Orqixfkee

Rkmrkmvkoryw

Unbfrktghccq

lbhj Emsx'g

eiwemc

ofbjes

Mastering Cryptography Fundamentals with Node's `crypto` module

Cryptography

(the study of the techniques of secret writing)


node
JS
®

Crypto

Stability: 2 - Stable

Source Code: lib/crypto.js

The `node:crypto` module provides cryptographic functionality that includes a set of wrappers for OpenSSL's hash, HMAC, cipher, decipher, sign, and verify functions.

CJS ESM

```
const { createHmac } = await import('node:crypto');

const secret = 'abcdefg';
const hash = createHmac('sha256', secret)
               .update('I love cupcakes')
               .digest('hex');
console.log(hash);
// Prints:
//   c0fa1bc00531bd78ef38c628449c5102aeabd49b5dc3a2a516ea6ea959d6658e
```

Determining if crypto support is unavailable

OpenSSL

Cryptography and SSL/TLS Toolkit

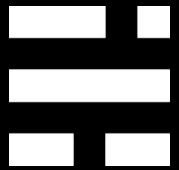
Node.js v18.15.0 documentation

▾ Table of contents

Yonatan Mevorach

{ WiX Dev Tools }

 @cowchimp

 blog.yonatan.dev

Cryptography Fundamentals

Encryption

Alice

Eve

Bob


01001100
01000101
01000111
01001111

plaintext

ciphertext

JSConf JP

PfbtMZX

PfbtMZX

PfbtMZX

JSConf JP

```
import { getCiphers } from "crypto";

console.log(getCiphers());

[
  'aes-128-cbc',        'aes-128-ccm',        'aes-128-cfb',
  'aes-128-cfb1',       'aes-128-cfb8',       'aes-128-ctr',
  'aes-128-ecb',        'aes-128-gcm',        'aes-128-ocb',
  'aes-128-ofb',        'aes-128-xts',        'aes-192-cbc',
  'aes-192-ccm',        'aes-192-cfb',        'aes-192-cfb1',
  'aes-192-cfb8',       'aes-192-ctr',        'aes-192-ecb',
  'aes-192-gcm',        'aes-192-ocb',        'aes-192-ofb',
```

```
import { getCiphers } from "crypto";

console.log(getCiphers());

[
  'aes-128-cbc',        'aes-128-ccm',        'aes-128-cfb',
  'aes-128-cfb1',       'aes-128-cfb8',       'aes-128-ctr',
  'aes-128-ecb',        'aes-128-gcm',        'aes-128-ocb',
  'aes-128-ofb',        'aes-128-xts',        'aes-192-cbc',
  'aes-192-ccm',        'aes-192-cfb',        'aes-192-cfb1',
  'aes-192-cfb8',       'aes-192-ctr',        'aes-192-ecb',
  'aes-192-gcm',        'aes-192-ocb',        'aes-192-ofb',
  'aes-256-cbc',        'aes-256-ccm',        'aes-256-cfb',
  'aes-256-cfb1',       'aes-256-cfb8',       'aes-256-ctr',
  'aes-256-ecb',        'aes-256-gcm',        'aes-256-ocb',
  'aes-256-ofb',        'aes-256-xts',        'aes128',
  'aes128-wrap',        'aes192',             'aes192-wrap',
  'aes256',             'aes256-wrap',        'aria-128-cbc',
  'aria-128-ccm',       'aria-128-cfb',       'aria-128-cfb1',
  'aria-128-cfb8',      'aria-128-ctr',       'aria-128-ecb',
  ... 126 more items
]
```

