public_key

version 0.1

Typeset in LaTeX from SGML source using the DocBuilder-0.9.8.4 Document System.

Contents

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 consepts 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").
```

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 X509 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.

```
-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.

time() = uct_time() | general_time()

uct_time() = {utcTime, "YYMMDDHHMMSSZ"}

general_time() = {generalTime, "YYYYMMDDHHMMSSZ"}

general_name() = {rfc822Name, string()} | {dNSName, string()} | {x400Address, string()} | {directoryName, {rdnSequence, [#AttributeTypeAndValue'{}]}} | | {eidPartyName, special_string()} | {eidPartyName, special_string(), special_string()} | {uniformResourceIdentifier, string()} | {ipAddress, string()} | {registeredId, oid()} | {otherName, term()}

special_string() = {teletexString, string()} | {printableString, string()} | {universalString, string()} | {utf8String, string()} | {bmpString, string()}

dist_reason() = unused | keyCompromise | cACompromise | affiliationChanged | superseded | cessationOfOperation | certificateHold | privilegeWithdrawn | aACompromise

1.3.2 PKIX Certificates

```
#'Certificate'{
                tbsCertificate,        % #'TBSCertificate'{}
                signatureAlgorithm,    % #'AlgorithmIdentifier'{}
                signature              % {0, binary()} - asn1 compact bitstring
               }.

#'TBSCertificate'{
          version,              % v1 | v2 | v3
          serialNumber,         % integer()
          signature,            % #'AlgorithmIdentifier'{}
          issuer,               % {rdnSequence, [#AttributeTypeAndValue'{}]}
          validity,             % #'Validity'{}
          subject,              % {rdnSequence, [#AttributeTypeAndValue'{}]}
          subjectPublicKeyInfo, % #'SubjectPublicKeyInfo'{}
          issuerUniqueID,       % binary() | asn1_novalue
          subjectUniqueID,      % binary() | asn1_novalue
          extensions            % [#'Extension'{}]
         }.

#'AlgorithmIdentifier'{
          algorithm,  % oid()
          parameters  % asn1_der_encoded()
         }.
#'SignatureAlgorithm'{
          algorithm,  % id_signature_algorithm()
          parameters  % public_key_params()
         }.
```

id_signature_algorithm() = ?oid_name_as_erlang_atom for available oid names see table below. Ex: ?'id-dsa-with-sha1'

OID name
id-dsa-with-sha1
md2WithRSAEncryption
md5WithRSAEncryption
sha1WithRSAEncryption
ecdsa-with-SHA1

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'

OID name	Value type
id-at-name	special_string()
id-at-surname	special_string()
id-at-givenName	special_string()
id-at-initials	special_string()
id-at-generationQualifier	special_string()
id-at-commonName	special_string()
id-at-localityName	special_string()
id-at-stateOrProvinceName	special_string()
id-at-organizationName	special_string()
id-at-title	special_string()
id-at-dnQualifier	{printableString, string()}
id-at-countryName	{printableString, string()}
id-at-serialNumber	{printableString, string()}
id-at-pseudonym	special_string()

Table 1.2: Attribute oids

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

#'SubjectPublicKeyInfo'{
          algorithm,       % #AlgorithmIdentifier{}
          subjectPublicKey % binary()
         }.

#'SubjectPublicKeyInfoAlgorithm'{
          algorithm,  % id_public_key_algorithm()
          parameters  % public_key_params()
         }.
```

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

OID name
rsaEncryption
id-dsa
dhpublicnumber
ecdsa-with-SHA1
id-keyExchangeAlgorithm

Table 1.3: Public key algorithm oids

```
#'Extension'{
          extnID,    % id_extensions() | oid()
          critical,  % boolean()
          extnValue  % asn1_der_encoded()
         }.
```

id_extensions() = ?oid_name_as_erlang_atom for available oid names see tables. Ex: ?’id-ce-authorityKeyIdentifier’Standard Certificate Extensions [page 5], Private Internet Extensions [page 8], CRL Extensions [page 9] and CRL Entry Extensions [page 9].

1.3.3 Standard certificate extensions

OID name	Value type
id-ce-authorityKeyIdentifier	#’AuthorityKeyIdentifier’{}
id-ce-subjectKeyIdentifier	oid()
id-ce-keyUsage	[key_usasge()]
id-ce-privateKeyUsagePeriod	#’PrivateKeyUsagePeriod’{}
id-ce-certificatePolicies	#’PolicyInformation’{}
id-ce-policyMappings	#’PolicyMappings_SEQOF’{}
id-ce-subjectAltName	general_name()
id-ce-issuerAltName	general_name()
id-ce-subjectDirectoryAttributes	[#’Attribute’{}]
id-ce-basicConstraints	#’BasicConstraints’{}
id-ce-nameConstraints	#’NameConstraints’{}

continued ...

... continued

id-ce-policyConstraints	#'PolicyConstraints'{}
id-ce-extKeyUsage	[id_key_purpose()]
id-ce-cRLDistributionPoints	#'DistributionPoint'{}
id-ce-inhibitAnyPolicy	integer()
id-ce-freshestCRL	[#'DistributionPoint'{}]

Table 1.4: Standard Certificate Extensions

key_usasge() = 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'

OID name
id-kp-serverAuth
id-kp-clientAuth
id-kp-codeSigning
id-kp-emailProtection
id-kp-timeStamping
id-kp-OCSPSigning

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()
         }.

#'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

OID name	Value type
id-pe-authorityInfoAccess	[#'AccessDescription'{}]
id-pe-subjectInfoAccess	[#'AccessDescription'{}]

Table 1.6: Private Internet Extensions

```
#'AccessDescription'{
           accessMethod,    % oid()
           accessLocation   % general_name()
         }).
```

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

OID name	Value type
id-ce-authorityKeyIdentifier	#'AuthorityKeyIdentifier{}
id-ce-issuerAltName	{rdnSequence, [#AttributeTypeAndValue'{}]}
id-ce-cRLNumber	integer()
id-ce-deltaCRLIndicator	integer()
id-ce-issuingDistributionPoint	#'IssuingDistributionPoint'{}
id-ce-freshestCRL	[#'Distributionpoint'{}]

Table 1.7: CRL Extensions

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

CRL Entry Extensions

OID name	Value type
id-ce-cRLReason	crl_reason()
id-ce-holdInstructionCode	oid()
id-ce-invalidityDate	general_time()
id-ce-certificateIssuer	general_name()

Table 1.8: CRL Entry Extensions

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

public_key Reference Manual

Short Summaries

- Erlang Module public_key [page 12] – API module for public key infrastructure.

public_key

No functions are exported.

public_key

Erlang Module

This module provides functions to handle public key infrastructure from RFC 3280 (X.509 certificates) and some parts of the PKCS-standard. Currently this application is only meant to be used by the new ssl implementation. The API is not yet mature enough to be released for common use but will be documented an made publicly available in the future.

List of Tables