Search packages or symbols

x509

package standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 40 | Imported by: 65,753

Details ✓ Valid go.mod file ✓ Redistributable license ✓ Tagged version ✓ Stable version

Learn more

Repository cs.opensource.google/go/go

Links Report a Vulnerability

Documentation

Documentation

Rendered for linux/amd64

Overview

Package x509 implements a subset of the X.509 standard.

It allows parsing and generating certificates, certificate signing requests, certificate revocation lists, and encoded public and private keys. It provides a certificate verifier, complete with a chain builder.

The package targets the X.509 technical profile defined by the IETF (RFC 2459/3280/5280), and as further restricted by the CA/Browser Forum Baseline Requirements. There is minimal support for features outside of these profiles, as the primary goal of the package is to provide compatibility with the publicly trusted TLS certificate ecosystem and its policies and constraints.

On macOS and Windows, certificate verification is handled by system APIs, but the package aims to apply consistent validation rules across operating systems.

Index

Examples

Constants

This section is empty.

Variables

View Source

```
var ErrUnsupportedAlgorithm = errors.New("x509: cannot verify signature: algorithm unim
```

ErrUnsupportedAlgorithm results from attempting to perform an operation that involves algorithms that are not currently implemented.

View Source

```
var IncorrectPasswordError = errors.New("x509: decryption password incorrect")
```

IncorrectPasswordError is returned when an incorrect password is detected.

Functions

func CreateCertificate

```
func CreateCertificate(rand io.Reader, template, parent *Certificate, pub, priv any)
([]byte, error)
```

CreateCertificate creates a new X.509 v3 certificate based on a template. The following members of template are currently used:

- AuthorityKeyId
- BasicConstraintsValid
- CRLDistributionPoints
- DNSNames
- EmailAddresses
- ExcludedDNSDomains
- ExcludedEmailAddresses
- ExcludedIPRanges
- ExcludedURIDomains
- ExtKeyUsage
- ExtraExtensions
- IPAddresses
- IsCA
- IssuingCertificateURL
- KeyUsage
- MaxPathLen
- MaxPathLenZero
- NotAfter
- NotBefore
- OCSPServer
- PermittedDNSDomains
- PermittedDNSDomainsCritical
- PermittedEmailAddresses
- PermittedIPRanges
- PermittedURIDomains
- PolicyIdentifiers
- SerialNumber
- SignatureAlgorithm
- Subject
- SubjectKeyId
- URIs
- UnknownExtKeyUsage

The certificate is signed by parent. If parent is equal to template then the certificate is self-signed. The parameter pub is the public key of the certificate to be generated and priv is the private key of the signer.

The returned slice is the certificate in DER encoding.

The currently supported key types are *rsa.PublicKey, *ecdsa.PublicKey and ed25519.PublicKey. pub must be a supported key type, and priv must be a crypto.Signer with a supported public key.

The AuthorityKeyId will be taken from the SubjectKeyId of parent, if any, unless the resulting certificate is self-signed. Otherwise the value from template will be used.

If SubjectKeyId from template is empty and the template is a CA, SubjectKeyId will be generated from the hash of the public key.

func CreateCertificateRequest

added in go1.3

```
func CreateCertificateRequest(rand io.Reader, template *CertificateRequest, priv any)
(csr []byte, err error)
```

CreateCertificateRequest creates a new certificate request based on a template. The following members of template are used:

- SignatureAlgorithm
- Subject
- DNSNames
- EmailAddresses
- IPAddresses
- URIs
- ExtraExtensions
- Attributes (deprecated)

priv is the private key to sign the CSR with, and the corresponding public key will be included in the CSR. It must implement crypto.Signer and its Public() method must return a *rsa.PublicKey or a *ecdsa.PublicKey or a ed25519.PublicKey. (A *rsa.PrivateKey, *ecdsa.PrivateKey or ed25519.PrivateKey satisfies this.)

The returned slice is the certificate request in DER encoding.

func CreateRevocationList

added in go1.15

```
func CreateRevocationList(rand io.Reader, template *RevocationList, issuer *Certificat
e, priv crypto.Signer) ([]byte, error)
```

CreateRevocationList creates a new X.509 v2 Certificate Revocation List, according to RFC 5280, based on template.

