/\*\*

\* HTTPS is the HTTP protocol over TLS/SSL. In Node.js this is implemented as a

\* separate module.

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

\*/

declare module 'https' {

import { Duplex } from 'node:stream';

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

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

import { URL } from 'node:url';

type ServerOptions = tls.SecureContextOptions & tls.TlsOptions & http.ServerOptions;

type RequestOptions = http.RequestOptions &

tls.SecureContextOptions & {

rejectUnauthorized?: boolean | undefined; // Defaults to true

servername?: string | undefined; // SNI TLS Extension

};

interface AgentOptions extends http.AgentOptions, tls.ConnectionOptions {

rejectUnauthorized?: boolean | undefined;

maxCachedSessions?: number | undefined;

}

/\*\*

\* An `Agent` object for HTTPS similar to `http.Agent`. See {@link request} for more information.

\* @since v0.4.5

\*/

class Agent extends http.Agent {

constructor(options?: AgentOptions);

options: AgentOptions;

}

interface Server extends http.Server {}

/\*\*

\* See `http.Server` for more information.

\* @since v0.3.4

\*/

class Server extends tls.Server {

constructor(requestListener?: http.RequestListener);

constructor(options: ServerOptions, requestListener?: http.RequestListener);

addListener(event: string, listener: (...args: any[]) => void): this;

addListener(event: 'keylog', listener: (line: Buffer, tlsSocket: tls.TLSSocket) => void): this;

addListener(event: 'newSession', listener: (sessionId: Buffer, sessionData: Buffer, callback: (err: Error, resp: Buffer) => void) => void): this;

addListener(event: 'OCSPRequest', listener: (certificate: Buffer, issuer: Buffer, callback: (err: Error | null, resp: Buffer) => void) => void): this;

addListener(event: 'resumeSession', listener: (sessionId: Buffer, callback: (err: Error, sessionData: Buffer) => void) => void): this;

addListener(event: 'secureConnection', listener: (tlsSocket: tls.TLSSocket) => void): this;

addListener(event: 'tlsClientError', listener: (err: Error, tlsSocket: tls.TLSSocket) => void): this;

addListener(event: 'close', listener: () => void): this;

addListener(event: 'connection', listener: (socket: Duplex) => void): this;

addListener(event: 'error', listener: (err: Error) => void): this;

addListener(event: 'listening', listener: () => void): this;

addListener(event: 'checkContinue', listener: http.RequestListener): this;

addListener(event: 'checkExpectation', listener: http.RequestListener): this;

addListener(event: 'clientError', listener: (err: Error, socket: Duplex) => void): this;

addListener(event: 'connect', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

addListener(event: 'request', listener: http.RequestListener): this;

addListener(event: 'upgrade', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

emit(event: string, ...args: any[]): boolean;

emit(event: 'keylog', line: Buffer, tlsSocket: tls.TLSSocket): boolean;

emit(event: 'newSession', sessionId: Buffer, sessionData: Buffer, callback: (err: Error, resp: Buffer) => void): boolean;

emit(event: 'OCSPRequest', certificate: Buffer, issuer: Buffer, callback: (err: Error | null, resp: Buffer) => void): boolean;

emit(event: 'resumeSession', sessionId: Buffer, callback: (err: Error, sessionData: Buffer) => void): boolean;

emit(event: 'secureConnection', tlsSocket: tls.TLSSocket): boolean;

emit(event: 'tlsClientError', err: Error, tlsSocket: tls.TLSSocket): boolean;

emit(event: 'close'): boolean;

emit(event: 'connection', socket: Duplex): boolean;

emit(event: 'error', err: Error): boolean;

emit(event: 'listening'): boolean;

emit(event: 'checkContinue', req: http.IncomingMessage, res: http.ServerResponse): boolean;

emit(event: 'checkExpectation', req: http.IncomingMessage, res: http.ServerResponse): boolean;

emit(event: 'clientError', err: Error, socket: Duplex): boolean;

emit(event: 'connect', req: http.IncomingMessage, socket: Duplex, head: Buffer): boolean;

emit(event: 'request', req: http.IncomingMessage, res: http.ServerResponse): boolean;

