/\*\*

\* The `crypto` module provides cryptographic functionality that includes a set of

\* wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign, and verify functions.

\*

\* ```js

\* const { createHmac } = await import('crypto');

\*

\* const secret = 'abcdefg';

\* const hash = createHmac('sha256', secret)

\* .update('I love cupcakes')

\* .digest('hex');

\* console.log(hash);

\* // Prints:

\* // c0fa1bc00531bd78ef38c628449c5102aeabd49b5dc3a2a516ea6ea959d6658e

\* ```

\* @see [source](https://github.com/nodejs/node/blob/v17.0.0/lib/crypto.js)

\*/

declare module 'crypto' {

import \* as stream from 'node:stream';

import { PeerCertificate } from 'node:tls';

interface Certificate {

/\*\*

\* @deprecated

\* @param spkac

\* @returns The challenge component of the `spkac` data structure,

\* which includes a public key and a challenge.

\*/

exportChallenge(spkac: BinaryLike): Buffer;

/\*\*

\* @deprecated

\* @param spkac

\* @param encoding The encoding of the spkac string.

\* @returns The public key component of the `spkac` data structure,

\* which includes a public key and a challenge.

\*/

exportPublicKey(spkac: BinaryLike, encoding?: string): Buffer;

/\*\*

\* @deprecated

\* @param spkac

\* @returns `true` if the given `spkac` data structure is valid,

\* `false` otherwise.

\*/

verifySpkac(spkac: NodeJS.ArrayBufferView): boolean;

}

const Certificate: Certificate & {

/\*\* @deprecated since v14.9.0 - Use static methods of `crypto.Certificate` instead. \*/

new (): Certificate;

/\*\* @deprecated since v14.9.0 - Use static methods of `crypto.Certificate` instead. \*/

(): Certificate;

/\*\*

\* @param spkac

\* @returns The challenge component of the `spkac` data structure,

\* which includes a public key and a challenge.

\*/

exportChallenge(spkac: BinaryLike): Buffer;

/\*\*

\* @param spkac

\* @param encoding The encoding of the spkac string.

\* @returns The public key component of the `spkac` data structure,

\* which includes a public key and a challenge.

\*/

exportPublicKey(spkac: BinaryLike, encoding?: string): Buffer;

/\*\*

\* @param spkac

\* @returns `true` if the given `spkac` data structure is valid,

\* `false` otherwise.

\*/

verifySpkac(spkac: NodeJS.ArrayBufferView): boolean;

};

namespace constants {

// https://nodejs.org/dist/latest-v10.x/docs/api/crypto.html#crypto\_crypto\_constants

const OPENSSL\_VERSION\_NUMBER: number;

/\*\* Applies multiple bug workarounds within OpenSSL. See https://www.openssl.org/docs/man1.0.2/ssl/SSL\_CTX\_set\_options.html for detail. \*/

const SSL\_OP\_ALL: number;

/\*\* Allows legacy insecure renegotiation between OpenSSL and unpatched clients or servers. See https://www.openssl.org/docs/man1.0.2/ssl/SSL\_CTX\_set\_options.html. \*/

const SSL\_OP\_ALLOW\_UNSAFE\_LEGACY\_RENEGOTIATION: number;

/\*\* Attempts to use the server's preferences instead of the client's when selecting a cipher. See https://www.openssl.org/docs/man1.0.2/ssl/SSL\_CTX\_set\_options.html. \*/

const SSL\_OP\_CIPHER\_SERVER\_PREFERENCE: number;

/\*\* Instructs OpenSSL to use Cisco's "speshul" version of DTLS\_BAD\_VER. \*/

const SSL\_OP\_CISCO\_ANYCONNECT: number;

/\*\* Instructs OpenSSL to turn on cookie exchange. \*/

const SSL\_OP\_COOKIE\_EXCHANGE: number;

/\*\* Instructs OpenSSL to add server-hello extension from an early version of the cryptopro draft. \*/

const SSL\_OP\_CRYPTOPRO\_TLSEXT\_BUG: number;

/\*\* Instructs OpenSSL to disable a SSL 3.0/TLS 1.0 vulnerability workaround added in OpenSSL 0.9.6d. \*/

const SSL\_OP\_DONT\_INSERT\_EMPTY\_FRAGMENTS: number;

/\*\* Instructs OpenSSL to always use the tmp\_rsa key when performing RSA operations. \*/

const SSL\_OP\_EPHEMERAL\_RSA: number;

/\*\* Allows initial connection to servers that do not support RI. \*/

const SSL\_OP\_LEGACY\_SERVER\_CONNECT: number;

const SSL\_OP\_MICROSOFT\_BIG\_SSLV3\_BUFFER: number;

const SSL\_OP\_MICROSOFT\_SESS\_ID\_BUG: number;

/\*\* Instructs OpenSSL to disable the workaround for a man-in-the-middle protocol-version vulnerability in the SSL 2.0 server implementation. \*/

const SSL\_OP\_MSIE\_SSLV2\_RSA\_PADDING: number;

const SSL\_OP\_NETSCAPE\_CA\_DN\_BUG: number;

const SSL\_OP\_NETSCAPE\_CHALLENGE\_BUG: number;

const SSL\_OP\_NETSCAPE\_DEMO\_CIPHER\_CHANGE\_BUG: number;

const SSL\_OP\_NETSCAPE\_REUSE\_CIPHER\_CHANGE\_BUG: number;

/\*\* Instructs OpenSSL to disable support for SSL/TLS compression. \*/

const SSL\_OP\_NO\_COMPRESSION: number;

const SSL\_OP\_NO\_QUERY\_MTU: number;

/\*\* Instructs OpenSSL to always start a new session when performing renegotiation. \*/

const SSL\_OP\_NO\_SESSION\_RESUMPTION\_ON\_RENEGOTIATION: number;

const SSL\_OP\_NO\_SSLv2: number;

const SSL\_OP\_NO\_SSLv3: number;

const SSL\_OP\_NO\_TICKET: number;

const SSL\_OP\_NO\_TLSv1: number;

const SSL\_OP\_NO\_TLSv1\_1: number;

const SSL\_OP\_NO\_TLSv1\_2: number;

const SSL\_OP\_PKCS1\_CHECK\_1: number;

const SSL\_OP\_PKCS1\_CHECK\_2: number;

/\*\* Instructs OpenSSL to always create a new key when using temporary/ephemeral DH parameters. \*/

const SSL\_OP\_SINGLE\_DH\_USE: number;

/\*\* Instructs OpenSSL to always create a new key when using temporary/ephemeral ECDH parameters. \*/

const SSL\_OP\_SINGLE\_ECDH\_USE: number;

const SSL\_OP\_SSLEAY\_080\_CLIENT\_DH\_BUG: number;

const SSL\_OP\_SSLREF2\_REUSE\_CERT\_TYPE\_BUG: number;

const SSL\_OP\_TLS\_BLOCK\_PADDING\_BUG: number;

const SSL\_OP\_TLS\_D5\_BUG: number;

/\*\* Instructs OpenSSL to disable version rollback attack detection. \*/

const SSL\_OP\_TLS\_ROLLBACK\_BUG: number;

const ENGINE\_METHOD\_RSA: number;

const ENGINE\_METHOD\_DSA: number;

const ENGINE\_METHOD\_DH: number;

const ENGINE\_METHOD\_RAND: number;

const ENGINE\_METHOD\_EC: number;

const ENGINE\_METHOD\_CIPHERS: number;

const ENGINE\_METHOD\_DIGESTS: number;

const ENGINE\_METHOD\_PKEY\_METHS: number;

const ENGINE\_METHOD\_PKEY\_ASN1\_METHS: number;

const ENGINE\_METHOD\_ALL: number;

const ENGINE\_METHOD\_NONE: number;

const DH\_CHECK\_P\_NOT\_SAFE\_PRIME: number;

const DH\_CHECK\_P\_NOT\_PRIME: number;

const DH\_UNABLE\_TO\_CHECK\_GENERATOR: number;

const DH\_NOT\_SUITABLE\_GENERATOR: number;

const ALPN\_ENABLED: number;

const RSA\_PKCS1\_PADDING: number;

const RSA\_SSLV23\_PADDING: number;

const RSA\_NO\_PADDING: number;

const RSA\_PKCS1\_OAEP\_PADDING: number;

const RSA\_X931\_PADDING: number;

const RSA\_PKCS1\_PSS\_PADDING: number;

/\*\* Sets the salt length for RSA\_PKCS1\_PSS\_PADDING to the digest size when signing or verifying. \*/

const RSA\_PSS\_SALTLEN\_DIGEST: number;

/\*\* Sets the salt length for RSA\_PKCS1\_PSS\_PADDING to the maximum permissible value when signing data. \*/

const RSA\_PSS\_SALTLEN\_MAX\_SIGN: number;

/\*\* Causes the salt length for RSA\_PKCS1\_PSS\_PADDING to be determined automatically when verifying a signature. \*/

const RSA\_PSS\_SALTLEN\_AUTO: number;

const POINT\_CONVERSION\_COMPRESSED: number;

const POINT\_CONVERSION\_UNCOMPRESSED: number;

const POINT\_CONVERSION\_HYBRID: number;

/\*\* Specifies the built-in default cipher list used by Node.js (colon-separated values). \*/

const defaultCoreCipherList: string;

/\*\* Specifies the active default cipher list used by the current Node.js process (colon-separated values). \*/

const defaultCipherList: string;

}

interface HashOptions extends stream.TransformOptions {

/\*\*

\* For XOF hash functions such as `shake256`, the

\* outputLength option can be used to specify the desired output length in bytes.

\*/

outputLength?: number | undefined;

}

/\*\* @deprecated since v10.0.0 \*/

const fips: boolean;

/\*\*

\* Creates and returns a `Hash` object that can be used to generate hash digests

\* using the given `algorithm`. Optional `options` argument controls stream

\* behavior. For XOF hash functions such as `'shake256'`, the `outputLength` option

\* can be used to specify the desired output length in bytes.

\*

\* The `algorithm` is dependent on the available algorithms supported by the

\* version of OpenSSL on the platform. Examples are `'sha256'`, `'sha512'`, etc.

\* On recent releases of OpenSSL, `openssl list -digest-algorithms`(`openssl list-message-digest-algorithms` for older versions of OpenSSL) will

\* display the available digest algorithms.

\*

\* Example: generating the sha256 sum of a file

\*

\* ```js

\* import {

\* createReadStream

\* } from 'fs';

\* import { argv } from 'process';

\* const {

\* createHash

\* } = await import('crypto');

\*

\* const filename = argv[2];

\*

\* const hash = createHash('sha256');

\*

\* const input = createReadStream(filename);

\* input.on('readable', () => {

\* // Only one element is going to be produced by the

\* // hash stream.

\* const data = input.read();

\* if (data)

\* hash.update(data);

\* else {

\* console.log(`${hash.digest('hex')} ${filename}`);

\* }

\* });

\* ```

\* @since v0.1.92

\* @param options `stream.transform` options

\*/

function createHash(algorithm: string, options?: HashOptions): Hash;

/\*\*

\* Creates and returns an `Hmac` object that uses the given `algorithm` and `key`.

\* Optional `options` argument controls stream behavior.

\*

\* The `algorithm` is dependent on the available algorithms supported by the

\* version of OpenSSL on the platform. Examples are `'sha256'`, `'sha512'`, etc.

\* On recent releases of OpenSSL, `openssl list -digest-algorithms`(`openssl list-message-digest-algorithms` for older versions of OpenSSL) will

\* display the available digest algorithms.

\*

\* The `key` is the HMAC key used to generate the cryptographic HMAC hash. If it is

\* a `KeyObject`, its type must be `secret`.

\*

\* Example: generating the sha256 HMAC of a file

\*

\* ```js

\* import {

\* createReadStream

\* } from 'fs';

\* import { argv } from 'process';

\* const {

\* createHmac

\* } = await import('crypto');

\*

\* const filename = argv[2];

\*

\* const hmac = createHmac('sha256', 'a secret');

\*

\* const input = createReadStream(filename);

\* input.on('readable', () => {

\* // Only one element is going to be produced by the

\* // hash stream.

\* const data = input.read();

\* if (data)

\* hmac.update(data);

\* else {

\* console.log(`${hmac.digest('hex')} ${filename}`);

\* }

\* });

\* ```

\* @since v0.1.94

\* @param options `stream.transform` options

\*/

function createHmac(algorithm: string, key: BinaryLike | KeyObject, options?: stream.TransformOptions): Hmac;

// https://nodejs.org/api/buffer.html#buffer\_buffers\_and\_character\_encodings

type BinaryToTextEncoding = 'base64' | 'base64url' | 'hex';

type CharacterEncoding = 'utf8' | 'utf-8' | 'utf16le' | 'latin1';

type LegacyCharacterEncoding = 'ascii' | 'binary' | 'ucs2' | 'ucs-2';

type Encoding = BinaryToTextEncoding | CharacterEncoding | LegacyCharacterEncoding;

type ECDHKeyFormat = 'compressed' | 'uncompressed' | 'hybrid';

/\*\*

\* The `Hash` class is a utility for creating hash digests of data. It can be

\* used in one of two ways:

\*

\* \* As a `stream` that is both readable and writable, where data is written

\* to produce a computed hash digest on the readable side, or

\* \* Using the `hash.update()` and `hash.digest()` methods to produce the

\* computed hash.

\*

\* The {@link createHash} method is used to create `Hash` instances. `Hash`objects are not to be created directly using the `new` keyword.

\*

\* Example: Using `Hash` objects as streams:

\*

\* ```js

\* const {

\* createHash

\* } = await import('crypto');

\*

\* const hash = createHash('sha256');

\*

\* hash.on('readable', () => {

\* // Only one element is going to be produced by the

\* // hash stream.

\* const data = hash.read();

\* if (data) {

\* console.log(data.toString('hex'));

\* // Prints:

\* // 6a2da20943931e9834fc12cfe5bb47bbd9ae43489a30726962b576f4e3993e50

\* }

\* });

\*

\* hash.write('some data to hash');

\* hash.end();

\* ```

\*

\* Example: Using `Hash` and piped streams:

\*

\* ```js

\* import { createReadStream } from 'fs';

\* import { stdout } from 'process';

\* const { createHash } = await import('crypto');

\*

\* const hash = createHash('sha256');

\*

\* const input = createReadStream('test.js');

\* input.pipe(hash).setEncoding('hex').pipe(stdout);

\* ```

\*

\* Example: Using the `hash.update()` and `hash.digest()` methods:

\*

\* ```js

\* const {

\* createHash

\* } = await import('crypto');

\*

\* const hash = createHash('sha256');

\*

\* hash.update('some data to hash');

\* console.log(hash.digest('hex'));

\* // Prints:

\* // 6a2da20943931e9834fc12cfe5bb47bbd9ae43489a30726962b576f4e3993e50

\* ```

\* @since v0.1.92

\*/

class Hash extends stream.Transform {

private constructor();

/\*\*

\* Creates a new `Hash` object that contains a deep copy of the internal state

\* of the current `Hash` object.

\*

\* The optional `options` argument controls stream behavior. For XOF hash

\* functions such as `'shake256'`, the `outputLength` option can be used to

\* specify the desired output length in bytes.

\*

\* An error is thrown when an attempt is made to copy the `Hash` object after

\* its `hash.digest()` method has been called.

\*

\* ```js

\* // Calculate a rolling hash.

\* const {

\* createHash

\* } = await import('crypto');

\*

\* const hash = createHash('sha256');

\*

\* hash.update('one');

\* console.log(hash.copy().digest('hex'));

\*

\* hash.update('two');

\* console.log(hash.copy().digest('hex'));

\*

\* hash.update('three');

\* console.log(hash.copy().digest('hex'));

\*

\* // Etc.

\* ```

\* @since v13.1.0

\* @param options `stream.transform` options

\*/

copy(options?: stream.TransformOptions): Hash;

/\*\*

\* Updates the hash content with the given `data`, the encoding of which

\* is given in `inputEncoding`.