The CRL is signed by priv which should be the private key associated with the public key in the issuer certificate.

The issuer may not be nil, and the crlSign bit must be set in KeyUsage in order to use it as a CRL issuer.

The issuer distinguished name CRL field and authority key identifier extension are populated using the issuer certificate. issuer must have SubjectKeyId set.

func DecryptPEMBlock DEPRECATED Show

added in go1.1

func EncryptPEMBlock DEPRECATED Show

added in go1.1

func IsEncryptedPEMBlock DEPRECATED Show

added in go1.1

func MarshalECPrivateKey

added in go1.2

```
func MarshalECPrivateKey(key *ecdsa.PrivateKey) ([]byte, error)
```

MarshalECPrivateKey converts an EC private key to SEC 1, ASN.1 DER form.

This kind of key is commonly encoded in PEM blocks of type "EC PRIVATE KEY". For a more flexible key format which is not EC specific, use MarshalPKCS8PrivateKey.

func MarshalPKCS1PrivateKey

```
func MarshalPKCS1PrivateKey(key *rsa.PrivateKey) []byte
```

MarshalPKCS1PrivateKey converts an RSA private key to PKCS #1, ASN.1 DER form.

This kind of key is commonly encoded in PEM blocks of type "RSA PRIVATE KEY". For a more flexible key format which is not RSA specific, use MarshalPKCS8PrivateKey.

func MarshalPKCS1PublicKey

added in go1.10

```
func MarshalPKCS1PublicKey(key *rsa.PublicKey) []byte
```

MarshalPKCS1PublicKey converts an RSA public key to PKCS #1, ASN.1 DER form.

This kind of key is commonly encoded in PEM blocks of type "RSA PUBLIC KEY".

func MarshalPKCS8PrivateKey

added in go1.10

```
func MarshalPKCS8PrivateKey(key any) ([]byte, error)
```

MarshalPKCS8PrivateKey converts a private key to PKCS #8, ASN.1 DER form.

The following key types are currently supported: *rsa.PrivateKey, *ecdsa.PrivateKey, ed25519.PrivateKey (not a pointer), and *ecdh.PrivateKey. Unsupported key types result in an error.

This kind of key is commonly encoded in PEM blocks of type "PRIVATE KEY".

func MarshalPKIXPublicKey

```
func MarshalPKIXPublicKey(pub any) ([]byte, error)
```

MarshalPKIXPublicKey converts a public key to PKIX, ASN.1 DER form. The encoded public key is a SubjectPublicKeyInfo structure (see RFC 5280, Section 4.1).

The following key types are currently supported: *rsa.PublicKey, *ecdsa.PublicKey, ed25519.PublicKey (not a pointer), and *ecdh.PublicKey. Unsupported key types result in an error.

This kind of key is commonly encoded in PEM blocks of type "PUBLIC KEY".

func ParseCRL DEPRECATED Show

func ParseDERCRL DEPRECATED Show

func ParseECPrivateKey

added in go1.1

```
func ParseECPrivateKey(der []byte) (*ecdsa.PrivateKey, error)
```

ParseECPrivateKey parses an EC private key in SEC 1, ASN.1 DER form.

This kind of key is commonly encoded in PEM blocks of type "EC PRIVATE KEY".

func ParsePKCS1PrivateKey

```
func ParsePKCS1PrivateKey(der []byte) (*rsa.PrivateKey, error)
```

ParsePKCS1PrivateKey parses an RSA private key in PKCS #1, ASN.1 DER form.

This kind of key is commonly encoded in PEM blocks of type "RSA PRIVATE KEY".

func ParsePKCS1PublicKey

added in go1.10

```
func ParsePKCS1PublicKey(der []byte) (*rsa.PublicKey, error)
```

ParsePKCS1PublicKey parses an RSA public key in PKCS #1, ASN.1 DER form.