emit(event: 'upgrade', req: http.IncomingMessage, socket: Duplex, head: Buffer): boolean;

on(event: string, listener: (...args: any[]) => void): this;

on(event: 'keylog', listener: (line: Buffer, tlsSocket: tls.TLSSocket) => void): this;

on(event: 'newSession', listener: (sessionId: Buffer, sessionData: Buffer, callback: (err: Error, resp: Buffer) => void) => void): this;

on(event: 'OCSPRequest', listener: (certificate: Buffer, issuer: Buffer, callback: (err: Error | null, resp: Buffer) => void) => void): this;

on(event: 'resumeSession', listener: (sessionId: Buffer, callback: (err: Error, sessionData: Buffer) => void) => void): this;

on(event: 'secureConnection', listener: (tlsSocket: tls.TLSSocket) => void): this;

on(event: 'tlsClientError', listener: (err: Error, tlsSocket: tls.TLSSocket) => void): this;

on(event: 'close', listener: () => void): this;

on(event: 'connection', listener: (socket: Duplex) => void): this;

on(event: 'error', listener: (err: Error) => void): this;

on(event: 'listening', listener: () => void): this;

on(event: 'checkContinue', listener: http.RequestListener): this;

on(event: 'checkExpectation', listener: http.RequestListener): this;

on(event: 'clientError', listener: (err: Error, socket: Duplex) => void): this;

on(event: 'connect', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

on(event: 'request', listener: http.RequestListener): this;

on(event: 'upgrade', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

once(event: string, listener: (...args: any[]) => void): this;

once(event: 'keylog', listener: (line: Buffer, tlsSocket: tls.TLSSocket) => void): this;

once(event: 'newSession', listener: (sessionId: Buffer, sessionData: Buffer, callback: (err: Error, resp: Buffer) => void) => void): this;

once(event: 'OCSPRequest', listener: (certificate: Buffer, issuer: Buffer, callback: (err: Error | null, resp: Buffer) => void) => void): this;

once(event: 'resumeSession', listener: (sessionId: Buffer, callback: (err: Error, sessionData: Buffer) => void) => void): this;

once(event: 'secureConnection', listener: (tlsSocket: tls.TLSSocket) => void): this;

once(event: 'tlsClientError', listener: (err: Error, tlsSocket: tls.TLSSocket) => void): this;

once(event: 'close', listener: () => void): this;

once(event: 'connection', listener: (socket: Duplex) => void): this;

once(event: 'error', listener: (err: Error) => void): this;

once(event: 'listening', listener: () => void): this;

once(event: 'checkContinue', listener: http.RequestListener): this;

once(event: 'checkExpectation', listener: http.RequestListener): this;

once(event: 'clientError', listener: (err: Error, socket: Duplex) => void): this;

once(event: 'connect', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

once(event: 'request', listener: http.RequestListener): this;

once(event: 'upgrade', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

prependListener(event: string, listener: (...args: any[]) => void): this;

prependListener(event: 'keylog', listener: (line: Buffer, tlsSocket: tls.TLSSocket) => void): this;

prependListener(event: 'newSession', listener: (sessionId: Buffer, sessionData: Buffer, callback: (err: Error, resp: Buffer) => void) => void): this;

prependListener(event: 'OCSPRequest', listener: (certificate: Buffer, issuer: Buffer, callback: (err: Error | null, resp: Buffer) => void) => void): this;

prependListener(event: 'resumeSession', listener: (sessionId: Buffer, callback: (err: Error, sessionData: Buffer) => void) => void): this;

prependListener(event: 'secureConnection', listener: (tlsSocket: tls.TLSSocket) => void): this;

prependListener(event: 'tlsClientError', listener: (err: Error, tlsSocket: tls.TLSSocket) => void): this;

prependListener(event: 'close', listener: () => void): this;

prependListener(event: 'connection', listener: (socket: Duplex) => void): this;

prependListener(event: 'error', listener: (err: Error) => void): this;

prependListener(event: 'listening', listener: () => void): this;

prependListener(event: 'checkContinue', listener: http.RequestListener): this;

prependListener(event: 'checkExpectation', listener: http.RequestListener): this;

prependListener(event: 'clientError', listener: (err: Error, socket: Duplex) => void): this;

