```
// Copyright 2016 Joyent, Inc.

module.exports = Certificate;

var assert = require('assert-plus');
var algs = require('./algs');
var crypto = require('crypto');
var Fingerprint = require('./fingerprint');
var Signature = require('./signature');
var errs = require('./errors');
var util = require('util');
var utils = require('./utils');
var Key = require('./key');
var PrivateKey = require('./private-key');
var Identity = require('./identity');

var formats = {};
formats['openssh'] = require('./formats/openssh-cert');
formats['x509'] = require('./formats/x509');
formats['pem'] = require('./formats/x509-pem');

var CertificateParseError = errs.CertificateParseError;
var InvalidAlgorithmError = errs.InvalidAlgorithmError;

function Certificate(opts) {
	assert.object(opts, 'options');
	assert.arrayOfObject(opts.subjects, 'options.subjects');
	utils.assertCompatible(opts.subjects[0], Identity, [1, 0],
	    'options.subjects');
	utils.assertCompatible(opts.subjectKey, Key, [1, 0],
	    'options.subjectKey');
	utils.assertCompatible(opts.issuer, Identity, [1, 0], 'options.issuer');
	if (opts.issuerKey !== undefined) {
		utils.assertCompatible(opts.issuerKey, Key, [1, 0],
		    'options.issuerKey');
	}
	assert.object(opts.signatures, 'options.signatures');
	assert.buffer(opts.serial, 'options.serial');
	assert.date(opts.validFrom, 'options.validFrom');
	assert.date(opts.validUntil, 'optons.validUntil');

	assert.optionalArrayOfString(opts.purposes, 'options.purposes');

	this._hashCache = {};

	this.subjects = opts.subjects;
	this.issuer = opts.issuer;
	this.subjectKey = opts.subjectKey;
	this.issuerKey = opts.issuerKey;
	this.signatures = opts.signatures;
	this.serial = opts.serial;
	this.validFrom = opts.validFrom;
	this.validUntil = opts.validUntil;
	this.purposes = opts.purposes;
}

Certificate.formats = formats;

Certificate.prototype.toBuffer = function (format, options) {
	if (format === undefined)
		format = 'x509';
	assert.string(format, 'format');
	assert.object(formats[format], 'formats[format]');
	assert.optionalObject(options, 'options');

	return (formats[format].write(this, options));
};

Certificate.prototype.toString = function (format, options) {
	if (format === undefined)
		format = 'pem';
	return (this.toBuffer(format, options).toString());
};

Certificate.prototype.fingerprint = function (algo) {
	if (algo === undefined)
		algo = 'sha256';
	assert.string(algo, 'algorithm');
	var opts = {
		type: 'certificate',
		hash: this.hash(algo),
		algorithm: algo
	};
	return (new Fingerprint(opts));
};

Certificate.prototype.hash = function (algo) {
	assert.string(algo, 'algorithm');
	algo = algo.toLowerCase();
	if (algs.hashAlgs[algo] === undefined)
		throw (new InvalidAlgorithmError(algo));

	if (this._hashCache[algo])
		return (this._hashCache[algo]);

	var hash = crypto.createHash(algo).
	    update(this.toBuffer('x509')).digest();
	this._hashCache[algo] = hash;
	return (hash);
};

Certificate.prototype.isExpired = function (when) {
	if (when === undefined)
		when = new Date();
	return (!((when.getTime() >= this.validFrom.getTime()) &&
		(when.getTime() < this.validUntil.getTime())));
};
```

```
Certificate.prototype.isSignedBy = function (issuerCert) {
	utils.assertCompatible(issuerCert, Certificate, [1, 0], 'issuer');

	if (!this.issuer.equals(issuerCert.subjects[0]))
		return (false);
	if (this.issuer.purposes && this.issuer.purposes.length > 0 &&
	    this.issuer.purposes.indexOf('ca') === -1) {
		return (false);
	}

	return (this.isSignedByKey(issuerCert.subjectKey));
};

Certificate.prototype.isSignedByKey = function (issuerKey) {
	utils.assertCompatible(issuerKey, Key, [1, 2], 'issuerKey');

	if (this.issuerKey !== undefined) {
		return (this.issuerKey.