\* If `encoding` is not provided, and the `data` is a string, an

\* encoding of `'utf8'` is enforced. If `data` is a `Buffer`, `TypedArray`, or`DataView`, then `inputEncoding` is ignored.

\*

\* This can be called many times with new data as it is streamed.

\* @since v0.1.92

\* @param inputEncoding The `encoding` of the `data` string.

\*/

update(data: BinaryLike): Hash;

update(data: string, inputEncoding: Encoding): Hash;

/\*\*

\* Calculates the digest of all of the data passed to be hashed (using the `hash.update()` method).

\* If `encoding` is provided a string will be returned; otherwise

\* a `Buffer` is returned.

\*

\* The `Hash` object can not be used again after `hash.digest()` method has been

\* called. Multiple calls will cause an error to be thrown.

\* @since v0.1.92

\* @param encoding The `encoding` of the return value.

\*/

digest(): Buffer;

digest(encoding: BinaryToTextEncoding): string;

}

/\*\*

\* The `Hmac` class is a utility for creating cryptographic HMAC digests. It can

\* be used in one of two ways:

\*

\* \* As a `stream` that is both readable and writable, where data is written

\* to produce a computed HMAC digest on the readable side, or

\* \* Using the `hmac.update()` and `hmac.digest()` methods to produce the

\* computed HMAC digest.

\*

\* The {@link createHmac} method is used to create `Hmac` instances. `Hmac`objects are not to be created directly using the `new` keyword.

\*

\* Example: Using `Hmac` objects as streams:

\*

\* ```js

\* const {

\* createHmac

\* } = await import('crypto');

\*

\* const hmac = createHmac('sha256', 'a secret');

\*

\* hmac.on('readable', () => {

\* // Only one element is going to be produced by the

\* // hash stream.

\* const data = hmac.read();

\* if (data) {

\* console.log(data.toString('hex'));

\* // Prints:

\* // 7fd04df92f636fd450bc841c9418e5825c17f33ad9c87c518115a45971f7f77e

\* }

\* });

\*

\* hmac.write('some data to hash');

\* hmac.end();

\* ```

\*

\* Example: Using `Hmac` and piped streams:

\*

\* ```js

\* import { createReadStream } from 'fs';

\* import { stdout } from 'process';

\* const {

\* createHmac

\* } = await import('crypto');

\*

\* const hmac = createHmac('sha256', 'a secret');

\*

\* const input = createReadStream('test.js');

\* input.pipe(hmac).pipe(stdout);

\* ```

\*

\* Example: Using the `hmac.update()` and `hmac.digest()` methods:

\*

\* ```js

\* const {

\* createHmac

\* } = await import('crypto');

\*

\* const hmac = createHmac('sha256', 'a secret');

\*

\* hmac.update('some data to hash');

\* console.log(hmac.digest('hex'));

\* // Prints:

\* // 7fd04df92f636fd450bc841c9418e5825c17f33ad9c87c518115a45971f7f77e

\* ```

\* @since v0.1.94

\*/

class Hmac extends stream.Transform {

private constructor();

/\*\*

\* Updates the `Hmac` content with the given `data`, the encoding of which

\* is given in `inputEncoding`.

\* If `encoding` is not provided, and the `data` is a string, an

\* encoding of `'utf8'` is enforced. If `data` is a `Buffer`, `TypedArray`, or`DataView`, then `inputEncoding` is ignored.

\*

\* This can be called many times with new data as it is streamed.

\* @since v0.1.94

\* @param inputEncoding The `encoding` of the `data` string.

\*/

update(data: BinaryLike): Hmac;

update(data: string, inputEncoding: Encoding): Hmac;

/\*\*

\* Calculates the HMAC digest of all of the data passed using `hmac.update()`.

\* If `encoding` is

\* provided a string is returned; otherwise a `Buffer` is returned;

\*

\* The `Hmac` object can not be used again after `hmac.digest()` has been

\* called. Multiple calls to `hmac.digest()` will result in an error being thrown.

\* @since v0.1.94

\* @param encoding The `encoding` of the return value.

\*/

digest(): Buffer;

digest(encoding: BinaryToTextEncoding): string;

}

type KeyObjectType = 'secret' | 'public' | 'private';

interface KeyExportOptions<T extends KeyFormat> {

type: 'pkcs1' | 'spki' | 'pkcs8' | 'sec1';

format: T;

cipher?: string | undefined;

passphrase?: string | Buffer | undefined;

}

interface JwkKeyExportOptions {

format: 'jwk';

}

interface JsonWebKey {

crv?: string | undefined;

d?: string | undefined;

dp?: string | undefined;

dq?: string | undefined;

e?: string | undefined;

k?: string | undefined;

kty?: string | undefined;

n?: string | undefined;

p?: string | undefined;

q?: string | undefined;

qi?: string | undefined;

x?: string | undefined;

y?: string | undefined;

[key: string]: unknown;

}

interface AsymmetricKeyDetails {

/\*\*

\* Key size in bits (RSA, DSA).

\*/

modulusLength?: number | undefined;

/\*\*

\* Public exponent (RSA).

\*/

publicExponent?: bigint | undefined;

/\*\*

\* Name of the message digest (RSA-PSS).

\*/

hashAlgorithm?: string | undefined;

/\*\*

\* Name of the message digest used by MGF1 (RSA-PSS).

\*/

mgf1HashAlgorithm?: string | undefined;

/\*\*

\* Minimal salt length in bytes (RSA-PSS).

\*/

saltLength?: number | undefined;

/\*\*

\* Size of q in bits (DSA).

\*/

divisorLength?: number | undefined;

/\*\*

\* Name of the curve (EC).

\*/

namedCurve?: string | undefined;

}

interface JwkKeyExportOptions {

format: 'jwk';

}

/\*\*

\* Node.js uses a `KeyObject` class to represent a symmetric or asymmetric key,

\* and each kind of key exposes different functions. The {@link createSecretKey}, {@link createPublicKey} and {@link createPrivateKey} methods are used to create `KeyObject`instances. `KeyObject`

\* objects are not to be created directly using the `new`keyword.

\*

\* Most applications should consider using the new `KeyObject` API instead of

\* passing keys as strings or `Buffer`s due to improved security features.

\*

\* `KeyObject` instances can be passed to other threads via `postMessage()`.

\* The receiver obtains a cloned `KeyObject`, and the `KeyObject` does not need to

\* be listed in the `transferList` argument.

\* @since v11.6.0

\*/

class KeyObject {

private constructor();

/\*\*

\* Example: Converting a `CryptoKey` instance to a `KeyObject`:

\*

\* ```js

\* const { webcrypto, KeyObject } = await import('crypto');

\* const { subtle } = webcrypto;

\*

\* const key = await subtle.generateKey({

\* name: 'HMAC',

\* hash: 'SHA-256',

\* length: 256

\* }, true, ['sign', 'verify']);

\*

\* const keyObject = KeyObject.from(key);

\* console.log(keyObject.symmetricKeySize);

\* // Prints: 32 (symmetric key size in bytes)

\* ```

\* @since v15.0.0

\*/

static from(key: webcrypto.CryptoKey): KeyObject;

/\*\*

\* For asymmetric keys, this property represents the type of the key. Supported key

\* types are:

\*

\* \* `'rsa'` (OID 1.2.840.113549.1.1.1)

\* \* `'rsa-pss'` (OID 1.2.840.113549.1.1.10)

\* \* `'dsa'` (OID 1.2.840.10040.4.1)

\* \* `'ec'` (OID 1.2.840.10045.2.1)

\* \* `'x25519'` (OID 1.3.101.110)

\* \* `'x448'` (OID 1.3.101.111)

\* \* `'ed25519'` (OID 1.3.101.112)

\* \* `'ed448'` (OID 1.3.101.113)

\* \* `'dh'` (OID 1.2.840.113549.1.3.1)

\*

\* This property is `undefined` for unrecognized `KeyObject` types and symmetric

\* keys.

\* @since v11.6.0

\*/

asymmetricKeyType?: KeyType | undefined;

/\*\*

\* For asymmetric keys, this property represents the size of the embedded key in

\* bytes. This property is `undefined` for symmetric keys.

\*/

asymmetricKeySize?: number | undefined;

/\*\*

\* This property exists only on asymmetric keys. Depending on the type of the key,

\* this object contains information about the key. None of the information obtained

\* through this property can be used to uniquely identify a key or to compromise

\* the security of the key.

\*

\* For RSA-PSS keys, if the key material contains a `RSASSA-PSS-params` sequence,

\* the `hashAlgorithm`, `mgf1HashAlgorithm`, and `saltLength` properties will be

\* set.

\*

\* Other key details might be exposed via this API using additional attributes.

\* @since v15.7.0

\*/

asymmetricKeyDetails?: AsymmetricKeyDetails | undefined;

/\*\*

\* For symmetric keys, the following encoding options can be used:

\*

\* For public keys, the following encoding options can be used:

\*

\* For private keys, the following encoding options can be used:

\*

\* The result type depends on the selected encoding format, when PEM the

\* result is a string, when DER it will be a buffer containing the data

\* encoded as DER, when [JWK](https://tools.ietf.org/html/rfc7517) it will be an object.

\*

\* When [JWK](https://tools.ietf.org/html/rfc7517) encoding format was selected, all other encoding options are

\* ignored.

\*

\* PKCS#1, SEC1, and PKCS#8 type keys can be encrypted by using a combination of

\* the `cipher` and `format` options. The PKCS#8 `type` can be used with any`format` to encrypt any key algorithm (RSA, EC, or DH) by specifying a`cipher`. PKCS#1 and SEC1 can only be

\* encrypted by specifying a `cipher`when the PEM `format` is used. For maximum compatibility, use PKCS#8 for

\* encrypted private keys. Since PKCS#8 defines its own

\* encryption mechanism, PEM-level encryption is not supported when encrypting

\* a PKCS#8 key. See [RFC 5208](https://www.rfc-editor.org/rfc/rfc5208.txt) for PKCS#8 encryption and [RFC 1421](https://www.rfc-editor.org/rfc/rfc1421.txt) for

\* PKCS#1 and SEC1 encryption.

\* @since v11.6.0

\*/

export(options: KeyExportOptions<'pem'>): string | Buffer;

export(options?: KeyExportOptions<'der'>): Buffer;

export(options?: JwkKeyExportOptions): JsonWebKey;

/\*\*

\* For secret keys, this property represents the size of the key in bytes. This

\* property is `undefined` for asymmetric keys.

\* @since v11.6.0

\*/

symmetricKeySize?: number | undefined;

/\*\*

\* Depending on the type of this `KeyObject`, this property is either`'secret'` for secret (symmetric) keys, `'public'` for public (asymmetric) keys

\* or `'private'` for private (asymmetric) keys.

\* @since v11.6.0

\*/

type: KeyObjectType;

}

type CipherCCMTypes = 'aes-128-ccm' | 'aes-192-ccm' | 'aes-256-ccm' | 'chacha20-poly1305';

type CipherGCMTypes = 'aes-128-gcm' | 'aes-192-gcm' | 'aes-256-gcm';

type BinaryLike = string | NodeJS.ArrayBufferView;

type CipherKey = BinaryLike | KeyObject;

interface CipherCCMOptions extends stream.TransformOptions {

authTagLength: number;

}

interface CipherGCMOptions extends stream.TransformOptions {

authTagLength?: number | undefined;

}

/\*\*

\* Creates and returns a `Cipher` object that uses the given `algorithm` and`password`.

\*

\* The `options` argument controls stream behavior and is optional except when a

\* cipher in CCM or OCB mode is used (e.g. `'aes-128-ccm'`). In that case, the`authTagLength` option is required and specifies the length of the

\* authentication tag in bytes, see `CCM mode`. In GCM mode, the `authTagLength`option is not required but can be used to set the length of the authentication

\* tag that will be returned by `getAuthTag()` and defaults to 16 bytes.

\*

\* The `algorithm` is dependent on OpenSSL, examples are `'aes192'`, etc. On

\* recent OpenSSL releases, `openssl list -cipher-algorithms`(`openssl list-cipher-algorithms` for older versions of OpenSSL) will

\* display the available cipher algorithms.

\*

\* The `password` is used to derive the cipher key and initialization vector (IV).

\* The value must be either a `'latin1'` encoded string, a `Buffer`, a`TypedArray`, or a `DataView`.

\*

\* The implementation of `crypto.createCipher()` derives keys using the OpenSSL

\* function [`EVP\_BytesToKey`](https://www.openssl.org/docs/man1.1.0/crypto/EVP\_BytesToKey.html) with the digest algorithm set to MD5, one

\* iteration, and no salt. The lack of salt allows dictionary attacks as the same

\* password always creates the same key. The low iteration count and

\* non-cryptographically secure hash algorithm allow passwords to be tested very

\* rapidly.

\*

\* In line with OpenSSL's recommendation to use a more modern algorithm instead of [`EVP\_BytesToKey`](https://www.openssl.org/docs/man1.1.0/crypto/EVP\_BytesToKey.html) it is recommended that

\* developers derive a key and IV on

\* their own using {@link scrypt} and to use {@link createCipheriv} to create the `Cipher` object. Users should not use ciphers with counter mode

\* (e.g. CTR, GCM, or CCM) in `crypto.createCipher()`. A warning is emitted when

\* they are used in order to avoid the risk of IV reuse that causes

\* vulnerabilities. For the case when IV is reused in GCM, see [Nonce-Disrespecting Adversaries](https://github.com/nonce-disrespect/nonce-disrespect) for details.

\* @since v0.1.94

\* @deprecated Since v10.0.0 - Use {@link createCipheriv} instead.

\* @param options `stream.transform` options

\*/

function createCipher(algorithm: CipherCCMTypes, password: BinaryLike, options: CipherCCMOptions): CipherCCM;

/\*\* @deprecated since v10.0.0 use `createCipheriv()` \*/

function createCipher(algorithm: CipherGCMTypes, password: BinaryLike, options?: CipherGCMOptions): CipherGCM;

/\*\* @deprecated since v10.0.0 use `createCipheriv()` \*/

function createCipher(algorithm: string, password: BinaryLike, options?: stream.TransformOptions): Cipher;

/\*\*

\* Creates and returns a `Cipher` object, with the given `algorithm`, `key` and

\* initialization vector (`iv`).

\*

\* The `options` argument controls stream behavior and is optional except when a

\* cipher in CCM or OCB mode is used (e.g. `'aes-128-ccm'`). In that case, the`authTagLength` option is required and specifies the length of the

\* authentication tag in bytes, see `CCM mode`. In GCM mode, the `authTagLength`option is not required but can be used to set the length of the authentication

\* tag that will be returned by `getAuthTag()` and defaults to 16 bytes.

\*

\* The `algorithm` is dependent on OpenSSL, examples are `'aes192'`, etc. On

\* recent OpenSSL releases, `openssl list -cipher-algorithms`(`openssl list-cipher-algorithms` for older versions of OpenSSL) will

\* display the available cipher algorithms.

\*

\* The `key` is the raw key used by the `algorithm` and `iv` is an [initialization vector](https://en.wikipedia.org/wiki/Initialization\_vector). Both arguments must be `'utf8'` encoded

\* strings,`Buffers`, `TypedArray`, or `DataView`s. The `key` may optionally be

\* a `KeyObject` of type `secret`. If the cipher does not need

\* an initialization vector, `iv` may be `null`.

\*

\* When passing strings for `key` or `iv`, please consider `caveats when using strings as inputs to cryptographic APIs`.

\*

\* Initialization vectors should be unpredictable and unique; ideally, they will be

\* cryptographically random. They do not have to be secret: IVs are typically just

\* added to ciphertext messages unencrypted. It may sound contradictory that

\* something has to be unpredictable and unique, but does not have to be secret;

\* remember that an attacker must not be able to predict ahead of time what a

\* given IV will be.