prependListener(event: 'connect', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

prependListener(event: 'request', listener: http.RequestListener): this;

prependListener(event: 'upgrade', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

prependOnceListener(event: string, listener: (...args: any[]) => void): this;

prependOnceListener(event: 'keylog', listener: (line: Buffer, tlsSocket: tls.TLSSocket) => void): this;

prependOnceListener(event: 'newSession', listener: (sessionId: Buffer, sessionData: Buffer, callback: (err: Error, resp: Buffer) => void) => void): this;

prependOnceListener(event: 'OCSPRequest', listener: (certificate: Buffer, issuer: Buffer, callback: (err: Error | null, resp: Buffer) => void) => void): this;

prependOnceListener(event: 'resumeSession', listener: (sessionId: Buffer, callback: (err: Error, sessionData: Buffer) => void) => void): this;

prependOnceListener(event: 'secureConnection', listener: (tlsSocket: tls.TLSSocket) => void): this;

prependOnceListener(event: 'tlsClientError', listener: (err: Error, tlsSocket: tls.TLSSocket) => void): this;

prependOnceListener(event: 'close', listener: () => void): this;

prependOnceListener(event: 'connection', listener: (socket: Duplex) => void): this;

prependOnceListener(event: 'error', listener: (err: Error) => void): this;

prependOnceListener(event: 'listening', listener: () => void): this;

prependOnceListener(event: 'checkContinue', listener: http.RequestListener): this;

prependOnceListener(event: 'checkExpectation', listener: http.RequestListener): this;

prependOnceListener(event: 'clientError', listener: (err: Error, socket: Duplex) => void): this;

prependOnceListener(event: 'connect', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

prependOnceListener(event: 'request', listener: http.RequestListener): this;

prependOnceListener(event: 'upgrade', listener: (req: http.IncomingMessage, socket: Duplex, head: Buffer) => void): this;

}

/\*\*

\* ```js

\* // curl -k https://localhost:8000/

\* const https = require('https');

\* const fs = require('fs');

\*

\* const options = {

\* key: fs.readFileSync('test/fixtures/keys/agent2-key.pem'),

\* cert: fs.readFileSync('test/fixtures/keys/agent2-cert.pem')

\* };

\*

\* https.createServer(options, (req, res) => {

\* res.writeHead(200);

\* res.end('hello world\n');

\* }).listen(8000);

\* ```

\*

\* Or

\*

\* ```js

\* const https = require('https');

\* const fs = require('fs');

\*

\* const options = {

\* pfx: fs.readFileSync('test/fixtures/test\_cert.pfx'),

\* passphrase: 'sample'

\* };

\*

\* https.createServer(options, (req, res) => {

\* res.writeHead(200);

\* res.end('hello world\n');

\* }).listen(8000);

\* ```

\* @since v0.3.4

\* @param options Accepts `options` from `createServer`, `createSecureContext` and `createServer`.

\* @param requestListener A listener to be added to the `'request'` event.

\*/

function createServer(requestListener?: http.RequestListener): Server;

function createServer(options: ServerOptions, requestListener?: http.RequestListener): Server;

/\*\*

\* Makes a request to a secure web server.

\*

\* The following additional `options` from `tls.connect()` are also accepted:`ca`, `cert`, `ciphers`, `clientCertEngine`, `crl`, `dhparam`, `ecdhCurve`,`honorCipherOrder`, `key`, `passphrase`,

\* `pfx`, `rejectUnauthorized`,`secureOptions`, `secureProtocol`, `servername`, `sessionIdContext`,`highWaterMark`.

\*

\* `options` can be an object, a string, or a `URL` object. If `options` is a

\* string, it is automatically parsed with `new URL()`. If it is a `URL` object, it will be automatically converted to an ordinary `options` object.

\*

\* `https.request()` returns an instance of the `http.ClientRequest` class. The `ClientRequest` instance is a writable stream. If one needs to

\* upload a file with a POST request, then write to the `ClientRequest` object.

