may contain encryption parameters if the private key is password protected, these are opaque to the user just pass the value returned by pem_to_der/1 to this function.
- `Password = string()`
  - Must be specified if `CipherInfo /= no_encryption`
- `PrivateKey = private_key()`  
- `Reason = term()`

Decodes an asn1 der encoded private key.

```haskell
pem_to_der(File) -> \{ok, [Entry]\}
```

Types:
- `File = path()`  
- `Password = string()`  
- `Entry = \{entry_type(), der_bin(), CipherInfo\}`
  - `CipherInfo = opaque() \| no_encryption`
    - CipherInfo may contain encryption parameters if the private key is password protected, these will be handled by the function decode_private_key/2.
- `entry_type() = cert \| cert_req \| rsa_private_key \| dsa_private_key \| dh_params`

Reads a PEM file and translates it into its asn1 der encoded parts.

```haskell
pkix_decode_cert(Cert, Type) -> \{ok, DecodedCert\} | \{error, Reason\}
```

Types:
- `Cert = asn1_der_encoded()`  
- `Type = plain \| otp`  
- `DecodeCert = x509_certificate()`
  - When `Type` is specified as `otp` the asn1 spec OTP-PKIX.asn1 is used to decode known extensions and enhance the signature field in `Certificate` and `TBSCertificate`. This is currently used by the new ssl implementation but not documented and supported for the public_key application.
- `Reason = term()`

Decodes an asn1 encoded pkix certificate.
List of Tables

1.1 Signature algorithm oids ........................................ 4
1.2 Attribute oids .................................................. 4
1.3 Public key algorithm oids ...................................... 5
1.4 Standard Certificate Extensions ............................... 6
1.5 Key purpose oids ............................................... 6
1.6 Private Internet Extensions ................................... 8
1.7 CRL Extensions .................................................. 9
1.8 CRL Entry Extensions .......................................... 9
Index of Modules and Functions

Modules are typed in this way.
Functions are typed in this way.

decode_private_key/1
  public_key, 13

decode_private_key/2
  public_key, 13

pem_to_der/1
  public_key, 13

pkix_decode_cert/2
  public_key, 13

public_key
  decode_private_key/1, 13
  decode_private_key/2, 13
  pem_to_der/1, 13
  pkix_decode_cert/2, 13