This kind of key is commonly encoded in PEM blocks of type "RSA PUBLIC KEY".

func ParsePKCS8PrivateKey

```
func ParsePKCS8PrivateKey(der []byte) (key any, err error)
```

ParsePKCS8PrivateKey parses an unencrypted private key in PKCS #8, ASN.1 DER form.

It returns a *rsa.PrivateKey, a *ecdsa.PrivateKey, a ed25519.PrivateKey (not a pointer), or a *ecdh.PublicKey (for X25519). More types might be supported in the future.

This kind of key is commonly encoded in PEM blocks of type "PRIVATE KEY".

func ParsePKIXPublicKey

```
func ParsePKIXPublicKey(derBytes []byte) (pub any, err error)
```

ParsePKIXPublicKey parses a public key in PKIX, ASN.1 DER form. The encoded public key is a SubjectPublicKeyInfo structure (see RFC 5280, Section 4.1).

It returns a *rsa.PublicKey, *dsa.PublicKey, *ecdsa.PublicKey, ed25519.PublicKey (not a pointer), or *ecdh.PublicKey (for X25519). More types might be supported in the future.

This kind of key is commonly encoded in PEM blocks of type "PUBLIC KEY".

▶ Example

func SetFallbackRoots

added in go1.20

```
func SetFallbackRoots(roots *CertPool)
```

SetFallbackRoots sets the roots to use during certificate verification, if no custom roots are specified and a platform verifier or a system certificate pool is not available (for instance in a container which does not have a root certificate bundle). SetFallbackRoots will panic if roots is nil.

SetFallbackRoots may only be called once, if called multiple times it will panic.

The fallback behavior can be forced on all platforms, even when there is a system certificate pool, by setting GODEBUG=x509usefallbackroots=1 (note that on Windows and macOS this will disable usage of the platform verification APIs and cause the pure Go verifier to be used). Setting x509usefallbackroots=1 without calling SetFallbackRoots has no effect.

Types

type CertPool

```
type CertPool struct {
	// contains filtered or unexported fields
}
```

CertPool is a set of certificates.

func NewCertPool

```
func NewCertPool() *CertPool
```

NewCertPool returns a new, empty CertPool.

func SystemCertPool

added in go1.7

```
func SystemCertPool() (*CertPool, error)
```

SystemCertPool returns a copy of the system cert pool.

On Unix systems other than macOS the environment variables SSL_CERT_FILE and SSL_CERT_DIR can be used to override the system default locations for the SSL certificate file and SSL certificate files directory, respectively. The latter can be a colon-separated list.

Any mutations to the returned pool are not written to disk and do not affect any other pool returned by SystemCertPool.

New changes in the system cert pool might not be reflected in subsequent calls.

func (*CertPool) AddCert

```
func (s *CertPool) AddCert(cert *Certificate)
```

AddCert adds a certificate to a pool.

func (*CertPool) AppendCertsFromPEM

```
func (s *CertPool) AppendCertsFromPEM(pemCerts []byte) (ok bool)
```

AppendCertsFromPEM attempts to parse a series of PEM encoded certificates. It appends any certificates found to s and reports whether any certificates were successfully parsed.

On many Linux systems, /etc/ssl/cert.pem will contain the system wide set of root CAs in a format suitable for this function.

func (*CertPool) Clone

added in go1.19

```
func (s *CertPool) Clone() *CertPool
```

Clone returns a copy of s.

func (*CertPool) Equal

added in go1.19

```
func (s *CertPool) Equal(other *CertPool) bool
```

Equal reports whether s and other are equal.