\*

\* ```js

\* const https = require('https');

\*

\* const options = {

\* hostname: 'encrypted.google.com',

\* port: 443,

\* path: '/',

\* method: 'GET'

\* };

\*

\* const req = https.request(options, (res) => {

\* console.log('statusCode:', res.statusCode);

\* console.log('headers:', res.headers);

\*

\* res.on('data', (d) => {

\* process.stdout.write(d);

\* });

\* });

\*

\* req.on('error', (e) => {

\* console.error(e);

\* });

\* req.end();

\* ```

\*

\* Example using options from `tls.connect()`:

\*

\* ```js

\* const options = {

\* hostname: 'encrypted.google.com',

\* port: 443,

\* path: '/',

\* method: 'GET',

\* key: fs.readFileSync('test/fixtures/keys/agent2-key.pem'),

\* cert: fs.readFileSync('test/fixtures/keys/agent2-cert.pem')

\* };

\* options.agent = new https.Agent(options);

\*

\* const req = https.request(options, (res) => {

\* // ...

\* });

\* ```

\*

\* Alternatively, opt out of connection pooling by not using an `Agent`.

\*

\* ```js

\* const options = {

\* hostname: 'encrypted.google.com',

\* port: 443,

\* path: '/',

\* method: 'GET',

\* key: fs.readFileSync('test/fixtures/keys/agent2-key.pem'),

\* cert: fs.readFileSync('test/fixtures/keys/agent2-cert.pem'),

\* agent: false

\* };

\*

\* const req = https.request(options, (res) => {

\* // ...

\* });

\* ```

\*

\* Example using a `URL` as `options`:

\*

\* ```js

\* const options = new URL('https://abc:xyz@example.com');

\*

\* const req = https.request(options, (res) => {

\* // ...

\* });

\* ```

\*

\* Example pinning on certificate fingerprint, or the public key (similar to`pin-sha256`):

\*

\* ```js

\* const tls = require('tls');

\* const https = require('https');

\* const crypto = require('crypto');

\*

\* function sha256(s) {

\* return crypto.createHash('sha256').update(s).digest('base64');

\* }

\* const options = {

\* hostname: 'github.com',

\* port: 443,

\* path: '/',

\* method: 'GET',

\* checkServerIdentity: function(host, cert) {

\* // Make sure the certificate is issued to the host we are connected to

\* const err = tls.checkServerIdentity(host, cert);

\* if (err) {

\* return err;

\* }

\*

\* // Pin the public key, similar to HPKP pin-sha25 pinning

\* const pubkey256 = 'pL1+qb9HTMRZJmuC/bB/ZI9d302BYrrqiVuRyW+DGrU=';

\* if (sha256(cert.pubkey) !== pubkey256) {

\* const msg = 'Certificate verification error: ' +

\* `The public key of '${cert.subject.CN}' ` +

\* 'does not match our pinned fingerprint';

\* return new Error(msg);

\* }

\*

\* // Pin the exact certificate, rather than the pub key

\* const cert256 = '25:FE:39:32:D9:63:8C:8A:FC:A1:9A:29:87:' +

\* 'D8:3E:4C:1D:98:DB:71:E4:1A:48:03:98:EA:22:6A:BD:8B:93:16';

\* if (cert.fingerprint256 !== cert256) {

\* const msg = 'Certificate verification error: ' +

\* `The certificate of '${cert.subject.CN}' ` +

\* 'does not match our pinned fingerprint';

\* return new Error(msg);

\* }

\*

\* // This loop is informational only.

\* // Print the certificate and public key fingerprints of all certs in the

\* // chain. Its common to pin the public key of the issuer on the public

\* // internet, while pinning the public key of the service in sensitive

\* // environments.

\* do {

\* console.log('Subject Common Name:', cert.subject.CN);

\* console.log(' Certificate SHA256 fingerprint:', cert.fingerprint256);

\*

\* hash = crypto.createHash('sha256');

\* console.log(' Public key ping-sha256:', sha256(cert.pubkey));

\*

\* lastprint256 = cert.fingerprint256;

\* cert = cert.issuerCertificate;

\* } while (cert.fingerprint256 !== lastprint256);

\*

\* },

\* };

\*

\* options.agent = new https.Agent(options);

\* const req = https.request(options, (res) => {

\* console.log('All OK. Server matched our pinned cert or public key');