		    fingerprint('sha512').matches(issuerKey));
	}

	var fmt = Object.keys(this.signatures)[0];
	var valid = formats[fmt].verify(this, issuerKey);
	if (valid)
		this.issuerKey = issuerKey;
	return (valid);
};

Certificate.prototype.signWith = function (key) {
	utils.assertCompatible(key, PrivateKey, [1, 2], 'key');
	var fmts = Object.keys(formats);
	var didOne = false;
	for (var i = 0; i < fmts.length; ++i) {
		if (fmts[i] !== 'pem') {
			var ret = formats[fmts[i]].sign(this, key);
			if (ret === true)
				didOne = true;
		}
	}
	if (!didOne) {
		throw (new Error('Failed to sign the certificate for any ' +
		    'available certificate formats'));
	}
};

Certificate.createSelfSigned = function (subjectOrSubjects, key, options) {
	var subjects;
	if (Array.isArray(subjectOrSubjects))
		subjects = subjectOrSubjects;
	else
		subjects = [subjectOrSubjects];

	assert.arrayOfObject(subjects);
	subjects.forEach(function (subject) {
		utils.assertCompatible(subject, Identity, [1, 0], 'subject');
	});

	utils.assertCompatible(key, PrivateKey, [1, 2], 'private key');

	assert.optionalObject(options, 'options');
	if (options === undefined)
		options = {};
	assert.optionalObject(options.validFrom, 'options.validFrom');
	assert.optionalObject(options.validUntil, 'options.validUntil');
	var validFrom = options.validFrom;
	var validUntil = options.validUntil;
	if (validFrom === undefined)
		validFrom = new Date();
	if (validUntil === undefined) {
		assert.optionalNumber(options.lifetime, 'options.lifetime');
		var lifetime = options.lifetime;
		if (lifetime === undefined)
			lifetime = 10*365*24*3600;
		validUntil = new Date();
		validUntil.setTime(validUntil.getTime() + lifetime*1000);
	}
	assert.optionalBuffer(options.serial, 'options.serial');
	var serial = options.serial;
	if (serial === undefined)
		serial = new Buffer('0000000000000001', 'hex');

	var purposes = options.purposes;
	if (purposes === undefined)
		purposes = [];

	if (purposes.indexOf('signature') === -1)
		purposes.push('signature');

	/* Self-signed certs are always CAs. */
	if (purposes.indexOf('ca') === -1)
		purposes.push('ca');
	if (purposes.indexOf('crl') === -1)
		purposes.push('crl');

	/*
	 * If we weren't explicitly given any other purposes, do the sensible
	 * thing and add some basic ones depending on the subject type.
	 */
	if (purposes.length <= 3) {
		var hostSubjects = subjects.filter(function (subject) {
			return (subject.type === 'host');
		});
		var userSubjects = subjects.filter(function (subject) {
			return (subject.type === 'user');
		});
		if (hostSubjects.length > 0) {
			if (purposes.indexOf('serverAuth') === -1)
				purposes.push('serverAuth');
			}
			if (userSubjects.length > 0) {
				if (purposes.indexOf('clientAuth') === -1)
					purposes.push('clientAuth');
			}
			if (userSubjects.length > 0 || hostSubjects.length > 0) {
				if (purposes.indexOf('keyAgreement') === -1)
					purposes.push('keyAgreement');
				if (key.type === 'rsa' &&
				    purposes.indexOf('encryption') === -1)
					purposes.push('encryption');
			}
		}

		var cert = new Certificate({
			subjects: subjects,
			issuer: subjects[0],
			subjectKey: key.toPublic(),
			issuerKey: key.toPublic(),
			signatures: {},
			serial: serial,
			validFrom: validFrom,
			validUntil: validUntil,
			purposes: purposes
		});
		cert.signWith(key);

		return (cert);
};

Certificate.create =
    function (subjectOrSubjects, key, issuer, issuerKey, options) {
		var subjects;
		if (Array.isArray(subjectOrSubjects))