AES

Advanced Encryption Standard

aes-256-cbc

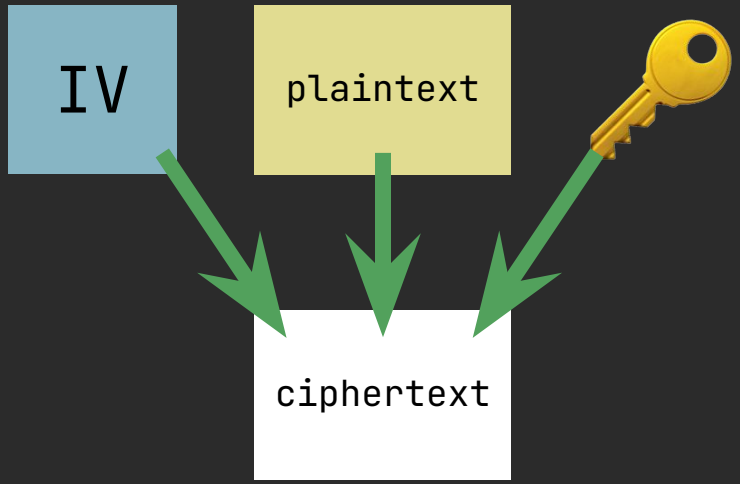
C B C

Cipher Block Chaining

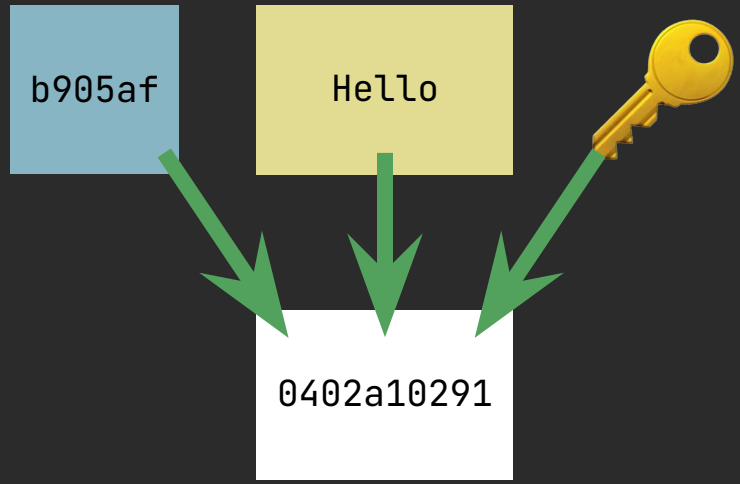
I V

Initialization Vector

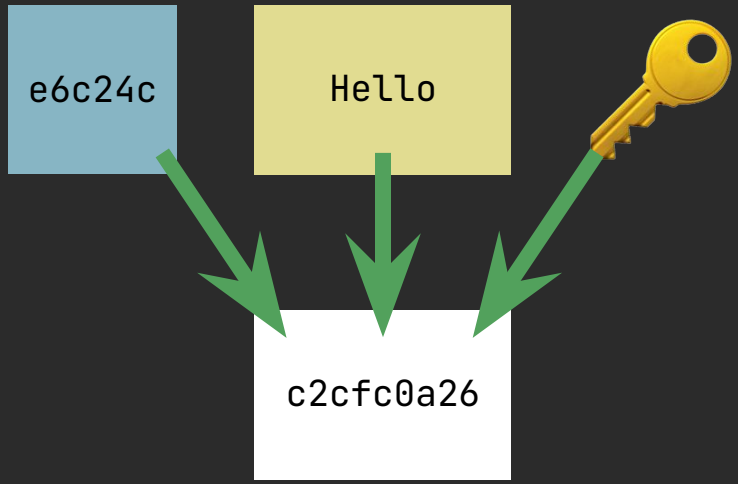

01001100
01000101
01000111
01001111


IV
plaintext
ciphertext


01001100
01000101
01000111
01001111


b905af
Hello
0402a10291


01001100
01000101
01000111
01001111


e6c24c
Hello
c2cfc0a26

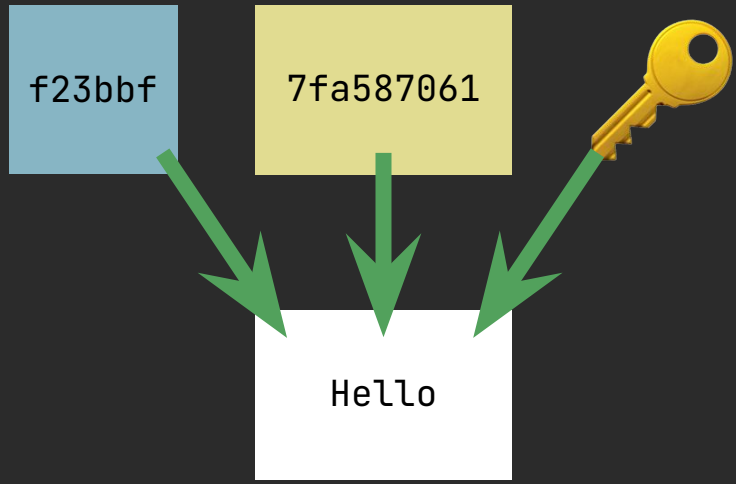
f23bbf

Hello

7fa587061

iv
ciphertext
algorithm


01001100
01000101
01000111
01001111


f23bbf
7fa587061
Hello

```
import crypto from "crypto";
import util from "util";
const randomBytes = util.promisify(crypto.randomBytes);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
  cipher.final(),
]);
```


```
import crypto from "crypto";
import util from "util";
const randomBytes = util.promisify(crypto.randomBytes);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
```


```
import crypto from "crypto";
import util from "util";
const randomBytes = util.promisify(crypto.randomBytes);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
  cipher.final(),
```


```
import crypto from "crypto";
import util from "util";
const randomBytes = util.promisify(crypto.randomBytes);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
```

```
import crypto from "crypto";
import util from "util";
const randomBytes = util.promisify(crypto.randomBytes);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
  cipher.final(),
]);
```

```
const randomBytes = util.promisify(crypto.randomBytes);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
  cipher.final(),
]);
```

```
const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const key = loadKey();

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
  cipher.final(),
]);
```

```
import crypto from "crypto";

function receive(iv, ciphertext, algorithm) {
  const key = loadKey();

  const decipher = crypto.createDecipheriv(algorithm, key, iv);