\* @since v0.1.94

\* @param options `stream.transform` options

\*/

function createCipheriv(algorithm: CipherCCMTypes, key: CipherKey, iv: BinaryLike | null, options: CipherCCMOptions): CipherCCM;

function createCipheriv(algorithm: CipherGCMTypes, key: CipherKey, iv: BinaryLike | null, options?: CipherGCMOptions): CipherGCM;

function createCipheriv(algorithm: string, key: CipherKey, iv: BinaryLike | null, options?: stream.TransformOptions): Cipher;

/\*\*

\* Instances of the `Cipher` class are used to encrypt data. The class can be

\* used in one of two ways:

\*

\* \* As a `stream` that is both readable and writable, where plain unencrypted

\* data is written to produce encrypted data on the readable side, or

\* \* Using the `cipher.update()` and `cipher.final()` methods to produce

\* the encrypted data.

\*

\* The {@link createCipher} or {@link createCipheriv} methods are

\* used to create `Cipher` instances. `Cipher` objects are not to be created

\* directly using the `new` keyword.

\*

\* Example: Using `Cipher` objects as streams:

\*

\* ```js

\* const {

\* scrypt,

\* randomFill,

\* createCipheriv

\* } = await import('crypto');

\*

\* const algorithm = 'aes-192-cbc';

\* const password = 'Password used to generate key';

\*

\* // First, we'll generate the key. The key length is dependent on the algorithm.

\* // In this case for aes192, it is 24 bytes (192 bits).

\* scrypt(password, 'salt', 24, (err, key) => {

\* if (err) throw err;

\* // Then, we'll generate a random initialization vector

\* randomFill(new Uint8Array(16), (err, iv) => {

\* if (err) throw err;

\*

\* // Once we have the key and iv, we can create and use the cipher...

\* const cipher = createCipheriv(algorithm, key, iv);

\*

\* let encrypted = '';

\* cipher.setEncoding('hex');

\*

\* cipher.on('data', (chunk) => encrypted += chunk);

\* cipher.on('end', () => console.log(encrypted));

\*

\* cipher.write('some clear text data');

\* cipher.end();

\* });

\* });

\* ```

\*

\* Example: Using `Cipher` and piped streams:

\*

\* ```js

\* import {

\* createReadStream,

\* createWriteStream,

\* } from 'fs';

\*

\* import {

\* pipeline

\* } from 'stream';

\*

\* const {

\* scrypt,

\* randomFill,

\* createCipheriv

\* } = await import('crypto');

\*

\* const algorithm = 'aes-192-cbc';

\* const password = 'Password used to generate key';

\*

\* // First, we'll generate the key. The key length is dependent on the algorithm.

\* // In this case for aes192, it is 24 bytes (192 bits).

\* scrypt(password, 'salt', 24, (err, key) => {

\* if (err) throw err;

\* // Then, we'll generate a random initialization vector

\* randomFill(new Uint8Array(16), (err, iv) => {

\* if (err) throw err;

\*

\* const cipher = createCipheriv(algorithm, key, iv);

\*

\* const input = createReadStream('test.js');

\* const output = createWriteStream('test.enc');

\*

\* pipeline(input, cipher, output, (err) => {

\* if (err) throw err;

\* });

\* });

\* });

\* ```

\*

\* Example: Using the `cipher.update()` and `cipher.final()` methods:

\*

\* ```js

\* const {

\* scrypt,

\* randomFill,

\* createCipheriv

\* } = await import('crypto');

\*

\* const algorithm = 'aes-192-cbc';

\* const password = 'Password used to generate key';

\*

\* // First, we'll generate the key. The key length is dependent on the algorithm.

\* // In this case for aes192, it is 24 bytes (192 bits).

\* scrypt(password, 'salt', 24, (err, key) => {

\* if (err) throw err;

\* // Then, we'll generate a random initialization vector

\* randomFill(new Uint8Array(16), (err, iv) => {

\* if (err) throw err;

\*

\* const cipher = createCipheriv(algorithm, key, iv);

\*

\* let encrypted = cipher.update('some clear text data', 'utf8', 'hex');

\* encrypted += cipher.final('hex');

\* console.log(encrypted);

\* });

\* });

\* ```

\* @since v0.1.94

\*/

class Cipher extends stream.Transform {

private constructor();

/\*\*

\* Updates the cipher with `data`. If the `inputEncoding` argument is given,

\* the `data`argument is a string using the specified encoding. If the `inputEncoding`argument is not given, `data` must be a `Buffer`, `TypedArray`, or`DataView`. If `data` is a `Buffer`,

\* `TypedArray`, or `DataView`, then`inputEncoding` is ignored.

\*

\* The `outputEncoding` specifies the output format of the enciphered

\* data. If the `outputEncoding`is specified, a string using the specified encoding is returned. If no`outputEncoding` is provided, a `Buffer` is returned.

\*

\* The `cipher.update()` method can be called multiple times with new data until `cipher.final()` is called. Calling `cipher.update()` after `cipher.final()` will result in an error being

\* thrown.

\* @since v0.1.94

\* @param inputEncoding The `encoding` of the data.

\* @param outputEncoding The `encoding` of the return value.

\*/

update(data: BinaryLike): Buffer;

update(data: string, inputEncoding: Encoding): Buffer;

update(data: NodeJS.ArrayBufferView, inputEncoding: undefined, outputEncoding: Encoding): string;

update(data: string, inputEncoding: Encoding | undefined, outputEncoding: Encoding): string;

/\*\*

\* Once the `cipher.final()` method has been called, the `Cipher` object can no

\* longer be used to encrypt data. Attempts to call `cipher.final()` more than

\* once will result in an error being thrown.

\* @since v0.1.94

\* @param outputEncoding The `encoding` of the return value.

\* @return Any remaining enciphered contents. If `outputEncoding` is specified, a string is returned. If an `outputEncoding` is not provided, a {@link Buffer} is returned.

\*/

final(): Buffer;

final(outputEncoding: BufferEncoding): string;

/\*\*

\* When using block encryption algorithms, the `Cipher` class will automatically

\* add padding to the input data to the appropriate block size. To disable the

\* default padding call `cipher.setAutoPadding(false)`.

\*

\* When `autoPadding` is `false`, the length of the entire input data must be a

\* multiple of the cipher's block size or `cipher.final()` will throw an error.

\* Disabling automatic padding is useful for non-standard padding, for instance

\* using `0x0` instead of PKCS padding.

\*

\* The `cipher.setAutoPadding()` method must be called before `cipher.final()`.

\* @since v0.7.1

\* @param [autoPadding=true]

\* @return for method chaining.

\*/

setAutoPadding(autoPadding?: boolean): this;

}

interface CipherCCM extends Cipher {

setAAD(

buffer: NodeJS.ArrayBufferView,

options: {

plaintextLength: number;

}

): this;

getAuthTag(): Buffer;

}

interface CipherGCM extends Cipher {

setAAD(

buffer: NodeJS.ArrayBufferView,

options?: {

plaintextLength: number;

}

): this;

getAuthTag(): Buffer;

}

/\*\*

\* Creates and returns a `Decipher` object that uses the given `algorithm` and`password` (key).

\*

\* The `options` argument controls stream behavior and is optional except when a

\* cipher in CCM or OCB mode is used (e.g. `'aes-128-ccm'`). In that case, the`authTagLength` option is required and specifies the length of the

\* authentication tag in bytes, see `CCM mode`.

\*

\* The implementation of `crypto.createDecipher()` derives keys using the OpenSSL

\* function [`EVP\_BytesToKey`](https://www.openssl.org/docs/man1.1.0/crypto/EVP\_BytesToKey.html) with the digest algorithm set to MD5, one

\* iteration, and no salt. The lack of salt allows dictionary attacks as the same

\* password always creates the same key. The low iteration count and

\* non-cryptographically secure hash algorithm allow passwords to be tested very

\* rapidly.

\*

\* In line with OpenSSL's recommendation to use a more modern algorithm instead of [`EVP\_BytesToKey`](https://www.openssl.org/docs/man1.1.0/crypto/EVP\_BytesToKey.html) it is recommended that

\* developers derive a key and IV on

\* their own using {@link scrypt} and to use {@link createDecipheriv} to create the `Decipher` object.

\* @since v0.1.94

\* @deprecated Since v10.0.0 - Use {@link createDecipheriv} instead.

\* @param options `stream.transform` options

\*/

function createDecipher(algorithm: CipherCCMTypes, password: BinaryLike, options: CipherCCMOptions): DecipherCCM;

/\*\* @deprecated since v10.0.0 use `createDecipheriv()` \*/

function createDecipher(algorithm: CipherGCMTypes, password: BinaryLike, options?: CipherGCMOptions): DecipherGCM;

/\*\* @deprecated since v10.0.0 use `createDecipheriv()` \*/

function createDecipher(algorithm: string, password: BinaryLike, options?: stream.TransformOptions): Decipher;

/\*\*

\* Creates and returns a `Decipher` object that uses the given `algorithm`, `key`and initialization vector (`iv`).

\*

\* The `options` argument controls stream behavior and is optional except when a

\* cipher in CCM or OCB mode is used (e.g. `'aes-128-ccm'`). In that case, the`authTagLength` option is required and specifies the length of the

\* authentication tag in bytes, see `CCM mode`. In GCM mode, the `authTagLength`option is not required but can be used to restrict accepted authentication tags

\* to those with the specified length.

\*

\* The `algorithm` is dependent on OpenSSL, examples are `'aes192'`, etc. On

\* recent OpenSSL releases, `openssl list -cipher-algorithms`(`openssl list-cipher-algorithms` for older versions of OpenSSL) will

\* display the available cipher algorithms.

\*

\* The `key` is the raw key used by the `algorithm` and `iv` is an [initialization vector](https://en.wikipedia.org/wiki/Initialization\_vector). Both arguments must be `'utf8'` encoded

\* strings,`Buffers`, `TypedArray`, or `DataView`s. The `key` may optionally be

\* a `KeyObject` of type `secret`. If the cipher does not need

\* an initialization vector, `iv` may be `null`.

\*

\* When passing strings for `key` or `iv`, please consider `caveats when using strings as inputs to cryptographic APIs`.

\*

\* Initialization vectors should be unpredictable and unique; ideally, they will be

\* cryptographically random. They do not have to be secret: IVs are typically just

\* added to ciphertext messages unencrypted. It may sound contradictory that

\* something has to be unpredictable and unique, but does not have to be secret;

\* remember that an attacker must not be able to predict ahead of time what a given

\* IV will be.

\* @since v0.1.94

\* @param options `stream.transform` options

\*/

function createDecipheriv(algorithm: CipherCCMTypes, key: CipherKey, iv: BinaryLike | null, options: CipherCCMOptions): DecipherCCM;

function createDecipheriv(algorithm: CipherGCMTypes, key: CipherKey, iv: BinaryLike | null, options?: CipherGCMOptions): DecipherGCM;

function createDecipheriv(algorithm: string, key: CipherKey, iv: BinaryLike | null, options?: stream.TransformOptions): Decipher;

/\*\*

\* Instances of the `Decipher` class are used to decrypt data. The class can be

\* used in one of two ways:

\*

\* \* As a `stream` that is both readable and writable, where plain encrypted

\* data is written to produce unencrypted data on the readable side, or

\* \* Using the `decipher.update()` and `decipher.final()` methods to

\* produce the unencrypted data.

\*

\* The {@link createDecipher} or {@link createDecipheriv} methods are

\* used to create `Decipher` instances. `Decipher` objects are not to be created

\* directly using the `new` keyword.

\*

\* Example: Using `Decipher` objects as streams:

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const {

\* scryptSync,

\* createDecipheriv

\* } = await import('crypto');

\*

\* const algorithm = 'aes-192-cbc';

\* const password = 'Password used to generate key';

\* // Key length is dependent on the algorithm. In this case for aes192, it is

\* // 24 bytes (192 bits).

\* // Use the async `crypto.scrypt()` instead.

\* const key = scryptSync(password, 'salt', 24);

\* // The IV is usually passed along with the ciphertext.

\* const iv = Buffer.alloc(16, 0); // Initialization vector.

\*

\* const decipher = createDecipheriv(algorithm, key, iv);

\*

\* let decrypted = '';

\* decipher.on('readable', () => {

\* while (null !== (chunk = decipher.read())) {

\* decrypted += chunk.toString('utf8');

\* }

\* });

\* decipher.on('end', () => {

\* console.log(decrypted);

\* // Prints: some clear text data

\* });

\*

\* // Encrypted with same algorithm, key and iv.

\* const encrypted =

\* 'e5f79c5915c02171eec6b212d5520d44480993d7d622a7c4c2da32f6efda0ffa';

\* decipher.write(encrypted, 'hex');

\* decipher.end();

\* ```

\*

\* Example: Using `Decipher` and piped streams:

\*

\* ```js

\* import {

\* createReadStream,

\* createWriteStream,

\* } from 'fs';

\* import { Buffer } from 'buffer';

\* const {

\* scryptSync,

\* createDecipheriv

\* } = await import('crypto');

\*

\* const algorithm = 'aes-192-cbc';

\* const password = 'Password used to generate key';

\* // Use the async `crypto.scrypt()` instead.

\* const key = scryptSync(password, 'salt', 24);

\* // The IV is usually passed along with the ciphertext.

\* const iv = Buffer.alloc(16, 0); // Initialization vector.

\*

\* const decipher = createDecipheriv(algorithm, key, iv);

\*

\* const input = createReadStream('test.enc');

\* const output = createWriteStream('test.js');

\*

\* input.pipe(decipher).pipe(output);

\* ```

\*

\* Example: Using the `decipher.update()` and `decipher.final()` methods:

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const {

\* scryptSync,

\* createDecipheriv

\* } = await import('crypto');

\*

\* const algorithm = 'aes-192-cbc';

\* const password = 'Password used to generate key';

\* // Use the async `crypto.scrypt()` instead.

\* const key = scryptSync(password, 'salt', 24);

\* // The IV is usually passed along with the ciphertext.

\* const iv = Buffer.alloc(16, 0); // Initialization vector.

\*

\* const decipher = createDecipheriv(algorithm, key, iv);

\*

\* // Encrypted using same algorithm, key and iv.

\* const encrypted =

\* 'e5f79c5915c02171eec6b212d5520d44480993d7d622a7c4c2da32f6efda0ffa';

\* let decrypted = decipher.update(encrypted, 'hex', 'utf8');

\* decrypted += decipher.final('utf8');

\* console.log(decrypted);

\* // Prints: some clear text data

\* ```

\* @since v0.1.94

\*/

class Decipher extends stream.Transform {

private constructor();

/\*\*

\* Updates the decipher with `data`. If the `inputEncoding` argument is given,

\* the `data`argument is a string using the specified encoding. If the `inputEncoding`argument is not given, `data` must be a `Buffer`. If `data` is a `Buffer` then `inputEncoding` is

\* ignored.

\*

\* The `outputEncoding` specifies the output format of the enciphered

\* data. If the `outputEncoding`is specified, a string using the specified encoding is returned. If no`outputEncoding` is provided, a `Buffer` is returned.

\*

\* The `decipher.update()` method can be called multiple times with new data until `decipher.final()` is called. Calling `decipher.update()` after `decipher.final()` will result in an error

\* being thrown.

\* @since v0.1.94

\* @param inputEncoding The `encoding` of the `data` string.

\* @param outputEncoding The `encoding` of the return value.

\*/

update(data: NodeJS.ArrayBufferView): Buffer;

update(data: string, inputEncoding: Encoding): Buffer;

update(data: NodeJS.ArrayBufferView, inputEncoding: undefined, outputEncoding: Encoding): string;

update(data: string, inputEncoding: Encoding | undefined, outputEncoding: Encoding): string;

/\*\*

\* Once the `decipher.final()` method has been called, the `Decipher` object can

\* no longer be used to decrypt data. Attempts to call `decipher.final()` more

\* than once will result in an error being thrown.

\* @since v0.1.94

\* @param outputEncoding The `encoding` of the return value.

\* @return Any remaining deciphered contents. If `outputEncoding` is specified, a string is returned. If an `outputEncoding` is not provided, a {@link Buffer} is returned.