func (*CertPool) Subjects DEPRECATED Show

type Certificate

```
type Certificate struct {
    Raw                     []byte // Complete ASN.1 DER content (certificate, signatur
    RawTBSCertificate       []byte // Certificate part of raw ASN.1 DER content.
    RawSubjectPublicKeyInfo []byte // DER encoded SubjectPublicKeyInfo.
    RawSubject              []byte // DER encoded Subject
    RawIssuer               []byte // DER encoded Issuer

    Signature          []byte
    SignatureAlgorithm SignatureAlgorithm

    PublicKeyAlgorithm PublicKeyAlgorithm
    PublicKey          any

    Version             int
    SerialNumber        *big.Int
    Issuer              pkix.Name
    Subject             pkix.Name
    NotBefore, NotAfter time.Time // Validity bounds.
    KeyUsage            KeyUsage
    // Extensions contains raw X.509 extensions. When parsing certificates,
    // this can be used to extract non-critical extensions that are not
    // parsed by this package. When marshaling certificates, the Extensions
    // field is ignored, see ExtraExtensions.
    Extensions []pkix.Extension

    // ExtraExtensions contains extensions to be copied, raw, into any
    // marshaled certificates. Values override any extensions that would
    // otherwise be produced based on the other fields. The ExtraExtensions
    // field is not populated when parsing certificates, see Extensions.
    ExtraExtensions []pkix.Extension

    // UnhandledCriticalExtensions contains a list of extension IDs that
    // were not (fully) processed when parsing. Verify will fail if this
    // slice is non-empty, unless verification is delegated to an OS
    // library which understands all the critical extensions.
    //
    // Users can access these extensions using Extensions and can remove
    // elements from this slice if they believe that they have been
    // handled.
    UnhandledCriticalExtensions []asn1.ObjectIdentifier

    ExtKeyUsage        []ExtKeyUsage           // Sequence of extended key usages.
    UnknownExtKeyUsage []asn1.ObjectIdentifier // Encountered extended key usages unknow

    // BasicConstraintsValid indicates whether IsCA, MaxPathLen,
    // and MaxPathLenZero are valid.
    BasicConstraintsValid bool
    IsCA                  bool

    // MaxPathLen and MaxPathLenZero indicate the presence and
    // value of the BasicConstraints' "pathLenConstraint".
    //
    // When parsing a certificate, a positive non-zero MaxPathLen
    // means that the field was specified, -1 means it was unset,
    // and MaxPathLenZero being true mean that the field was
    // explicitly set to zero. The case of MaxPathLen==0 with MaxPathLenZero==false
    // should be treated equivalent to -1 (unset).
    //
    // When generating a certificate, an unset pathLenConstraint
    // can be requested with either MaxPathLen == -1 or using the
    // zero value for both MaxPathLen and MaxPathLenZero.
    MaxPathLen int
    // MaxPathLenZero indicates that BasicConstraintsValid==true
    // and MaxPathLen==0 should be interpreted as an actual
    // maximum path length of zero. Otherwise, that combination is
    // interpreted as MaxPathLen not being set.
    MaxPathLenZero bool

    SubjectKeyId   []byte
    AuthorityKeyId []byte
    // RFC 5280, 4.2.2.1 (Authority Information Access)
    OCSPServer            []string
    IssuingCertificateURL []string

    // Subject Alternate Name values. (Note that these values may not be valid
    // if invalid values were contained within a parsed certificate. For
    // example, an element of DNSNames may not be a valid DNS domain name.)
    DNSNames       []string
    EmailAddresses []string
    IPAddresses    []net.IP
    URIs           []*url.URL

    // Name constraints
    PermittedDNSDomainsCritical bool // if true then the name constraints are marked cr
    PermittedDNSDomains         []string
    ExcludedDNSDomains          []string
    PermittedIPRanges           []*net.IPNet
    ExcludedIPRanges            []*net.IPNet
    PermittedEmailAddresses     []string
    ExcludedEmailAddresses      []string
    PermittedURIDomains         []string
    ExcludedURIDomains          []string

    // CRL Distribution Points
    CRLDistributionPoints []string

    PolicyIdentifiers []asn1.ObjectIdentifier
}
```

A Certificate represents an X.509 certificate.

func ParseCertificate

```
func ParseCertificate(der []byte) (*Certificate, error)
```

ParseCertificate parses a single certificate from the given ASN.1 DER data.

func ParseCertificates

```
func ParseCertificates(der []byte) ([]*Certificate, error)
```

ParseCertificates parses one or more certificates from the given ASN.1 DER data. The certificates must be concatenated with no intermediate padding.

func (*Certificate) CheckCRLSignature DEPRECATED Show

func (*Certificate) CheckSignature

```
func (c *Certificate) CheckSignature(algo SignatureAlgorithm, signed, signature []byt
e) error
```

CheckSignature verifies that signature is a valid signature over signed from c's public key.