			subjects = subjectOrSubjects;
		else
			subjects = [subjectOrSubjects];

		assert.arrayOfObject(subjects);
		subjects.forEach(function (subject) {
			utils.assertCompatible(subject, Identity, [1, 0], 'subject');
		});

		utils.assertCompatible(key, Key, [1, 0], 'key');
		if (PrivateKey.isPrivateKey(key))
			key = key.toPublic();
		utils.assertCompatible(issuer, Identity, [1, 0], 'issuer');
		utils.assertCompatible(issuerKey, PrivateKey, [1, 2], 'issuer key');

		assert.optionalObject(options, 'options');
		if (options === undefined)
			options = {};
		assert.optionalObject(options.validFrom, 'options.validFrom');
		assert.optionalObject(options.validUntil, 'options.validUntil');
```

```
var validFrom = options.validFrom;
var validUntil = options.validUntil;
if (validFrom === undefined)
	validFrom = new Date();
if (validUntil === undefined) {
	assert.optionalNumber(options.lifetime, 'options.lifetime');
	var lifetime = options.lifetime;
	if (lifetime === undefined)
		lifetime = 10*365*24*3600;
	validUntil = new Date();
	validUntil.setTime(validUntil.getTime() + lifetime*1000);
}
assert.optionalBuffer(options.serial, 'options.serial');
var serial = options.serial;
if (serial === undefined)
	serial = new Buffer('0000000000000001', 'hex');

var purposes = options.purposes;
if (purposes === undefined)
	purposes = [];

if (purposes.indexOf('signature') === -1)
	purposes.push('signature');

if (options.ca === true) {
	if (purposes.indexOf('ca') === -1)
		purposes.push('ca');
	if (purposes.indexOf('crl') === -1)
		purposes.push('crl');
}

var hostSubjects = subjects.filter(function (subject) {
	return (subject.type === 'host');
});
var userSubjects = subjects.filter(function (subject) {
	return (subject.type === 'user');
});
if (hostSubjects.length > 0) {
	if (purposes.indexOf('serverAuth') === -1)
		purposes.push('serverAuth');
}
if (userSubjects.length > 0) {
	if (purposes.indexOf('clientAuth') === -1)
		purposes.push('clientAuth');
}
if (userSubjects.length > 0 || hostSubjects.length > 0) {
	if (purposes.indexOf('keyAgreement') === -1)
		purposes.push('keyAgreement');
	if (key.type === 'rsa' &&
	    purposes.indexOf('encryption') === -1)
		purposes.push('encryption');
}

var cert = new Certificate({
			subjects: subjects,
			issuer: issuer,
			subjectKey: key,
			issuerKey: issuerKey.toPublic(),
			signatures: {},
			serial: serial,
			validFrom: validFrom,
			validUntil: validUntil,
			purposes: purposes
		});
		cert.signWith(issuerKey);

		return (cert);
};

Certificate.parse = function (data, format, options) {
		if (typeof (data) !== 'string')
			assert.buffer(data, 'data');
		if (format === undefined)
			format = 'auto';
		assert.string(format, 'format');
		if (typeof (options) === 'string')
			options = { filename: options };
		assert.optionalObject(options, 'options');
		if (options === undefined)
			options = {};
		assert.optionalString(options.filename, 'options.filename');
		if (options.filename === undefined)
			options.filename = '(unnamed)';

		assert.object(formats[format], 'formats[format]');

		try {
			var k = formats[format].read(data, options);
			return (k);
		} catch (e) {
			throw (new CertificateParseError(options.filename, format, e));
		}
};

Certificate.isCertificate = function (obj, ver) {
		return (utils.isCompatible(obj, Certificate, ver));
};

/*
 * API versions for Certificate:
 * [1,0] -- initial ver
 */
Certificate.prototype._sshpkApiVersion = [1, 0];

Certificate._oldVersionDetect = function (obj) {
		return ([1, 0]);
};
```