\*/

final(): Buffer;

final(outputEncoding: BufferEncoding): string;

/\*\*

\* When data has been encrypted without standard block padding, calling`decipher.setAutoPadding(false)` will disable automatic padding to prevent `decipher.final()` from checking for and

\* removing padding.

\*

\* Turning auto padding off will only work if the input data's length is a

\* multiple of the ciphers block size.

\*

\* The `decipher.setAutoPadding()` method must be called before `decipher.final()`.

\* @since v0.7.1

\* @param [autoPadding=true]

\* @return for method chaining.

\*/

setAutoPadding(auto\_padding?: boolean): this;

}

interface DecipherCCM extends Decipher {

setAuthTag(buffer: NodeJS.ArrayBufferView): this;

setAAD(

buffer: NodeJS.ArrayBufferView,

options: {

plaintextLength: number;

}

): this;

}

interface DecipherGCM extends Decipher {

setAuthTag(buffer: NodeJS.ArrayBufferView): this;

setAAD(

buffer: NodeJS.ArrayBufferView,

options?: {

plaintextLength: number;

}

): this;

}

interface PrivateKeyInput {

key: string | Buffer;

format?: KeyFormat | undefined;

type?: 'pkcs1' | 'pkcs8' | 'sec1' | undefined;

passphrase?: string | Buffer | undefined;

}

interface PublicKeyInput {

key: string | Buffer;

format?: KeyFormat | undefined;

type?: 'pkcs1' | 'spki' | undefined;

}

/\*\*

\* Asynchronously generates a new random secret key of the given `length`. The`type` will determine which validations will be performed on the `length`.

\*

\* ```js

\* const {

\* generateKey

\* } = await import('crypto');

\*

\* generateKey('hmac', { length: 64 }, (err, key) => {

\* if (err) throw err;

\* console.log(key.export().toString('hex')); // 46e..........620

\* });

\* ```

\* @since v15.0.0

\* @param type The intended use of the generated secret key. Currently accepted values are `'hmac'` and `'aes'`.

\*/

function generateKey(

type: 'hmac' | 'aes',

options: {

length: number;

},

callback: (err: Error | null, key: KeyObject) => void

): void;

/\*\*

\* Synchronously generates a new random secret key of the given `length`. The`type` will determine which validations will be performed on the `length`.

\*

\* ```js

\* const {

\* generateKeySync

\* } = await import('crypto');

\*

\* const key = generateKeySync('hmac', { length: 64 });

\* console.log(key.export().toString('hex')); // e89..........41e

\* ```

\* @since v15.0.0

\* @param type The intended use of the generated secret key. Currently accepted values are `'hmac'` and `'aes'`.

\*/

function generateKeySync(

type: 'hmac' | 'aes',

options: {

length: number;

}

): KeyObject;

interface JsonWebKeyInput {

key: JsonWebKey;

format: 'jwk';

}

/\*\*

\* Creates and returns a new key object containing a private key. If `key` is a

\* string or `Buffer`, `format` is assumed to be `'pem'`; otherwise, `key`must be an object with the properties described above.

\*

\* If the private key is encrypted, a `passphrase` must be specified. The length

\* of the passphrase is limited to 1024 bytes.

\* @since v11.6.0

\*/

function createPrivateKey(key: PrivateKeyInput | string | Buffer | JsonWebKeyInput): KeyObject;

/\*\*

\* Creates and returns a new key object containing a public key. If `key` is a

\* string or `Buffer`, `format` is assumed to be `'pem'`; if `key` is a `KeyObject`with type `'private'`, the public key is derived from the given private key;

\* otherwise, `key` must be an object with the properties described above.

\*

\* If the format is `'pem'`, the `'key'` may also be an X.509 certificate.

\*

\* Because public keys can be derived from private keys, a private key may be

\* passed instead of a public key. In that case, this function behaves as if {@link createPrivateKey} had been called, except that the type of the

\* returned `KeyObject` will be `'public'` and that the private key cannot be

\* extracted from the returned `KeyObject`. Similarly, if a `KeyObject` with type`'private'` is given, a new `KeyObject` with type `'public'` will be returned

\* and it will be impossible to extract the private key from the returned object.

\* @since v11.6.0

\*/

function createPublicKey(key: PublicKeyInput | string | Buffer | KeyObject | JsonWebKeyInput): KeyObject;

/\*\*

\* Creates and returns a new key object containing a secret key for symmetric

\* encryption or `Hmac`.

\* @since v11.6.0

\* @param encoding The string encoding when `key` is a string.

\*/

function createSecretKey(key: NodeJS.ArrayBufferView): KeyObject;

function createSecretKey(key: string, encoding: BufferEncoding): KeyObject;

/\*\*

\* Creates and returns a `Sign` object that uses the given `algorithm`. Use {@link getHashes} to obtain the names of the available digest algorithms.

\* Optional `options` argument controls the `stream.Writable` behavior.

\*

\* In some cases, a `Sign` instance can be created using the name of a signature

\* algorithm, such as `'RSA-SHA256'`, instead of a digest algorithm. This will use

\* the corresponding digest algorithm. This does not work for all signature

\* algorithms, such as `'ecdsa-with-SHA256'`, so it is best to always use digest

\* algorithm names.

\* @since v0.1.92

\* @param options `stream.Writable` options

\*/

function createSign(algorithm: string, options?: stream.WritableOptions): Sign;

type DSAEncoding = 'der' | 'ieee-p1363';

interface SigningOptions {

/\*\*

\* @See crypto.constants.RSA\_PKCS1\_PADDING

\*/

padding?: number | undefined;

saltLength?: number | undefined;

dsaEncoding?: DSAEncoding | undefined;

}

interface SignPrivateKeyInput extends PrivateKeyInput, SigningOptions {}

interface SignKeyObjectInput extends SigningOptions {

key: KeyObject;

}

interface VerifyPublicKeyInput extends PublicKeyInput, SigningOptions {}

interface VerifyKeyObjectInput extends SigningOptions {

key: KeyObject;

}

type KeyLike = string | Buffer | KeyObject;

/\*\*

\* The `Sign` class is a utility for generating signatures. It can be used in one

\* of two ways:

\*

\* \* As a writable `stream`, where data to be signed is written and the `sign.sign()` method is used to generate and return the signature, or

\* \* Using the `sign.update()` and `sign.sign()` methods to produce the

\* signature.

\*

\* The {@link createSign} method is used to create `Sign` instances. The

\* argument is the string name of the hash function to use. `Sign` objects are not

\* to be created directly using the `new` keyword.

\*

\* Example: Using `Sign` and `Verify` objects as streams:

\*

\* ```js

\* const {

\* generateKeyPairSync,

\* createSign,

\* createVerify

\* } = await import('crypto');

\*

\* const { privateKey, publicKey } = generateKeyPairSync('ec', {

\* namedCurve: 'sect239k1'

\* });

\*

\* const sign = createSign('SHA256');

\* sign.write('some data to sign');

\* sign.end();

\* const signature = sign.sign(privateKey, 'hex');

\*

\* const verify = createVerify('SHA256');

\* verify.write('some data to sign');

\* verify.end();

\* console.log(verify.verify(publicKey, signature, 'hex'));

\* // Prints: true

\* ```

\*

\* Example: Using the `sign.update()` and `verify.update()` methods:

\*

\* ```js

\* const {

\* generateKeyPairSync,

\* createSign,

\* createVerify

\* } = await import('crypto');

\*

\* const { privateKey, publicKey } = generateKeyPairSync('rsa', {

\* modulusLength: 2048,

\* });

\*

\* const sign = createSign('SHA256');

\* sign.update('some data to sign');

\* sign.end();

\* const signature = sign.sign(privateKey);

\*

\* const verify = createVerify('SHA256');

\* verify.update('some data to sign');

\* verify.end();

\* console.log(verify.verify(publicKey, signature));

\* // Prints: true

\* ```

\* @since v0.1.92

\*/

class Sign extends stream.Writable {

private constructor();

/\*\*

\* Updates the `Sign` content with the given `data`, the encoding of which

\* is given in `inputEncoding`.

\* If `encoding` is not provided, and the `data` is a string, an

\* encoding of `'utf8'` is enforced. If `data` is a `Buffer`, `TypedArray`, or`DataView`, then `inputEncoding` is ignored.

\*

\* This can be called many times with new data as it is streamed.

\* @since v0.1.92

\* @param inputEncoding The `encoding` of the `data` string.

\*/

update(data: BinaryLike): this;

update(data: string, inputEncoding: Encoding): this;

/\*\*

\* Calculates the signature on all the data passed through using either `sign.update()` or `sign.write()`.

\*

\* If `privateKey` is not a `KeyObject`, this function behaves as if`privateKey` had been passed to {@link createPrivateKey}. If it is an

\* object, the following additional properties can be passed:

\*

\* If `outputEncoding` is provided a string is returned; otherwise a `Buffer` is returned.

\*

\* The `Sign` object can not be again used after `sign.sign()` method has been

\* called. Multiple calls to `sign.sign()` will result in an error being thrown.

\* @since v0.1.92

\*/

sign(privateKey: KeyLike | SignKeyObjectInput | SignPrivateKeyInput): Buffer;

sign(privateKey: KeyLike | SignKeyObjectInput | SignPrivateKeyInput, outputFormat: BinaryToTextEncoding): string;

}

/\*\*

\* Creates and returns a `Verify` object that uses the given algorithm.

\* Use {@link getHashes} to obtain an array of names of the available

\* signing algorithms. Optional `options` argument controls the`stream.Writable` behavior.

\*

\* In some cases, a `Verify` instance can be created using the name of a signature

\* algorithm, such as `'RSA-SHA256'`, instead of a digest algorithm. This will use

\* the corresponding digest algorithm. This does not work for all signature

\* algorithms, such as `'ecdsa-with-SHA256'`, so it is best to always use digest

\* algorithm names.

\* @since v0.1.92

\* @param options `stream.Writable` options

\*/

function createVerify(algorithm: string, options?: stream.WritableOptions): Verify;

/\*\*

\* The `Verify` class is a utility for verifying signatures. It can be used in one

\* of two ways:

\*

\* \* As a writable `stream` where written data is used to validate against the

\* supplied signature, or

\* \* Using the `verify.update()` and `verify.verify()` methods to verify

\* the signature.

\*

\* The {@link createVerify} method is used to create `Verify` instances.`Verify` objects are not to be created directly using the `new` keyword.

\*

\* See `Sign` for examples.

\* @since v0.1.92

\*/

class Verify extends stream.Writable {

private constructor();

/\*\*

\* Updates the `Verify` content with the given `data`, the encoding of which

\* is given in `inputEncoding`.

\* If `inputEncoding` is not provided, and the `data` is a string, an

\* encoding of `'utf8'` is enforced. If `data` is a `Buffer`, `TypedArray`, or`DataView`, then `inputEncoding` is ignored.

\*

\* This can be called many times with new data as it is streamed.

\* @since v0.1.92

\* @param inputEncoding The `encoding` of the `data` string.

\*/

update(data: BinaryLike): Verify;

update(data: string, inputEncoding: Encoding): Verify;

/\*\*

\* Verifies the provided data using the given `object` and `signature`.

\*

\* If `object` is not a `KeyObject`, this function behaves as if`object` had been passed to {@link createPublicKey}. If it is an

\* object, the following additional properties can be passed:

\*

\* The `signature` argument is the previously calculated signature for the data, in

\* the `signatureEncoding`.

\* If a `signatureEncoding` is specified, the `signature` is expected to be a

\* string; otherwise `signature` is expected to be a `Buffer`,`TypedArray`, or `DataView`.

\*

\* The `verify` object can not be used again after `verify.verify()` has been

\* called. Multiple calls to `verify.verify()` will result in an error being

\* thrown.

\*

\* Because public keys can be derived from private keys, a private key may

\* be passed instead of a public key.

\* @since v0.1.92

\*/

verify(object: KeyLike | VerifyKeyObjectInput | VerifyPublicKeyInput, signature: NodeJS.ArrayBufferView): boolean;

verify(object: KeyLike | VerifyKeyObjectInput | VerifyPublicKeyInput, signature: string, signature\_format?: BinaryToTextEncoding): boolean;

}

/\*\*

\* Creates a `DiffieHellman` key exchange object using the supplied `prime` and an

\* optional specific `generator`.

\*

\* The `generator` argument can be a number, string, or `Buffer`. If`generator` is not specified, the value `2` is used.

\*

\* If `primeEncoding` is specified, `prime` is expected to be a string; otherwise

\* a `Buffer`, `TypedArray`, or `DataView` is expected.

\*

\* If `generatorEncoding` is specified, `generator` is expected to be a string;

\* otherwise a number, `Buffer`, `TypedArray`, or `DataView` is expected.

\* @since v0.11.12

\* @param primeEncoding The `encoding` of the `prime` string.

\* @param [generator=2]

\* @param generatorEncoding The `encoding` of the `generator` string.

\*/

function createDiffieHellman(primeLength: number, generator?: number | NodeJS.ArrayBufferView): DiffieHellman;

function createDiffieHellman(prime: NodeJS.ArrayBufferView): DiffieHellman;

function createDiffieHellman(prime: string, primeEncoding: BinaryToTextEncoding): DiffieHellman;

function createDiffieHellman(prime: string, primeEncoding: BinaryToTextEncoding, generator: number | NodeJS.ArrayBufferView): DiffieHellman;

function createDiffieHellman(prime: string, primeEncoding: BinaryToTextEncoding, generator: string, generatorEncoding: BinaryToTextEncoding): DiffieHellman;

/\*\*

\* The `DiffieHellman` class is a utility for creating Diffie-Hellman key

\* exchanges.

\*

\* Instances of the `DiffieHellman` class can be created using the {@link createDiffieHellman} function.

\*

\* ```js

\* import assert from 'assert';

\*

\* const {

\* createDiffieHellman

\* } = await import('crypto');

\*

\* // Generate Alice's keys...

\* const alice = createDiffieHellman(2048);

\* const aliceKey = alice.generateKeys();

\*

\* // Generate Bob's keys...

\* const bob = createDiffieHellman(alice.getPrime(), alice.getGenerator());

\* const bobKey = bob.generateKeys();

\*

\* // Exchange and generate the secret...

\* const aliceSecret = alice.computeSecret(bobKey);

\* const bobSecret = bob.computeSecret(aliceKey);

\*

\* // OK

\* assert.strictEqual(aliceSecret.toString('hex'), bobSecret.toString('hex'));

\* ```

\* @since v0.5.0

\*/

class DiffieHellman {

private constructor();

/\*\*

\* Generates private and public Diffie-Hellman key values, and returns

\* the public key in the specified `encoding`. This key should be

\* transferred to the other party.

\* If `encoding` is provided a string is returned; otherwise a `Buffer` is returned.

\* @since v0.5.0

\* @param encoding The `encoding` of the return value.

\*/

generateKeys(): Buffer;

generateKeys(encoding: BinaryToTextEncoding): string;

/\*\*

\* Computes the shared secret using `otherPublicKey` as the other

\* party's public key and returns the computed shared secret. The supplied

\* key is interpreted using the specified `inputEncoding`, and secret is

\* encoded using specified `outputEncoding`.

\* If the `inputEncoding` is not

\* provided, `otherPublicKey` is expected to be a `Buffer`,`TypedArray`, or `DataView`.

\*

\* If `outputEncoding` is given a string is returned; otherwise, a `Buffer` is returned.

\* @since v0.5.0

\* @param inputEncoding The `encoding` of an `otherPublicKey` string.

\* @param outputEncoding The `encoding` of the return value.

\*/

computeSecret(otherPublicKey: NodeJS.ArrayBufferView): Buffer;

computeSecret(otherPublicKey: string, inputEncoding: BinaryToTextEncoding): Buffer;

computeSecret(otherPublicKey: NodeJS.ArrayBufferView, outputEncoding: BinaryToTextEncoding): string;

computeSecret(otherPublicKey: string, inputEncoding: BinaryToTextEncoding, outputEncoding: BinaryToTextEncoding): string;

/\*\*

\* Returns the Diffie-Hellman prime in the specified `encoding`.