This is a low-level API that performs no validity checks on the certificate.

MD5WithRSA signatures are rejected, while SHA1WithRSA and ECDSAWithSHA1 signatures are currently accepted.

func (*Certificate) CheckSignatureFrom

```
func (c *Certificate) CheckSignatureFrom(parent *Certificate) error
```

CheckSignatureFrom verifies that the signature on c is a valid signature from parent.

This is a low-level API that performs very limited checks, and not a full path verifier. Most users should use Certificate.Verify instead.

func (*Certificate) CreateCRL DEPRECATED Show

func (*Certificate) Equal

```
func (c *Certificate) Equal(other *Certificate) bool
```

func (*Certificate) Verify

```
func (c *Certificate) Verify(opts VerifyOptions) (chains [][]*Certificate, err error)
```

Verify attempts to verify c by building one or more chains from c to a certificate in opts.Roots, using certificates in opts.Intermediates if needed. If successful, it returns one or more chains where the first element of the chain is c and the last element is from opts.Roots.

If opts.Roots is nil, the platform verifier might be used, and verification details might differ from what is described below. If system roots are unavailable the returned error will be of type SystemRootsError.

Name constraints in the intermediates will be applied to all names claimed in the chain, not just opts.DNSName. Thus it is invalid for a leaf to claim example.com if an intermediate doesn't permit it, even if example.com is not the name being validated. Note that DirectoryName constraints are not supported.

Name constraint validation follows the rules from RFC 5280, with the addition that DNS name constraints may use the leading period format defined for emails and URIs. When a constraint has a leading period it indicates that at least one additional label must be prepended to the constrained name to be considered valid.

Extended Key Usage values are enforced nested down a chain, so an intermediate or root that enumerates EKUs prevents a leaf from asserting an EKU not in that list. (While this is not specified, it is common practice in order to limit the types of certificates a CA can issue.)

Certificates that use SHA1WithRSA and ECDSAWithSHA1 signatures are not supported, and will not be used to build chains.

Certificates other than c in the returned chains should not be modified.

WARNING: this function doesn't do any revocation checking.

▶ Example

func (*Certificate) VerifyHostname

```
func (c *Certificate) VerifyHostname(h string) error
```

VerifyHostname returns nil if c is a valid certificate for the named host. Otherwise it returns an error describing the mismatch.

IP addresses can be optionally enclosed in square brackets and are checked against the IPAddresses field. Other names are checked case insensitively against the DNSNames field. If the names are valid hostnames, the certificate fields can have a wildcard as the left-most label.

Note that the legacy Common Name field is ignored.

type CertificateInvalidError

```
type CertificateInvalidError struct {
    Cert   *Certificate
    Reason InvalidReason
    Detail string
}
```

CertificateInvalidError results when an odd error occurs. Users of this library probably want to handle all these errors uniformly.

func (CertificateInvalidError) Error

```
func (e CertificateInvalidError) Error() string
```

type CertificateRequest

added in go1.3

```
type CertificateRequest struct {
    Raw                      []byte // Complete ASN.1 DER content (CSR, signature algor
    RawTBSCertificateRequest []byte // Certificate request info part of raw ASN.1 DER co
    RawSubjectPublicKeyInfo  []byte // DER encoded SubjectPublicKeyInfo.
    RawSubject               []byte // DER encoded Subject.

    Version            int
    Signature          []byte
    SignatureAlgorithm SignatureAlgorithm

    PublicKeyAlgorithm PublicKeyAlgorithm
    PublicKey          any
    Subject pkix.Name

    // Attributes contains the CSR attributes that can parse as
    // pkix.AttributeTypeAndValueSET.
    //
    // Deprecated: Use Extensions and ExtraExtensions instead for parsing and
    // generating the requestedExtensions attribute.
    Attributes []pkix.AttributeTypeAndValueSET

    // Extensions contains all requested extensions, in raw form. When parsing
    // CSRs, this can be used to extract extensions that are not parsed by this
    // package.
    Extensions []pkix.Extension

    // ExtraExtensions contains extensions to be copied, raw, into any CSR
    // marshaled by CreateCertificateRequest. Values override any extensions
    // that would otherwise be produced based on the other fields but are
    // overridden by any extensions specified in Attributes.
    //
    // The ExtraExtensions field is not populated by ParseCertificateRequest,
    // see Extensions instead.
    ExtraExtensions []pkix.Extension

    // Subject Alternate Name values.
    DNSNames       []string
    EmailAddresses []string
    IPAddresses    []net.IP
    URIs           []*url.URL
}
```