\* console.log('statusCode:', res.statusCode);

\* // Print the HPKP values

\* console.log('headers:', res.headers['public-key-pins']);

\*

\* res.on('data', (d) => {});

\* });

\*

\* req.on('error', (e) => {

\* console.error(e.message);

\* });

\* req.end();

\* ```

\*

\* Outputs for example:

\*

\* ```text

\* Subject Common Name: github.com

\* Certificate SHA256 fingerprint: 25:FE:39:32:D9:63:8C:8A:FC:A1:9A:29:87:D8:3E:4C:1D:98:DB:71:E4:1A:48:03:98:EA:22:6A:BD:8B:93:16

\* Public key ping-sha256: pL1+qb9HTMRZJmuC/bB/ZI9d302BYrrqiVuRyW+DGrU=

\* Subject Common Name: DigiCert SHA2 Extended Validation Server CA

\* Certificate SHA256 fingerprint: 40:3E:06:2A:26:53:05:91:13:28:5B:AF:80:A0:D4:AE:42:2C:84:8C:9F:78:FA:D0:1F:C9:4B:C5:B8:7F:EF:1A

\* Public key ping-sha256: RRM1dGqnDFsCJXBTHky16vi1obOlCgFFn/yOhI/y+ho=

\* Subject Common Name: DigiCert High Assurance EV Root CA

\* Certificate SHA256 fingerprint: 74:31:E5:F4:C3:C1:CE:46:90:77:4F:0B:61:E0:54:40:88:3B:A9:A0:1E:D0:0B:A6:AB:D7:80:6E:D3:B1:18:CF

\* Public key ping-sha256: WoiWRyIOVNa9ihaBciRSC7XHjliYS9VwUGOIud4PB18=

\* All OK. Server matched our pinned cert or public key

\* statusCode: 200

\* headers: max-age=0; pin-sha256="WoiWRyIOVNa9ihaBciRSC7XHjliYS9VwUGOIud4PB18="; pin-sha256="RRM1dGqnDFsCJXBTHky16vi1obOlCgFFn/yOhI/y+ho=";

\* pin-sha256="k2v657xBsOVe1PQRwOsHsw3bsGT2VzIqz5K+59sNQws="; pin-sha256="K87oWBWM9UZfyddvDfoxL+8lpNyoUB2ptGtn0fv6G2Q="; pin-sha256="IQBnNBEiFuhj+8x6X8XLgh01V9Ic5/V3IRQLNFFc7v4=";

\* pin-sha256="iie1VXtL7HzAMF+/PVPR9xzT80kQxdZeJ+zduCB3uj0="; pin-sha256="LvRiGEjRqfzurezaWuj8Wie2gyHMrW5Q06LspMnox7A="; includeSubDomains

\* ```

\* @since v0.3.6

\* @param options Accepts all `options` from `request`, with some differences in default values:

\*/

function request(options: RequestOptions | string | URL, callback?: (res: http.IncomingMessage) => void): http.ClientRequest;

function request(url: string | URL, options: RequestOptions, callback?: (res: http.IncomingMessage) => void): http.ClientRequest;

/\*\*

\* Like `http.get()` but for HTTPS.

\*

\* `options` can be an object, a string, or a `URL` object. If `options` is a

\* string, it is automatically parsed with `new URL()`. If it is a `URL` object, it will be automatically converted to an ordinary `options` object.

\*

\* ```js

\* const https = require('https');

\*

\* https.get('https://encrypted.google.com/', (res) => {

\* console.log('statusCode:', res.statusCode);

\* console.log('headers:', res.headers);

\*

\* res.on('data', (d) => {

\* process.stdout.write(d);

\* });

\*

\* }).on('error', (e) => {

\* console.error(e);

\* });

\* ```

\* @since v0.3.6

\* @param options Accepts the same `options` as {@link request}, with the `method` always set to `GET`.

\*/

function get(options: RequestOptions | string | URL, callback?: (res: http.IncomingMessage) => void): http.ClientRequest;

function get(url: string | URL, options: RequestOptions, callback?: (res: http.IncomingMessage) => void): http.ClientRequest;

let globalAgent: Agent;

}

declare module 'node:https' {

export \* from 'https';

}