\* If `encoding` is provided a string is

\* returned; otherwise a `Buffer` is returned.

\* @since v0.5.0

\* @param encoding The `encoding` of the return value.

\*/

getPrime(): Buffer;

getPrime(encoding: BinaryToTextEncoding): string;

/\*\*

\* Returns the Diffie-Hellman generator in the specified `encoding`.

\* If `encoding` is provided a string is

\* returned; otherwise a `Buffer` is returned.

\* @since v0.5.0

\* @param encoding The `encoding` of the return value.

\*/

getGenerator(): Buffer;

getGenerator(encoding: BinaryToTextEncoding): string;

/\*\*

\* Returns the Diffie-Hellman public key in the specified `encoding`.

\* If `encoding` is provided a

\* string is returned; otherwise a `Buffer` is returned.

\* @since v0.5.0

\* @param encoding The `encoding` of the return value.

\*/

getPublicKey(): Buffer;

getPublicKey(encoding: BinaryToTextEncoding): string;

/\*\*

\* Returns the Diffie-Hellman private key in the specified `encoding`.

\* If `encoding` is provided a

\* string is returned; otherwise a `Buffer` is returned.

\* @since v0.5.0

\* @param encoding The `encoding` of the return value.

\*/

getPrivateKey(): Buffer;

getPrivateKey(encoding: BinaryToTextEncoding): string;

/\*\*

\* Sets the Diffie-Hellman public key. If the `encoding` argument is provided,`publicKey` is expected

\* to be a string. If no `encoding` is provided, `publicKey` is expected

\* to be a `Buffer`, `TypedArray`, or `DataView`.

\* @since v0.5.0

\* @param encoding The `encoding` of the `publicKey` string.

\*/

setPublicKey(publicKey: NodeJS.ArrayBufferView): void;

setPublicKey(publicKey: string, encoding: BufferEncoding): void;

/\*\*

\* Sets the Diffie-Hellman private key. If the `encoding` argument is provided,`privateKey` is expected

\* to be a string. If no `encoding` is provided, `privateKey` is expected

\* to be a `Buffer`, `TypedArray`, or `DataView`.

\* @since v0.5.0

\* @param encoding The `encoding` of the `privateKey` string.

\*/

setPrivateKey(privateKey: NodeJS.ArrayBufferView): void;

setPrivateKey(privateKey: string, encoding: BufferEncoding): void;

/\*\*

\* A bit field containing any warnings and/or errors resulting from a check

\* performed during initialization of the `DiffieHellman` object.

\*

\* The following values are valid for this property (as defined in `constants`module):

\*

\* \* `DH\_CHECK\_P\_NOT\_SAFE\_PRIME`

\* \* `DH\_CHECK\_P\_NOT\_PRIME`

\* \* `DH\_UNABLE\_TO\_CHECK\_GENERATOR`

\* \* `DH\_NOT\_SUITABLE\_GENERATOR`

\* @since v0.11.12

\*/

verifyError: number;

}

/\*\*

\* Creates a predefined `DiffieHellmanGroup` key exchange object. The

\* supported groups are: `'modp1'`, `'modp2'`, `'modp5'` (defined in [RFC 2412](https://www.rfc-editor.org/rfc/rfc2412.txt), but see `Caveats`) and `'modp14'`, `'modp15'`,`'modp16'`, `'modp17'`,

\* `'modp18'` (defined in [RFC 3526](https://www.rfc-editor.org/rfc/rfc3526.txt)). The

\* returned object mimics the interface of objects created by {@link createDiffieHellman}, but will not allow changing

\* the keys (with `diffieHellman.setPublicKey()`, for example). The

\* advantage of using this method is that the parties do not have to

\* generate nor exchange a group modulus beforehand, saving both processor

\* and communication time.

\*

\* Example (obtaining a shared secret):

\*

\* ```js

\* const {

\* getDiffieHellman

\* } = await import('crypto');

\* const alice = getDiffieHellman('modp14');

\* const bob = getDiffieHellman('modp14');

\*

\* alice.generateKeys();

\* bob.generateKeys();

\*

\* const aliceSecret = alice.computeSecret(bob.getPublicKey(), null, 'hex');

\* const bobSecret = bob.computeSecret(alice.getPublicKey(), null, 'hex');

\*

\* // aliceSecret and bobSecret should be the same

\* console.log(aliceSecret === bobSecret);

\* ```

\* @since v0.7.5

\*/

function getDiffieHellman(groupName: string): DiffieHellman;

/\*\*

\* Provides an asynchronous Password-Based Key Derivation Function 2 (PBKDF2)

\* implementation. A selected HMAC digest algorithm specified by `digest` is

\* applied to derive a key of the requested byte length (`keylen`) from the`password`, `salt` and `iterations`.

\*

\* The supplied `callback` function is called with two arguments: `err` and`derivedKey`. If an error occurs while deriving the key, `err` will be set;

\* otherwise `err` will be `null`. By default, the successfully generated`derivedKey` will be passed to the callback as a `Buffer`. An error will be

\* thrown if any of the input arguments specify invalid values or types.

\*

\* If `digest` is `null`, `'sha1'` will be used. This behavior is deprecated,

\* please specify a `digest` explicitly.

\*

\* The `iterations` argument must be a number set as high as possible. The

\* higher the number of iterations, the more secure the derived key will be,

\* but will take a longer amount of time to complete.

\*

\* The `salt` should be as unique as possible. It is recommended that a salt is

\* random and at least 16 bytes long. See [NIST SP 800-132](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf) for details.

\*

\* When passing strings for `password` or `salt`, please consider `caveats when using strings as inputs to cryptographic APIs`.

\*

\* ```js

\* const {

\* pbkdf2

\* } = await import('crypto');

\*

\* pbkdf2('secret', 'salt', 100000, 64, 'sha512', (err, derivedKey) => {

\* if (err) throw err;

\* console.log(derivedKey.toString('hex')); // '3745e48...08d59ae'

\* });

\* ```

\*

\* The `crypto.DEFAULT\_ENCODING` property can be used to change the way the`derivedKey` is passed to the callback. This property, however, has been

\* deprecated and use should be avoided.

\*

\* ```js

\* import crypto from 'crypto';

\* crypto.DEFAULT\_ENCODING = 'hex';

\* crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', (err, derivedKey) => {

\* if (err) throw err;

\* console.log(derivedKey); // '3745e48...aa39b34'

\* });

\* ```

\*

\* An array of supported digest functions can be retrieved using {@link getHashes}.

\*

\* This API uses libuv's threadpool, which can have surprising and

\* negative performance implications for some applications; see the `UV\_THREADPOOL\_SIZE` documentation for more information.

\* @since v0.5.5

\*/

function pbkdf2(password: BinaryLike, salt: BinaryLike, iterations: number, keylen: number, digest: string, callback: (err: Error | null, derivedKey: Buffer) => void): void;

/\*\*

\* Provides a synchronous Password-Based Key Derivation Function 2 (PBKDF2)

\* implementation. A selected HMAC digest algorithm specified by `digest` is

\* applied to derive a key of the requested byte length (`keylen`) from the`password`, `salt` and `iterations`.

\*

\* If an error occurs an `Error` will be thrown, otherwise the derived key will be

\* returned as a `Buffer`.

\*

\* If `digest` is `null`, `'sha1'` will be used. This behavior is deprecated,

\* please specify a `digest` explicitly.

\*

\* The `iterations` argument must be a number set as high as possible. The

\* higher the number of iterations, the more secure the derived key will be,

\* but will take a longer amount of time to complete.

\*

\* The `salt` should be as unique as possible. It is recommended that a salt is

\* random and at least 16 bytes long. See [NIST SP 800-132](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf) for details.

\*

\* When passing strings for `password` or `salt`, please consider `caveats when using strings as inputs to cryptographic APIs`.

\*

\* ```js

\* const {

\* pbkdf2Sync

\* } = await import('crypto');

\*

\* const key = pbkdf2Sync('secret', 'salt', 100000, 64, 'sha512');

\* console.log(key.toString('hex')); // '3745e48...08d59ae'

\* ```

\*

\* The `crypto.DEFAULT\_ENCODING` property may be used to change the way the`derivedKey` is returned. This property, however, is deprecated and use

\* should be avoided.

\*

\* ```js

\* import crypto from 'crypto';

\* crypto.DEFAULT\_ENCODING = 'hex';

\* const key = crypto.pbkdf2Sync('secret', 'salt', 100000, 512, 'sha512');

\* console.log(key); // '3745e48...aa39b34'

\* ```

\*

\* An array of supported digest functions can be retrieved using {@link getHashes}.

\* @since v0.9.3

\*/

function pbkdf2Sync(password: BinaryLike, salt: BinaryLike, iterations: number, keylen: number, digest: string): Buffer;

/\*\*

\* Generates cryptographically strong pseudorandom data. The `size` argument

\* is a number indicating the number of bytes to generate.

\*

\* If a `callback` function is provided, the bytes are generated asynchronously

\* and the `callback` function is invoked with two arguments: `err` and `buf`.

\* If an error occurs, `err` will be an `Error` object; otherwise it is `null`. The`buf` argument is a `Buffer` containing the generated bytes.

\*

\* ```js

\* // Asynchronous

\* const {

\* randomBytes

\* } = await import('crypto');

\*

\* randomBytes(256, (err, buf) => {

\* if (err) throw err;

\* console.log(`${buf.length} bytes of random data: ${buf.toString('hex')}`);

\* });

\* ```

\*

\* If the `callback` function is not provided, the random bytes are generated

\* synchronously and returned as a `Buffer`. An error will be thrown if

\* there is a problem generating the bytes.

\*

\* ```js

\* // Synchronous

\* const {

\* randomBytes

\* } = await import('crypto');

\*

\* const buf = randomBytes(256);

\* console.log(

\* `${buf.length} bytes of random data: ${buf.toString('hex')}`);

\* ```

\*

\* The `crypto.randomBytes()` method will not complete until there is

\* sufficient entropy available.

\* This should normally never take longer than a few milliseconds. The only time

\* when generating the random bytes may conceivably block for a longer period of

\* time is right after boot, when the whole system is still low on entropy.

\*

\* This API uses libuv's threadpool, which can have surprising and

\* negative performance implications for some applications; see the `UV\_THREADPOOL\_SIZE` documentation for more information.

\*

\* The asynchronous version of `crypto.randomBytes()` is carried out in a single

\* threadpool request. To minimize threadpool task length variation, partition

\* large `randomBytes` requests when doing so as part of fulfilling a client

\* request.

\* @since v0.5.8

\* @param size The number of bytes to generate. The `size` must not be larger than `2\*\*31 - 1`.

\* @return if the `callback` function is not provided.

\*/

function randomBytes(size: number): Buffer;

function randomBytes(size: number, callback: (err: Error | null, buf: Buffer) => void): void;

function pseudoRandomBytes(size: number): Buffer;

function pseudoRandomBytes(size: number, callback: (err: Error | null, buf: Buffer) => void): void;

/\*\*

\* Return a random integer `n` such that `min <= n < max`. This

\* implementation avoids [modulo bias](https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates\_shuffle#Modulo\_bias).

\*

\* The range (`max - min`) must be less than 248. `min` and `max` must

\* be [safe integers](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Number/isSafeInteger).

\*

\* If the `callback` function is not provided, the random integer is

\* generated synchronously.

\*

\* ```js

\* // Asynchronous

\* const {

\* randomInt

\* } = await import('crypto');

\*

\* randomInt(3, (err, n) => {

\* if (err) throw err;

\* console.log(`Random number chosen from (0, 1, 2): ${n}`);

\* });

\* ```

\*

\* ```js

\* // Synchronous

\* const {

\* randomInt

\* } = await import('crypto');

\*

\* const n = randomInt(3);

\* console.log(`Random number chosen from (0, 1, 2): ${n}`);

\* ```

\*

\* ```js

\* // With `min` argument

\* const {

\* randomInt

\* } = await import('crypto');

\*

\* const n = randomInt(1, 7);

\* console.log(`The dice rolled: ${n}`);

\* ```

\* @since v14.10.0, v12.19.0

\* @param [min=0] Start of random range (inclusive).

\* @param max End of random range (exclusive).

\* @param callback `function(err, n) {}`.

\*/

function randomInt(max: number): number;

function randomInt(min: number, max: number): number;

function randomInt(max: number, callback: (err: Error | null, value: number) => void): void;

function randomInt(min: number, max: number, callback: (err: Error | null, value: number) => void): void;

/\*\*

\* Synchronous version of {@link randomFill}.

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const { randomFillSync } = await import('crypto');

\*

\* const buf = Buffer.alloc(10);

\* console.log(randomFillSync(buf).toString('hex'));

\*

\* randomFillSync(buf, 5);

\* console.log(buf.toString('hex'));

\*

\* // The above is equivalent to the following:

\* randomFillSync(buf, 5, 5);

\* console.log(buf.toString('hex'));

\* ```

\*

\* Any `ArrayBuffer`, `TypedArray` or `DataView` instance may be passed as`buffer`.

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const { randomFillSync } = await import('crypto');

\*

\* const a = new Uint32Array(10);

\* console.log(Buffer.from(randomFillSync(a).buffer,

\* a.byteOffset, a.byteLength).toString('hex'));

\*

\* const b = new DataView(new ArrayBuffer(10));

\* console.log(Buffer.from(randomFillSync(b).buffer,

\* b.byteOffset, b.byteLength).toString('hex'));

\*

\* const c = new ArrayBuffer(10);

\* console.log(Buffer.from(randomFillSync(c)).toString('hex'));

\* ```

\* @since v7.10.0, v6.13.0

\* @param buffer Must be supplied. The size of the provided `buffer` must not be larger than `2\*\*31 - 1`.

\* @param [offset=0]

\* @param [size=buffer.length - offset]

\* @return The object passed as `buffer` argument.

\*/

function randomFillSync<T extends NodeJS.ArrayBufferView>(buffer: T, offset?: number, size?: number): T;

/\*\*

\* This function is similar to {@link randomBytes} but requires the first

\* argument to be a `Buffer` that will be filled. It also

\* requires that a callback is passed in.

\*

\* If the `callback` function is not provided, an error will be thrown.

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const { randomFill } = await import('crypto');

\*

\* const buf = Buffer.alloc(10);

\* randomFill(buf, (err, buf) => {

\* if (err) throw err;

\* console.log(buf.toString('hex'));

\* });

\*

\* randomFill(buf, 5, (err, buf) => {

\* if (err) throw err;

\* console.log(buf.toString('hex'));

\* });

\*

\* // The above is equivalent to the following:

\* randomFill(buf, 5, 5, (err, buf) => {

\* if (err) throw err;

\* console.log(buf.toString('hex'));

\* });

\* ```

\*

\* Any `ArrayBuffer`, `TypedArray`, or `DataView` instance may be passed as`buffer`.

\*

\* While this includes instances of `Float32Array` and `Float64Array`, this

\* function should not be used to generate random floating-point numbers. The

\* result may contain `+Infinity`, `-Infinity`, and `NaN`, and even if the array

\* contains finite numbers only, they are not drawn from a uniform random

\* distribution and have no meaningful lower or upper bounds.

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const { randomFill } = await import('crypto');

\*

\* const a = new Uint32Array(10);

\* randomFill(a, (err, buf) => {

\* if (err) throw err;

\* console.log(Buffer.from(buf.buffer, buf.byteOffset, buf.byteLength)

\* .toString('hex'));

\* });

\*

\* const b = new DataView(new ArrayBuffer(10));

\* randomFill(b, (err, buf) => {

\* if (err) throw err;

\* console.log(Buffer.from(buf.buffer, buf.byteOffset, buf.byteLength)

\* .toString('hex'));

\* });

\*

\* const c = new ArrayBuffer(10);

\* randomFill(c, (err, buf) => {

\* if (err) throw err;

\* console.log(Buffer.from(buf).toString('hex'));

\* });

\* ```

\*

\* This API uses libuv's threadpool, which can have surprising and

\* negative performance implications for some applications; see the `UV\_THREADPOOL\_SIZE` documentation for more information.