CertificateRequest represents a PKCS #10, certificate signature request.

func ParseCertificateRequest

added in go1.3

```
func ParseCertificateRequest(asn1Data []byte) (*CertificateRequest, error)
```

ParseCertificateRequest parses a single certificate request from the given ASN.1 DER data.

func (*CertificateRequest) CheckSignature

added in go1.5

```
func (c *CertificateRequest) CheckSignature() error
```

CheckSignature reports whether the signature on c is valid.

type ConstraintViolationError

```
type ConstraintViolationError struct{}
```

ConstraintViolationError results when a requested usage is not permitted by a certificate. For example: checking a signature when the public key isn't a certificate signing key.

func (ConstraintViolationError) Error

```
func (ConstraintViolationError) Error() string
```

type ExtKeyUsage

```
type ExtKeyUsage int
```

ExtKeyUsage represents an extended set of actions that are valid for a given key. Each of the ExtKeyUsage* constants define a unique action.

```
const (
    ExtKeyUsageAny ExtKeyUsage = iota
    ExtKeyUsageServerAuth
    ExtKeyUsageClientAuth
    ExtKeyUsageCodeSigning
    ExtKeyUsageEmailProtection
    ExtKeyUsageIPSECEndSystem
    ExtKeyUsageIPSECTunnel
    ExtKeyUsageIPSECUser
    ExtKeyUsageTimeStamping
    ExtKeyUsageOCSPSigning
    ExtKeyUsageMicrosoftServerGatedCrypto
    ExtKeyUsageNetscapeServerGatedCrypto
    ExtKeyUsageMicrosoftCommercialCodeSigning
    ExtKeyUsageMicrosoftKernelCodeSigning
)
```

type HostnameError

```
type HostnameError struct {
    Certificate *Certificate
    Host        string
}
```

HostnameError results when the set of authorized names doesn't match the requested name.

func (HostnameError) Error

```
func (h HostnameError) Error() string
```

type InsecureAlgorithmError

added in go1.6

```
type InsecureAlgorithmError SignatureAlgorithm
```

An InsecureAlgorithmError indicates that the SignatureAlgorithm used to generate the signature is not secure, and the signature has been rejected.

To temporarily restore support for SHA-1 signatures, include the value "x509sha1=1" in the GODEBUG environment variable. Note that this option will be removed in a future release.

func (InsecureAlgorithmError) Error

added in go1.6

```
func (e InsecureAlgorithmError) Error() string
```

type InvalidReason

```
type InvalidReason int
```

```
const (
	// NotAuthorizedToSign results when a certificate is signed by another
	// which isn't marked as a CA certificate.
	NotAuthorizedToSign InvalidReason = iota
	// Expired results when a certificate has expired, based on the time
	// given in the VerifyOptions.
	Expired
	// CANotAuthorizedForThisName results when an intermediate or root
	// certificate has a name constraint which doesn't permit a DNS or
	// other name (including IP address) in the leaf certificate.
	CANotAuthorizedForThisName
	// TooManyIntermediates results when a path length constraint is
	// violated.
	TooManyIntermediates
	// IncompatibleUsage results when the certificate's key usage indicates
	// that it may only be used for a different purpose.
	IncompatibleUsage
	// NameMismatch results when the subject name of a parent certificate
	// does not match the issuer name in the child.
	NameMismatch
	// NameConstraintsWithoutSANs is a legacy error and is no longer returned.
	NameConstraintsWithoutSANs
	// UnconstrainedName results when a CA certificate contains permitted
	// name constraints, but leaf certificate contains a name of an
	// unsupported or unconstrained type.
	UnconstrainedName
	// TooManyConstraints results when the number of comparison operations
	// needed to check a certificate exceeds the limit set by
	// VerifyOptions.MaxConstraintComparisions. This limit exists to
	// prevent pathological certificates can consuming excessive amounts of
	// CPU time to verify.
	TooManyConstraints
	// CANotAuthorizedForExtKeyUsage results when an intermediate or root
	// certificate does not permit a requested extended key usage.
	CANotAuthorizedForExtKeyUsage
)
```