\*

\* The asynchronous version of `crypto.randomFill()` is carried out in a single

\* threadpool request. To minimize threadpool task length variation, partition

\* large `randomFill` requests when doing so as part of fulfilling a client

\* request.

\* @since v7.10.0, v6.13.0

\* @param buffer Must be supplied. The size of the provided `buffer` must not be larger than `2\*\*31 - 1`.

\* @param [offset=0]

\* @param [size=buffer.length - offset]

\* @param callback `function(err, buf) {}`.

\*/

function randomFill<T extends NodeJS.ArrayBufferView>(buffer: T, callback: (err: Error | null, buf: T) => void): void;

function randomFill<T extends NodeJS.ArrayBufferView>(buffer: T, offset: number, callback: (err: Error | null, buf: T) => void): void;

function randomFill<T extends NodeJS.ArrayBufferView>(buffer: T, offset: number, size: number, callback: (err: Error | null, buf: T) => void): void;

interface ScryptOptions {

cost?: number | undefined;

blockSize?: number | undefined;

parallelization?: number | undefined;

N?: number | undefined;

r?: number | undefined;

p?: number | undefined;

maxmem?: number | undefined;

}

/\*\*

\* Provides an asynchronous [scrypt](https://en.wikipedia.org/wiki/Scrypt) implementation. Scrypt is a password-based

\* key derivation function that is designed to be expensive computationally and

\* memory-wise in order to make brute-force attacks unrewarding.

\*

\* The `salt` should be as unique as possible. It is recommended that a salt is

\* random and at least 16 bytes long. See [NIST SP 800-132](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf) for details.

\*

\* When passing strings for `password` or `salt`, please consider `caveats when using strings as inputs to cryptographic APIs`.

\*

\* The `callback` function is called with two arguments: `err` and `derivedKey`.`err` is an exception object when key derivation fails, otherwise `err` is`null`. `derivedKey` is passed to the

\* callback as a `Buffer`.

\*

\* An exception is thrown when any of the input arguments specify invalid values

\* or types.

\*

\* ```js

\* const {

\* scrypt

\* } = await import('crypto');

\*

\* // Using the factory defaults.

\* scrypt('password', 'salt', 64, (err, derivedKey) => {

\* if (err) throw err;

\* console.log(derivedKey.toString('hex')); // '3745e48...08d59ae'

\* });

\* // Using a custom N parameter. Must be a power of two.

\* scrypt('password', 'salt', 64, { N: 1024 }, (err, derivedKey) => {

\* if (err) throw err;

\* console.log(derivedKey.toString('hex')); // '3745e48...aa39b34'

\* });

\* ```

\* @since v10.5.0

\*/

function scrypt(password: BinaryLike, salt: BinaryLike, keylen: number, callback: (err: Error | null, derivedKey: Buffer) => void): void;

function scrypt(password: BinaryLike, salt: BinaryLike, keylen: number, options: ScryptOptions, callback: (err: Error | null, derivedKey: Buffer) => void): void;

/\*\*

\* Provides a synchronous [scrypt](https://en.wikipedia.org/wiki/Scrypt) implementation. Scrypt is a password-based

\* key derivation function that is designed to be expensive computationally and

\* memory-wise in order to make brute-force attacks unrewarding.

\*

\* The `salt` should be as unique as possible. It is recommended that a salt is

\* random and at least 16 bytes long. See [NIST SP 800-132](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf) for details.

\*

\* When passing strings for `password` or `salt`, please consider `caveats when using strings as inputs to cryptographic APIs`.

\*

\* An exception is thrown when key derivation fails, otherwise the derived key is

\* returned as a `Buffer`.

\*

\* An exception is thrown when any of the input arguments specify invalid values

\* or types.

\*

\* ```js

\* const {

\* scryptSync

\* } = await import('crypto');

\* // Using the factory defaults.

\*

\* const key1 = scryptSync('password', 'salt', 64);

\* console.log(key1.toString('hex')); // '3745e48...08d59ae'

\* // Using a custom N parameter. Must be a power of two.

\* const key2 = scryptSync('password', 'salt', 64, { N: 1024 });

\* console.log(key2.toString('hex')); // '3745e48...aa39b34'

\* ```

\* @since v10.5.0

\*/

function scryptSync(password: BinaryLike, salt: BinaryLike, keylen: number, options?: ScryptOptions): Buffer;

interface RsaPublicKey {

key: KeyLike;

padding?: number | undefined;

}

interface RsaPrivateKey {

key: KeyLike;

passphrase?: string | undefined;

/\*\*

\* @default 'sha1'

\*/

oaepHash?: string | undefined;

oaepLabel?: NodeJS.TypedArray | undefined;

padding?: number | undefined;

}

/\*\*

\* Encrypts the content of `buffer` with `key` and returns a new `Buffer` with encrypted content. The returned data can be decrypted using

\* the corresponding private key, for example using {@link privateDecrypt}.

\*

\* If `key` is not a `KeyObject`, this function behaves as if`key` had been passed to {@link createPublicKey}. If it is an

\* object, the `padding` property can be passed. Otherwise, this function uses`RSA\_PKCS1\_OAEP\_PADDING`.

\*

\* Because RSA public keys can be derived from private keys, a private key may

\* be passed instead of a public key.

\* @since v0.11.14

\*/

function publicEncrypt(key: RsaPublicKey | RsaPrivateKey | KeyLike, buffer: NodeJS.ArrayBufferView): Buffer;

/\*\*

\* Decrypts `buffer` with `key`.`buffer` was previously encrypted using

\* the corresponding private key, for example using {@link privateEncrypt}.

\*

\* If `key` is not a `KeyObject`, this function behaves as if`key` had been passed to {@link createPublicKey}. If it is an

\* object, the `padding` property can be passed. Otherwise, this function uses`RSA\_PKCS1\_PADDING`.

\*

\* Because RSA public keys can be derived from private keys, a private key may

\* be passed instead of a public key.

\* @since v1.1.0

\*/

function publicDecrypt(key: RsaPublicKey | RsaPrivateKey | KeyLike, buffer: NodeJS.ArrayBufferView): Buffer;

/\*\*

\* Decrypts `buffer` with `privateKey`. `buffer` was previously encrypted using

\* the corresponding public key, for example using {@link publicEncrypt}.

\*

\* If `privateKey` is not a `KeyObject`, this function behaves as if`privateKey` had been passed to {@link createPrivateKey}. If it is an

\* object, the `padding` property can be passed. Otherwise, this function uses`RSA\_PKCS1\_OAEP\_PADDING`.

\* @since v0.11.14

\*/

function privateDecrypt(privateKey: RsaPrivateKey | KeyLike, buffer: NodeJS.ArrayBufferView): Buffer;

/\*\*

\* Encrypts `buffer` with `privateKey`. The returned data can be decrypted using

\* the corresponding public key, for example using {@link publicDecrypt}.

\*

\* If `privateKey` is not a `KeyObject`, this function behaves as if`privateKey` had been passed to {@link createPrivateKey}. If it is an

\* object, the `padding` property can be passed. Otherwise, this function uses`RSA\_PKCS1\_PADDING`.

\* @since v1.1.0

\*/

function privateEncrypt(privateKey: RsaPrivateKey | KeyLike, buffer: NodeJS.ArrayBufferView): Buffer;

/\*\*

\* ```js

\* const {

\* getCiphers

\* } = await import('crypto');

\*

\* console.log(getCiphers()); // ['aes-128-cbc', 'aes-128-ccm', ...]

\* ```

\* @since v0.9.3

\* @return An array with the names of the supported cipher algorithms.

\*/

function getCiphers(): string[];

/\*\*

\* ```js

\* const {

\* getCurves

\* } = await import('crypto');

\*

\* console.log(getCurves()); // ['Oakley-EC2N-3', 'Oakley-EC2N-4', ...]

\* ```

\* @since v2.3.0

\* @return An array with the names of the supported elliptic curves.

\*/

function getCurves(): string[];

/\*\*

\* @since v10.0.0

\* @return `1` if and only if a FIPS compliant crypto provider is currently in use, `0` otherwise. A future semver-major release may change the return type of this API to a {boolean}.

\*/

function getFips(): 1 | 0;

/\*\*

\* ```js

\* const {

\* getHashes

\* } = await import('crypto');

\*

\* console.log(getHashes()); // ['DSA', 'DSA-SHA', 'DSA-SHA1', ...]

\* ```

\* @since v0.9.3

\* @return An array of the names of the supported hash algorithms, such as `'RSA-SHA256'`. Hash algorithms are also called "digest" algorithms.

\*/

function getHashes(): string[];

/\*\*

\* The `ECDH` class is a utility for creating Elliptic Curve Diffie-Hellman (ECDH)

\* key exchanges.

\*

\* Instances of the `ECDH` class can be created using the {@link createECDH} function.

\*

\* ```js

\* import assert from 'assert';

\*

\* const {

\* createECDH

\* } = await import('crypto');

\*

\* // Generate Alice's keys...

\* const alice = createECDH('secp521r1');

\* const aliceKey = alice.generateKeys();

\*

\* // Generate Bob's keys...

\* const bob = createECDH('secp521r1');

\* const bobKey = bob.generateKeys();

\*

\* // Exchange and generate the secret...

\* const aliceSecret = alice.computeSecret(bobKey);

\* const bobSecret = bob.computeSecret(aliceKey);

\*

\* assert.strictEqual(aliceSecret.toString('hex'), bobSecret.toString('hex'));

\* // OK

\* ```

\* @since v0.11.14

\*/

class ECDH {

private constructor();

/\*\*

\* Converts the EC Diffie-Hellman public key specified by `key` and `curve` to the

\* format specified by `format`. The `format` argument specifies point encoding

\* and can be `'compressed'`, `'uncompressed'` or `'hybrid'`. The supplied key is

\* interpreted using the specified `inputEncoding`, and the returned key is encoded

\* using the specified `outputEncoding`.

\*

\* Use {@link getCurves} to obtain a list of available curve names.

\* On recent OpenSSL releases, `openssl ecparam -list\_curves` will also display

\* the name and description of each available elliptic curve.

\*

\* If `format` is not specified the point will be returned in `'uncompressed'`format.

\*

\* If the `inputEncoding` is not provided, `key` is expected to be a `Buffer`,`TypedArray`, or `DataView`.

\*

\* Example (uncompressing a key):

\*

\* ```js

\* const {

\* createECDH,

\* ECDH

\* } = await import('crypto');

\*

\* const ecdh = createECDH('secp256k1');

\* ecdh.generateKeys();

\*

\* const compressedKey = ecdh.getPublicKey('hex', 'compressed');

\*

\* const uncompressedKey = ECDH.convertKey(compressedKey,

\* 'secp256k1',

\* 'hex',

\* 'hex',

\* 'uncompressed');

\*

\* // The converted key and the uncompressed public key should be the same

\* console.log(uncompressedKey === ecdh.getPublicKey('hex'));

\* ```

\* @since v10.0.0

\* @param inputEncoding The `encoding` of the `key` string.

\* @param outputEncoding The `encoding` of the return value.

\* @param [format='uncompressed']

\*/

static convertKey(

key: BinaryLike,

curve: string,

inputEncoding?: BinaryToTextEncoding,

outputEncoding?: 'latin1' | 'hex' | 'base64' | 'base64url',

format?: 'uncompressed' | 'compressed' | 'hybrid'

): Buffer | string;

/\*\*

\* Generates private and public EC Diffie-Hellman key values, and returns

\* the public key in the specified `format` and `encoding`. This key should be

\* transferred to the other party.

\*

\* The `format` argument specifies point encoding and can be `'compressed'` or`'uncompressed'`. If `format` is not specified, the point will be returned in`'uncompressed'` format.

\*

\* If `encoding` is provided a string is returned; otherwise a `Buffer` is returned.

\* @since v0.11.14

\* @param encoding The `encoding` of the return value.

\* @param [format='uncompressed']

\*/

generateKeys(): Buffer;

generateKeys(encoding: BinaryToTextEncoding, format?: ECDHKeyFormat): string;

/\*\*

\* Computes the shared secret using `otherPublicKey` as the other

\* party's public key and returns the computed shared secret. The supplied

\* key is interpreted using specified `inputEncoding`, and the returned secret

\* is encoded using the specified `outputEncoding`.

\* If the `inputEncoding` is not

\* provided, `otherPublicKey` is expected to be a `Buffer`, `TypedArray`, or`DataView`.

\*

\* If `outputEncoding` is given a string will be returned; otherwise a `Buffer` is returned.

\*

\* `ecdh.computeSecret` will throw an`ERR\_CRYPTO\_ECDH\_INVALID\_PUBLIC\_KEY` error when `otherPublicKey`lies outside of the elliptic curve. Since `otherPublicKey` is

\* usually supplied from a remote user over an insecure network,

\* be sure to handle this exception accordingly.

\* @since v0.11.14

\* @param inputEncoding The `encoding` of the `otherPublicKey` string.

\* @param outputEncoding The `encoding` of the return value.

\*/

computeSecret(otherPublicKey: NodeJS.ArrayBufferView): Buffer;

computeSecret(otherPublicKey: string, inputEncoding: BinaryToTextEncoding): Buffer;

computeSecret(otherPublicKey: NodeJS.ArrayBufferView, outputEncoding: BinaryToTextEncoding): string;

computeSecret(otherPublicKey: string, inputEncoding: BinaryToTextEncoding, outputEncoding: BinaryToTextEncoding): string;

/\*\*

\* If `encoding` is specified, a string is returned; otherwise a `Buffer` is

\* returned.

\* @since v0.11.14

\* @param encoding The `encoding` of the return value.

\* @return The EC Diffie-Hellman in the specified `encoding`.

\*/

getPrivateKey(): Buffer;

getPrivateKey(encoding: BinaryToTextEncoding): string;

/\*\*

\* The `format` argument specifies point encoding and can be `'compressed'` or`'uncompressed'`. If `format` is not specified the point will be returned in`'uncompressed'` format.

\*

\* If `encoding` is specified, a string is returned; otherwise a `Buffer` is

\* returned.

\* @since v0.11.14

\* @param encoding The `encoding` of the return value.

\* @param [format='uncompressed']

\* @return The EC Diffie-Hellman public key in the specified `encoding` and `format`.

\*/

getPublicKey(): Buffer;

getPublicKey(encoding: BinaryToTextEncoding, format?: ECDHKeyFormat): string;

/\*\*

\* Sets the EC Diffie-Hellman private key.

\* If `encoding` is provided, `privateKey` is expected

\* to be a string; otherwise `privateKey` is expected to be a `Buffer`,`TypedArray`, or `DataView`.

\*

\* If `privateKey` is not valid for the curve specified when the `ECDH` object was

\* created, an error is thrown. Upon setting the private key, the associated

\* public point (key) is also generated and set in the `ECDH` object.

\* @since v0.11.14

\* @param encoding The `encoding` of the `privateKey` string.

\*/

setPrivateKey(privateKey: NodeJS.ArrayBufferView): void;

setPrivateKey(privateKey: string, encoding: BinaryToTextEncoding): void;

}

/\*\*

\* Creates an Elliptic Curve Diffie-Hellman (`ECDH`) key exchange object using a

\* predefined curve specified by the `curveName` string. Use {@link getCurves} to obtain a list of available curve names. On recent

\* OpenSSL releases, `openssl ecparam -list\_curves` will also display the name

\* and description of each available elliptic curve.

\* @since v0.11.14

\*/

function createECDH(curveName: string): ECDH;

/\*\*

\* This function is based on a constant-time algorithm.

\* Returns true if `a` is equal to `b`, without leaking timing information that

\* would allow an attacker to guess one of the values. This is suitable for

\* comparing HMAC digests or secret values like authentication cookies or [capability urls](https://www.w3.org/TR/capability-urls/).

\*

\* `a` and `b` must both be `Buffer`s, `TypedArray`s, or `DataView`s, and they

\* must have the same byte length.

\*

\* If at least one of `a` and `b` is a `TypedArray` with more than one byte per

\* entry, such as `Uint16Array`, the result will be computed using the platform

\* byte order.

\*

\* Use of `crypto.timingSafeEqual` does not guarantee that the \_surrounding\_ code

\* is timing-safe. Care should be taken to ensure that the surrounding code does

\* not introduce timing vulnerabilities.

\* @since v6.6.0

\*/

function timingSafeEqual(a: NodeJS.ArrayBufferView, b: NodeJS.ArrayBufferView): boolean;

/\*\* @deprecated since v10.0.0 \*/

const DEFAULT\_ENCODING: BufferEncoding;

type KeyType = 'rsa' | 'rsa-pss' | 'dsa' | 'ec' | 'ed25519' | 'ed448' | 'x25519' | 'x448';

type KeyFormat = 'pem' | 'der';

interface BasePrivateKeyEncodingOptions<T extends KeyFormat> {

format: T;

cipher?: string | undefined;

passphrase?: string | undefined;

}

interface KeyPairKeyObjectResult {

publicKey: KeyObject;

privateKey: KeyObject;

}

interface ED25519KeyPairKeyObjectOptions {}

interface ED448KeyPairKeyObjectOptions {}

interface X25519KeyPairKeyObjectOptions {}

interface X448KeyPairKeyObjectOptions {}

interface ECKeyPairKeyObjectOptions {

/\*\*

\* Name of the curve to use

\*/

namedCurve: string;

}

interface RSAKeyPairKeyObjectOptions {

/\*\*

\* Key size in bits

\*/

modulusLength: number;

/\*\*

\* Public exponent

\* @default 0x10001

\*/

publicExponent?: number | undefined;

}

interface RSAPSSKeyPairKeyObjectOptions {

/\*\*

\* Key size in bits

\*/

modulusLength: number;

/\*\*

\* Public exponent

\* @default 0x10001

\*/

publicExponent?: number | undefined;

/\*\*

\* Name of the message digest

\*/

hashAlgorithm?: string;

/\*\*

\* Name of the message digest used by MGF1

\*/

mgf1HashAlgorithm?: string;

/\*\*

\* Minimal salt length in bytes

\*/

saltLength?: string;

}

interface DSAKeyPairKeyObjectOptions {

/\*\*

\* Key size in bits

\*/

modulusLength: number;

/\*\*

\* Size of q in bits

\*/

divisorLength: number;

}

interface RSAKeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

/\*\*

\* Key size in bits

\*/

modulusLength: number;

/\*\*

\* Public exponent

\* @default 0x10001

\*/

publicExponent?: number | undefined;

publicKeyEncoding: {

type: 'pkcs1' | 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs1' | 'pkcs8';

};

}

interface RSAPSSKeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

/\*\*

\* Key size in bits

\*/

modulusLength: number;

/\*\*

\* Public exponent

\* @default 0x10001

\*/

publicExponent?: number | undefined;

/\*\*

\* Name of the message digest

\*/

hashAlgorithm?: string;

/\*\*

\* Name of the message digest used by MGF1

\*/

mgf1HashAlgorithm?: string;

/\*\*

\* Minimal salt length in bytes

\*/

saltLength?: string;

publicKeyEncoding: {

type: 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs8';

};

}

interface DSAKeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

/\*\*

\* Key size in bits

\*/

modulusLength: number;

/\*\*

\* Size of q in bits

\*/

divisorLength: number;

publicKeyEncoding: {

type: 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs8';

};

}

interface ECKeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

/\*\*

\* Name of the curve to use.

\*/

namedCurve: string;

publicKeyEncoding: {

type: 'pkcs1' | 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'sec1' | 'pkcs8';

};

}

interface ED25519KeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

publicKeyEncoding: {

type: 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs8';

};

}

interface ED448KeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

publicKeyEncoding: {

type: 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs8';

};

}

interface X25519KeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

publicKeyEncoding: {

type: 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs8';

};

}

interface X448KeyPairOptions<PubF extends KeyFormat, PrivF extends KeyFormat> {

publicKeyEncoding: {

type: 'spki';

format: PubF;

};

privateKeyEncoding: BasePrivateKeyEncodingOptions<PrivF> & {

type: 'pkcs8';

};

}

interface KeyPairSyncResult<T1 extends string | Buffer, T2 extends string | Buffer> {

publicKey: T1;

privateKey: T2;

}

/\*\*

\* Generates a new asymmetric key pair of the given `type`. RSA, RSA-PSS, DSA, EC,

\* Ed25519, Ed448, X25519, X448, and DH are currently supported.

\*

\* If a `publicKeyEncoding` or `privateKeyEncoding` was specified, this function

\* behaves as if `keyObject.export()` had been called on its result. Otherwise,

\* the respective part of the key is returned as a `KeyObject`.

\*

\* When encoding public keys, it is recommended to use `'spki'`. When encoding

\* private keys, it is recommended to use `'pkcs8'` with a strong passphrase,

\* and to keep the passphrase confidential.

\*

\* ```js

\* const {

\* generateKeyPairSync

\* } = await import('crypto');

\*

\* const {

\* publicKey,

\* privateKey,

\* } = generateKeyPairSync('rsa', {

\* modulusLength: 4096,

\* publicKeyEncoding: {

\* type: 'spki',

\* format: 'pem'

\* },

\* privateKeyEncoding: {

\* type: 'pkcs8',

\* format: 'pem',

\* cipher: 'aes-256-cbc',

\* passphrase: 'top secret'

\* }

\* });

\* ```

\*

\* The return value `{ publicKey, privateKey }` represents the generated key pair.

\* When PEM encoding was selected, the respective key will be a string, otherwise

\* it will be a buffer containing the data encoded as DER.

\* @since v10.12.0

\* @param type Must be `'rsa'`, `'rsa-pss'`, `'dsa'`, `'ec'`, `'ed25519'`, `'ed448'`, `'x25519'`, `'x448'`, or `'dh'`.

\*/

function generateKeyPairSync(type: 'rsa', options: RSAKeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'rsa', options: RSAKeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'rsa', options: RSAKeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'rsa', options: RSAKeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'rsa', options: RSAKeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'rsa-pss', options: RSAPSSKeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'dsa', options: DSAKeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'dsa', options: DSAKeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'dsa', options: DSAKeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'dsa', options: DSAKeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'dsa', options: DSAKeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'ec', options: ECKeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'ec', options: ECKeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'ec', options: ECKeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'ec', options: ECKeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'ec', options: ECKeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'ed25519', options: ED25519KeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'ed25519', options: ED25519KeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'ed25519', options: ED25519KeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'ed25519', options: ED25519KeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'ed25519', options?: ED25519KeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'ed448', options: ED448KeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'ed448', options: ED448KeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'ed448', options: ED448KeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'ed448', options: ED448KeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'ed448', options?: ED448KeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'x25519', options: X25519KeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'x25519', options: X25519KeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'x25519', options: X25519KeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'x25519', options: X25519KeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'x25519', options?: X25519KeyPairKeyObjectOptions): KeyPairKeyObjectResult;

function generateKeyPairSync(type: 'x448', options: X448KeyPairOptions<'pem', 'pem'>): KeyPairSyncResult<string, string>;

function generateKeyPairSync(type: 'x448', options: X448KeyPairOptions<'pem', 'der'>): KeyPairSyncResult<string, Buffer>;

function generateKeyPairSync(type: 'x448', options: X448KeyPairOptions<'der', 'pem'>): KeyPairSyncResult<Buffer, string>;

function generateKeyPairSync(type: 'x448', options: X448KeyPairOptions<'der', 'der'>): KeyPairSyncResult<Buffer, Buffer>;

function generateKeyPairSync(type: 'x448', options?: X448KeyPairKeyObjectOptions): KeyPairKeyObjectResult;

/\*\*

\* Generates a new asymmetric key pair of the given `type`. RSA, RSA-PSS, DSA, EC,

\* Ed25519, Ed448, X25519, X448, and DH are currently supported.

\*

\* If a `publicKeyEncoding` or `privateKeyEncoding` was specified, this function

\* behaves as if `keyObject.export()` had been called on its result. Otherwise,

\* the respective part of the key is returned as a `KeyObject`.

\*

\* It is recommended to encode public keys as `'spki'` and private keys as`'pkcs8'` with encryption for long-term storage:

\*

\* ```js

\* const {

\* generateKeyPair

\* } = await import('crypto');

\*

\* generateKeyPair('rsa', {

\* modulusLength: 4096,

\* publicKeyEncoding: {

\* type: 'spki',

\* format: 'pem'

\* },

\* privateKeyEncoding: {

\* type: 'pkcs8',

\* format: 'pem',

\* cipher: 'aes-256-cbc',

\* passphrase: 'top secret'

\* }

\* }, (err, publicKey, privateKey) => {

\* // Handle errors and use the generated key pair.

\* });

\* ```

\*

\* On completion, `callback` will be called with `err` set to `undefined` and`publicKey` / `privateKey` representing the generated key pair.

\*

\* If this method is invoked as its `util.promisify()` ed version, it returns

\* a `Promise` for an `Object` with `publicKey` and `privateKey` properties.

\* @since v10.12.0

\* @param type Must be `'rsa'`, `'rsa-pss'`, `'dsa'`, `'ec'`, `'ed25519'`, `'ed448'`, `'x25519'`, `'x448'`, or `'dh'`.

\*/

function generateKeyPair(type: 'rsa', options: RSAKeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'rsa', options: RSAKeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'rsa', options: RSAKeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'rsa', options: RSAKeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'rsa', options: RSAKeyPairKeyObjectOptions, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'rsa-pss', options: RSAPSSKeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'rsa-pss', options: RSAPSSKeyPairKeyObjectOptions, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'dsa', options: DSAKeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'dsa', options: DSAKeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'dsa', options: DSAKeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'dsa', options: DSAKeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'dsa', options: DSAKeyPairKeyObjectOptions, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'ec', options: ECKeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'ec', options: ECKeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'ec', options: ECKeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'ec', options: ECKeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'ec', options: ECKeyPairKeyObjectOptions, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'ed25519', options: ED25519KeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'ed25519', options: ED25519KeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'ed25519', options: ED25519KeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'ed25519', options: ED25519KeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'ed25519', options: ED25519KeyPairKeyObjectOptions | undefined, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'ed448', options: ED448KeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'ed448', options: ED448KeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'ed448', options: ED448KeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'ed448', options: ED448KeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'ed448', options: ED448KeyPairKeyObjectOptions | undefined, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'x25519', options: X25519KeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'x25519', options: X25519KeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'x25519', options: X25519KeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'x25519', options: X25519KeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'x25519', options: X25519KeyPairKeyObjectOptions | undefined, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

function generateKeyPair(type: 'x448', options: X448KeyPairOptions<'pem', 'pem'>, callback: (err: Error | null, publicKey: string, privateKey: string) => void): void;

function generateKeyPair(type: 'x448', options: X448KeyPairOptions<'pem', 'der'>, callback: (err: Error | null, publicKey: string, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'x448', options: X448KeyPairOptions<'der', 'pem'>, callback: (err: Error | null, publicKey: Buffer, privateKey: string) => void): void;

function generateKeyPair(type: 'x448', options: X448KeyPairOptions<'der', 'der'>, callback: (err: Error | null, publicKey: Buffer, privateKey: Buffer) => void): void;

function generateKeyPair(type: 'x448', options: X448KeyPairKeyObjectOptions | undefined, callback: (err: Error | null, publicKey: KeyObject, privateKey: KeyObject) => void): void;

namespace generateKeyPair {

function \_\_promisify\_\_(

type: 'rsa',

options: RSAKeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'rsa',

options: RSAKeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'rsa',

options: RSAKeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'rsa',

options: RSAKeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'rsa', options: RSAKeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'rsa-pss',

options: RSAPSSKeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'rsa-pss',

options: RSAPSSKeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'rsa-pss',

options: RSAPSSKeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'rsa-pss',

options: RSAPSSKeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'rsa-pss', options: RSAPSSKeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'dsa',

options: DSAKeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'dsa',

options: DSAKeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'dsa',

options: DSAKeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'dsa',

options: DSAKeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'dsa', options: DSAKeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'ec',

options: ECKeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'ec',

options: ECKeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'ec',

options: ECKeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'ec',

options: ECKeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'ec', options: ECKeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'ed25519',

options: ED25519KeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'ed25519',

options: ED25519KeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'ed25519',

options: ED25519KeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'ed25519',

options: ED25519KeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'ed25519', options?: ED25519KeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'ed448',

options: ED448KeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'ed448',

options: ED448KeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'ed448',

options: ED448KeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'ed448',

options: ED448KeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'ed448', options?: ED448KeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'x25519',

options: X25519KeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'x25519',

options: X25519KeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'x25519',

options: X25519KeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'x25519',

options: X25519KeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'x25519', options?: X25519KeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

function \_\_promisify\_\_(

type: 'x448',

options: X448KeyPairOptions<'pem', 'pem'>

): Promise<{

publicKey: string;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'x448',

options: X448KeyPairOptions<'pem', 'der'>

): Promise<{

publicKey: string;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(

type: 'x448',

options: X448KeyPairOptions<'der', 'pem'>

): Promise<{

publicKey: Buffer;

privateKey: string;

}>;

function \_\_promisify\_\_(

type: 'x448',

options: X448KeyPairOptions<'der', 'der'>

): Promise<{

publicKey: Buffer;

privateKey: Buffer;

}>;

function \_\_promisify\_\_(type: 'x448', options?: X448KeyPairKeyObjectOptions): Promise<KeyPairKeyObjectResult>;

}

/\*\*

\* Calculates and returns the signature for `data` using the given private key and

\* algorithm. If `algorithm` is `null` or `undefined`, then the algorithm is

\* dependent upon the key type (especially Ed25519 and Ed448).

\*

\* If `key` is not a `KeyObject`, this function behaves as if `key` had been

\* passed to {@link createPrivateKey}. If it is an object, the following

\* additional properties can be passed:

\*

\* If the `callback` function is provided this function uses libuv's threadpool.

\* @since v12.0.0

\*/

function sign(algorithm: string | null | undefined, data: NodeJS.ArrayBufferView, key: KeyLike | SignKeyObjectInput | SignPrivateKeyInput): Buffer;

function sign(

algorithm: string | null | undefined,

data: NodeJS.ArrayBufferView,

key: KeyLike | SignKeyObjectInput | SignPrivateKeyInput,

callback: (error: Error | null, data: Buffer) => void

): void;

/\*\*

\* Verifies the given signature for `data` using the given key and algorithm. If`algorithm` is `null` or `undefined`, then the algorithm is dependent upon the

\* key type (especially Ed25519 and Ed448).

\*

\* If `key` is not a `KeyObject`, this function behaves as if `key` had been

\* passed to {@link createPublicKey}. If it is an object, the following

\* additional properties can be passed:

\*

\* The `signature` argument is the previously calculated signature for the `data`.

\*

\* Because public keys can be derived from private keys, a private key or a public

\* key may be passed for `key`.

\*

\* If the `callback` function is provided this function uses libuv's threadpool.

\* @since v12.0.0

\*/

function verify(algorithm: string | null | undefined, data: NodeJS.ArrayBufferView, key: KeyLike | VerifyKeyObjectInput | VerifyPublicKeyInput, signature: NodeJS.ArrayBufferView): boolean;

function verify(

algorithm: string | null | undefined,

data: NodeJS.ArrayBufferView,

key: KeyLike | VerifyKeyObjectInput | VerifyPublicKeyInput,

signature: NodeJS.ArrayBufferView,

callback: (error: Error | null, result: boolean) => void

): void;

/\*\*

\* Computes the Diffie-Hellman secret based on a `privateKey` and a `publicKey`.

\* Both keys must have the same `asymmetricKeyType`, which must be one of `'dh'`(for Diffie-Hellman), `'ec'` (for ECDH), `'x448'`, or `'x25519'` (for ECDH-ES).