type KeyUsage

```
type KeyUsage int
```

KeyUsage represents the set of actions that are valid for a given key. It's a bitmap of the KeyUsage* constants.

```
const (
    KeyUsageDigitalSignature KeyUsage = 1 << iota
    KeyUsageContentCommitment
    KeyUsageKeyEncipherment
    KeyUsageDataEncipherment
    KeyUsageKeyAgreement
    KeyUsageCertSign
    KeyUsageCRLSign
    KeyUsageEncipherOnly
    KeyUsageDecipherOnly
)
```

type PEMCipher

added in go1.1

```
type PEMCipher int
```

```
const (
    PEMCipherDES PEMCipher
    PEMCipher3DES
    PEMCipherAES128
    PEMCipherAES192
    PEMCipherAES256
)
```

Possible values for the EncryptPEMBlock encryption algorithm.

type PublicKeyAlgorithm

```
type PublicKeyAlgorithm int
```

```
const (
    UnknownPublicKeyAlgorithm PublicKeyAlgorithm = iota
    RSA
    DSA // Only supported for parsing.
    ECDSA
    Ed25519
)
```

func (PublicKeyAlgorithm) String

added in go1.10

```
func (algo PublicKeyAlgorithm) String() string
```

type RevocationList

added in go1.15

```
type RevocationList struct {
    // Raw contains the complete ASN.1 DER content of the CRL (tbsCertList,
    // signatureAlgorithm, and signatureValue.)
    Raw []byte
    // RawTBSRevocationList contains just the tbsCertList portion of the ASN.1
    // DER.
    RawTBSRevocationList []byte
    // RawIssuer contains the DER encoded Issuer.
    RawIssuer []byte

    // Issuer contains the DN of the issuing certificate.
    Issuer pkix.Name
    // AuthorityKeyId is used to identify the public key associated with the
    // issuing certificate. It is populated from the authorityKeyIdentifier
    // extension when parsing a CRL. It is ignored when creating a CRL; the
    // extension is populated from the issuing certificate itself.
    AuthorityKeyId []byte

    Signature []byte
    // SignatureAlgorithm is used to determine the signature algorithm to be
    // used when signing the CRL. If 0 the default algorithm for the signing
    // key will be used.
    SignatureAlgorithm SignatureAlgorithm

    // RevokedCertificates is used to populate the revokedCertificates
    // sequence in the CRL, it may be empty. RevokedCertificates may be nil,
    // in which case an empty CRL will be created.
    RevokedCertificates []pkix.RevokedCertificate

    // Number is used to populate the X.509 v2 cRLNumber extension in the CRL,
    // which should be a monotonically increasing sequence number for a given
    // CRL scope and CRL issuer. It is also populated from the cRLNumber
    // extension when parsing a CRL.
    Number *big.Int

    // ThisUpdate is used to populate the thisUpdate field in the CRL, which
    // indicates the issuance date of the CRL.
    ThisUpdate time.Time
    // NextUpdate is used to populate the nextUpdate field in the CRL, which
    // indicates the date by which the next CRL will be issued. NextUpdate
    // must be greater than ThisUpdate.
    NextUpdate time.Time

    // Extensions contains raw X.509 extensions. When creating a CRL,
    // the Extensions field is ignored, see ExtraExtensions.
    Extensions []pkix.Extension

    // ExtraExtensions contains any additional extensions to add directly to
    // the CRL.
    ExtraExtensions []pkix.Extension
}
```