\* @since v13.9.0, v12.17.0

\*/

function diffieHellman(options: { privateKey: KeyObject; publicKey: KeyObject }): Buffer;

type CipherMode = 'cbc' | 'ccm' | 'cfb' | 'ctr' | 'ecb' | 'gcm' | 'ocb' | 'ofb' | 'stream' | 'wrap' | 'xts';

interface CipherInfoOptions {

/\*\*

\* A test key length.

\*/

keyLength?: number | undefined;

/\*\*

\* A test IV length.

\*/

ivLength?: number | undefined;

}

interface CipherInfo {

/\*\*

\* The name of the cipher.

\*/

name: string;

/\*\*

\* The nid of the cipher.

\*/

nid: number;

/\*\*

\* The block size of the cipher in bytes.

\* This property is omitted when mode is 'stream'.

\*/

blockSize?: number | undefined;

/\*\*

\* The expected or default initialization vector length in bytes.

\* This property is omitted if the cipher does not use an initialization vector.

\*/

ivLength?: number | undefined;

/\*\*

\* The expected or default key length in bytes.

\*/

keyLength: number;

/\*\*

\* The cipher mode.

\*/

mode: CipherMode;

}

/\*\*

\* Returns information about a given cipher.

\*

\* Some ciphers accept variable length keys and initialization vectors. By default,

\* the `crypto.getCipherInfo()` method will return the default values for these

\* ciphers. To test if a given key length or iv length is acceptable for given

\* cipher, use the `keyLength` and `ivLength` options. If the given values are

\* unacceptable, `undefined` will be returned.

\* @since v15.0.0

\* @param nameOrNid The name or nid of the cipher to query.

\*/

function getCipherInfo(nameOrNid: string | number, options?: CipherInfoOptions): CipherInfo | undefined;

/\*\*

\* HKDF is a simple key derivation function defined in RFC 5869\. The given `ikm`,`salt` and `info` are used with the `digest` to derive a key of `keylen` bytes.

\*

\* The supplied `callback` function is called with two arguments: `err` and`derivedKey`. If an errors occurs while deriving the key, `err` will be set;

\* otherwise `err` will be `null`. The successfully generated `derivedKey` will

\* be passed to the callback as an [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer). An error will be thrown if any

\* of the input arguments specify invalid values or types.

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const {

\* hkdf

\* } = await import('crypto');

\*

\* hkdf('sha512', 'key', 'salt', 'info', 64, (err, derivedKey) => {

\* if (err) throw err;

\* console.log(Buffer.from(derivedKey).toString('hex')); // '24156e2...5391653'

\* });

\* ```

\* @since v15.0.0

\* @param digest The digest algorithm to use.

\* @param ikm The input keying material. It must be at least one byte in length.

\* @param salt The salt value. Must be provided but can be zero-length.

\* @param info Additional info value. Must be provided but can be zero-length, and cannot be more than 1024 bytes.

\* @param keylen The length of the key to generate. Must be greater than 0. The maximum allowable value is `255` times the number of bytes produced by the selected digest function (e.g. `sha512`

\* generates 64-byte hashes, making the maximum HKDF output 16320 bytes).

\*/

function hkdf(digest: string, irm: BinaryLike | KeyObject, salt: BinaryLike, info: BinaryLike, keylen: number, callback: (err: Error | null, derivedKey: ArrayBuffer) => void): void;

/\*\*

\* Provides a synchronous HKDF key derivation function as defined in RFC 5869\. The

\* given `ikm`, `salt` and `info` are used with the `digest` to derive a key of`keylen` bytes.

\*

\* The successfully generated `derivedKey` will be returned as an [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer).

\*

\* An error will be thrown if any of the input arguments specify invalid values or

\* types, or if the derived key cannot be generated.

\*

\* ```js

\* import { Buffer } from 'buffer';

\* const {

\* hkdfSync

\* } = await import('crypto');

\*

\* const derivedKey = hkdfSync('sha512', 'key', 'salt', 'info', 64);

\* console.log(Buffer.from(derivedKey).toString('hex')); // '24156e2...5391653'

\* ```

\* @since v15.0.0

\* @param digest The digest algorithm to use.

\* @param ikm The input keying material. It must be at least one byte in length.

\* @param salt The salt value. Must be provided but can be zero-length.

\* @param info Additional info value. Must be provided but can be zero-length, and cannot be more than 1024 bytes.

\* @param keylen The length of the key to generate. Must be greater than 0. The maximum allowable value is `255` times the number of bytes produced by the selected digest function (e.g. `sha512`

\* generates 64-byte hashes, making the maximum HKDF output 16320 bytes).

\*/

function hkdfSync(digest: string, ikm: BinaryLike | KeyObject, salt: BinaryLike, info: BinaryLike, keylen: number): ArrayBuffer;

interface SecureHeapUsage {

/\*\*

\* The total allocated secure heap size as specified using the `--secure-heap=n` command-line flag.

\*/

total: number;

/\*\*

\* The minimum allocation from the secure heap as specified using the `--secure-heap-min` command-line flag.

\*/

min: number;

/\*\*

\* The total number of bytes currently allocated from the secure heap.

\*/

used: number;

/\*\*

\* The calculated ratio of `used` to `total` allocated bytes.

\*/

utilization: number;

}

/\*\*

\* @since v15.6.0

\*/

function secureHeapUsed(): SecureHeapUsage;

interface RandomUUIDOptions {

/\*\*

\* By default, to improve performance,

\* Node.js will pre-emptively generate and persistently cache enough

\* random data to generate up to 128 random UUIDs. To generate a UUID

\* without using the cache, set `disableEntropyCache` to `true`.

\*

\* @default `false`

\*/

disableEntropyCache?: boolean | undefined;

}

/\*\*

\* Generates a random [RFC 4122](https://www.rfc-editor.org/rfc/rfc4122.txt) version 4 UUID. The UUID is generated using a

\* cryptographic pseudorandom number generator.

\* @since v15.6.0, v14.17.0

\*/

function randomUUID(options?: RandomUUIDOptions): string;

interface X509CheckOptions {

/\*\*

\* @default 'always'

\*/

subject: 'always' | 'never';

/\*\*

\* @default true

\*/

wildcards: boolean;

/\*\*

\* @default true

\*/

partialWildcards: boolean;

/\*\*

\* @default false

\*/

multiLabelWildcards: boolean;

/\*\*

\* @default false

\*/

singleLabelSubdomains: boolean;

}

/\*\*

\* Encapsulates an X509 certificate and provides read-only access to

\* its information.

\*

\* ```js

\* const { X509Certificate } = await import('crypto');

\*

\* const x509 = new X509Certificate('{... pem encoded cert ...}');

\*

\* console.log(x509.subject);

\* ```

\* @since v15.6.0

\*/

class X509Certificate {

/\*\*

\* Will be \`true\` if this is a Certificate Authority (ca) certificate.

\* @since v15.6.0

\*/

readonly ca: boolean;

/\*\*

\* The SHA-1 fingerprint of this certificate.

\* @since v15.6.0

\*/

readonly fingerprint: string;

/\*\*

\* The SHA-256 fingerprint of this certificate.

\* @since v15.6.0

\*/

readonly fingerprint256: string;

/\*\*

\* The complete subject of this certificate.

\* @since v15.6.0

\*/

readonly subject: string;

/\*\*

\* The subject alternative name specified for this certificate.

\* @since v15.6.0

\*/

readonly subjectAltName: string;

/\*\*

\* The information access content of this certificate.

\* @since v15.6.0

\*/

readonly infoAccess: string;

/\*\*

\* An array detailing the key usages for this certificate.

\* @since v15.6.0

\*/

readonly keyUsage: string[];

/\*\*

\* The issuer identification included in this certificate.

\* @since v15.6.0

\*/

readonly issuer: string;

/\*\*

\* The issuer certificate or `undefined` if the issuer certificate is not

\* available.

\* @since v15.9.0

\*/

readonly issuerCertificate?: X509Certificate | undefined;

/\*\*

\* The public key `KeyObject` for this certificate.

\* @since v15.6.0

\*/

readonly publicKey: KeyObject;

/\*\*

\* A `Buffer` containing the DER encoding of this certificate.

\* @since v15.6.0

\*/

readonly raw: Buffer;

/\*\*

\* The serial number of this certificate.

\* @since v15.6.0

\*/

readonly serialNumber: string;

/\*\*

\* The date/time from which this certificate is considered valid.

\* @since v15.6.0

\*/

readonly validFrom: string;

/\*\*

\* The date/time until which this certificate is considered valid.

\* @since v15.6.0

\*/

readonly validTo: string;

constructor(buffer: BinaryLike);

/\*\*

\* Checks whether the certificate matches the given email address.

\* @since v15.6.0

\* @return Returns `email` if the certificate matches, `undefined` if it does not.

\*/

checkEmail(email: string, options?: X509CheckOptions): string | undefined;

/\*\*

\* Checks whether the certificate matches the given host name.

\* @since v15.6.0

\* @return Returns `name` if the certificate matches, `undefined` if it does not.

\*/

checkHost(name: string, options?: X509CheckOptions): string | undefined;

/\*\*

\* Checks whether the certificate matches the given IP address (IPv4 or IPv6).

\* @since v15.6.0

\* @return Returns `ip` if the certificate matches, `undefined` if it does not.

\*/

checkIP(ip: string, options?: X509CheckOptions): string | undefined;

/\*\*

\* Checks whether this certificate was issued by the given `otherCert`.

\* @since v15.6.0

\*/

checkIssued(otherCert: X509Certificate): boolean;

/\*\*

\* Checks whether the public key for this certificate is consistent with

\* the given private key.

\* @since v15.6.0

\* @param privateKey A private key.

\*/

checkPrivateKey(privateKey: KeyObject): boolean;

/\*\*

\* There is no standard JSON encoding for X509 certificates. The`toJSON()` method returns a string containing the PEM encoded

\* certificate.

\* @since v15.6.0

\*/

toJSON(): string;

/\*\*

\* Returns information about this certificate using the legacy `certificate object` encoding.

\* @since v15.6.0

\*/

toLegacyObject(): PeerCertificate;

/\*\*

\* Returns the PEM-encoded certificate.

\* @since v15.6.0

\*/

toString(): string;

/\*\*

\* Verifies that this certificate was signed by the given public key.

\* Does not perform any other validation checks on the certificate.

\* @since v15.6.0

\* @param publicKey A public key.

\*/

verify(publicKey: KeyObject): boolean;

}

type LargeNumberLike = NodeJS.ArrayBufferView | SharedArrayBuffer | ArrayBuffer | bigint;

interface GeneratePrimeOptions {

add?: LargeNumberLike | undefined;

rem?: LargeNumberLike | undefined;

/\*\*

\* @default false

\*/

safe?: boolean | undefined;

bigint?: boolean | undefined;

}

interface GeneratePrimeOptionsBigInt extends GeneratePrimeOptions {

bigint: true;

}

interface GeneratePrimeOptionsArrayBuffer extends GeneratePrimeOptions {

bigint?: false | undefined;

}

/\*\*

\* Generates a pseudorandom prime of `size` bits.

\*

\* If `options.safe` is `true`, the prime will be a safe prime -- that is,`(prime - 1) / 2` will also be a prime.

\*

\* The `options.add` and `options.rem` parameters can be used to enforce additional

\* requirements, e.g., for Diffie-Hellman:

\*

\* \* If `options.add` and `options.rem` are both set, the prime will satisfy the

\* condition that `prime % add = rem`.

\* \* If only `options.add` is set and `options.safe` is not `true`, the prime will

\* satisfy the condition that `prime % add = 1`.

\* \* If only `options.add` is set and `options.safe` is set to `true`, the prime

\* will instead satisfy the condition that `prime % add = 3`. This is necessary

\* because `prime % add = 1` for `options.add > 2` would contradict the condition

\* enforced by `options.safe`.

\* \* `options.rem` is ignored if `options.add` is not given.

\*

\* Both `options.add` and `options.rem` must be encoded as big-endian sequences

\* if given as an `ArrayBuffer`, `SharedArrayBuffer`, `TypedArray`, `Buffer`, or`DataView`.

\*

\* By default, the prime is encoded as a big-endian sequence of octets

\* in an [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer). If the `bigint` option is `true`, then a

\* [bigint](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/BigInt) is provided.

\* @since v15.8.0

\* @param size The size (in bits) of the prime to generate.

\*/

function generatePrime(size: number, callback: (err: Error | null, prime: ArrayBuffer) => void): void;

function generatePrime(size: number, options: GeneratePrimeOptionsBigInt, callback: (err: Error | null, prime: bigint) => void): void;

function generatePrime(size: number, options: GeneratePrimeOptionsArrayBuffer, callback: (err: Error | null, prime: ArrayBuffer) => void): void;

function generatePrime(size: number, options: GeneratePrimeOptions, callback: (err: Error | null, prime: ArrayBuffer | bigint) => void): void;

/\*\*

\* Generates a pseudorandom prime of `size` bits.

\*

\* If `options.safe` is `true`, the prime will be a safe prime -- that is,`(prime - 1) / 2` will also be a prime.

\*

\* The `options.add` and `options.rem` parameters can be used to enforce additional

\* requirements, e.g., for Diffie-Hellman:

\*

\* \* If `options.add` and `options.rem` are both set, the prime will satisfy the

\* condition that `prime % add = rem`.

\* \* If only `options.add` is set and `options.safe` is not `true`, the prime will

\* satisfy the condition that `prime % add = 1`.

\* \* If only `options.add` is set and `options.safe` is set to `true`, the prime

\* will instead satisfy the condition that `prime % add = 3`. This is necessary

\* because `prime % add = 1` for `options.add > 2` would contradict the condition

\* enforced by `options.safe`.

\* \* `options.rem` is ignored if `options.add` is not given.

\*

\* Both `options.add` and `options.rem` must be encoded as big-endian sequences

\* if given as an `ArrayBuffer`, `SharedArrayBuffer`, `TypedArray`, `Buffer`, or`DataView`.

\*

\* By default, the prime is encoded as a big-endian sequence of octets

\* in an [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/ArrayBuffer). If the `bigint` option is `true`, then a

\* [bigint](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/BigInt) is provided.

\* @since v15.8.0

\* @param size The size (in bits) of the prime to generate.

\*/

function generatePrimeSync(size: number): ArrayBuffer;

function generatePrimeSync(size: number, options: GeneratePrimeOptionsBigInt): bigint;

function generatePrimeSync(size: number, options: GeneratePrimeOptionsArrayBuffer): ArrayBuffer;

function generatePrimeSync(size: number, options: GeneratePrimeOptions): ArrayBuffer | bigint;

interface CheckPrimeOptions {

/\*\*

\* The number of Miller-Rabin probabilistic primality iterations to perform.

\* When the value is 0 (zero), a number of checks is used that yields a false positive rate of at most 2-64 for random input.

\* Care must be used when selecting a number of checks.

\* Refer to the OpenSSL documentation for the BN\_is\_prime\_ex function nchecks options for more details.

\*

\* @default 0

\*/

checks?: number | undefined;

}

/\*\*

\* Checks the primality of the `candidate`.

\* @since v15.8.0

\* @param candidate A possible prime encoded as a sequence of big endian octets of arbitrary length.

\*/

function checkPrime(value: LargeNumberLike, callback: (err: Error | null, result: boolean) => void): void;

function checkPrime(value: LargeNumberLike, options: CheckPrimeOptions, callback: (err: Error | null, result: boolean) => void): void;

/\*\*

\* Checks the primality of the `candidate`.

\* @since v15.8.0

\* @param candidate A possible prime encoded as a sequence of big endian octets of arbitrary length.

\* @return `true` if the candidate is a prime with an error probability less than `0.25 \*\* options.checks`.

\*/

function checkPrimeSync(candidate: LargeNumberLike, options?: CheckPrimeOptions): boolean;

namespace webcrypto {

class CryptoKey {} // placeholder

}

}

declare module 'node:crypto' {

export \* from 'crypto';

}