RevocationList contains the fields used to create an X.509 v2 Certificate Revocation list with CreateRevocationList.

func ParseRevocationList

added in go1.19

```
func ParseRevocationList(der []byte) (*RevocationList, error)
```

ParseRevocationList parses a X509 v2 Certificate Revocation List from the given ASN.1 DER data.

func (*RevocationList) CheckSignatureFrom

added in go1.19

```
func (rl *RevocationList) CheckSignatureFrom(parent *Certificate) error
```

CheckSignatureFrom verifies that the signature on rl is a valid signature from issuer.

type SignatureAlgorithm

```
type SignatureAlgorithm int
```

```
const (
    UnknownSignatureAlgorithm SignatureAlgorithm = iota

    MD2WithRSA  // Unsupported.
    MD5WithRSA  // Only supported for signing, not verification.
    SHA1WithRSA // Only supported for signing, and verification of CRLs, CSRs, and OCSP
    SHA256WithRSA
    SHA384WithRSA
    SHA512WithRSA
    DSAWithSHA1   // Unsupported.
    DSAWithSHA256 // Unsupported.
    ECDSAWithSHA1 // Only supported for signing, and verification of CRLs, CSRs, and OC
    ECDSAWithSHA256
    ECDSAWithSHA384
    ECDSAWithSHA512
    SHA256WithRSAPSS
    SHA384WithRSAPSS
    SHA512WithRSAPSS
    PureEd25519
)
```

func (SignatureAlgorithm) String

added in go1.6

```
func (algo SignatureAlgorithm) String() string
```

type SystemRootsError

added in go1.1

```
type SystemRootsError struct {
    Err error
}
```

SystemRootsError results when we fail to load the system root certificates.

func (SystemRootsError) Error

added in go1.1

```
func (se SystemRootsError) Error() string
```

func (SystemRootsError) Unwrap

added in go1.16

```
func (se SystemRootsError) Unwrap() error
```

type UnhandledCriticalExtension

```
type UnhandledCriticalExtension struct{}
```

func (UnhandledCriticalExtension) Error

```
func (h UnhandledCriticalExtension) Error() string
```

type UnknownAuthorityError

```
type UnknownAuthorityError struct {
    Cert *Certificate
    // contains filtered or unexported fields
}
```

UnknownAuthorityError results when the certificate issuer is unknown

func (UnknownAuthorityError) Error

```
func (e UnknownAuthorityError) Error() string
```

type VerifyOptions

```
type VerifyOptions struct {
    // DNSName, if set, is checked against the leaf certificate with
    // Certificate.VerifyHostname or the platform verifier.
    DNSName string

    // Intermediates is an optional pool of certificates that are not trust
    // anchors, but can be used to form a chain from the leaf certificate to a
    // root certificate.
    Intermediates *CertPool
    // Roots is the set of trusted root certificates the leaf certificate needs
    // to chain up to. If nil, the system roots or the platform verifier are used.
    Roots *CertPool

    // CurrentTime is used to check the validity of all certificates in the
    // chain. If zero, the current time is used.
    CurrentTime time.Time
    // KeyUsages specifies which Extended Key Usage values are acceptable. A
    // chain is accepted if it allows any of the listed values. An empty list
    // means ExtKeyUsageServerAuth. To accept any key usage, include ExtKeyUsageAny.
    KeyUsages []ExtKeyUsage

    // MaxConstraintComparisions is the maximum number of comparisons to
    // perform when checking a given certificate's name constraints. If
    // zero, a sensible default is used. This limit prevents pathological
    // certificates from consuming excessive amounts of CPU time when
    // validating. It does not apply to the platform verifier.
    MaxConstraintComparisions int
}
```

VerifyOptions contains parameters for Certificate.Verify.

Source Files

View all

cert_pool.go	pkcs1.go	root_unix.go
notboring.go	pkcs8.go	sec1.go
parser.go	root.go	verify.go
pem_decrypt.go	root_linux.go	x509.go

Directories

Expand all

pkix

Package pkix contains shared, low level structures used for ASN.1 parsing and serialization of X.509 certificates, CRL and OCSP.

internal