  const plaintext = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);

  console.log(plaintext.toString("utf8"));
  // Hello JSConf JP
}
```

```
import crypto from "crypto";

function receive(iv, ciphertext, algorithm) {
  const key = loadKey();

  const decipher = crypto.createDecipheriv(algorithm, key, iv);

  const plaintext = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);

  console.log(plaintext.toString("utf8"));
  // Hello JSConf JP
}
```

```
import crypto from "crypto";

function receive(iv, ciphertext, algorithm) {
  const key = loadKey();

  const decipher = crypto.createDecipheriv(algorithm, key, iv);

  const plaintext = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);

  console.log(plaintext.toString("utf8"));
  // Hello JSConf JP
}
```

```
import crypto from "crypto";

function receive(iv, ciphertext, algorithm) {
  const key = loadKey();

  const decipher = crypto.createDecipheriv(algorithm, key, iv);

  const plaintext = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);

  console.log(plaintext.toString("utf8"));
  // Hello JSConf JP
}
```

```
import crypto from "crypto";

function receive(iv, ciphertext, algorithm) {
  const key = loadKey();

  const decipher = crypto.createDecipheriv(algorithm, key, iv);

  const plaintext = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);

  console.log(plaintext.toString("utf8"));
  // Hello JSConf JP
}
```

```
function receive(iv, ciphertext, algorithm) {
  const key = loadKey();

  const decipher = crypto.createDecipheriv(algorithm, key, iv);

  const plaintext = Buffer.concat([
    decipher.update(ciphertext),
    decipher.final(),
  ]);

  console.log(plaintext.toString("utf8"));
  // Hello JSConf JP
}
```

```
import crypto from "crypto";
import util from "util";
const randomBytes = util.promisify(crypto.randomBytes);
const scrypt = util.promisify(crypto.scrypt);

const algorithm = "aes-256-cbc";

const plaintext = Buffer.from("Hello JSConf JP", "utf8");

const iv = await randomBytes(16);

const password = "p@@sw0rd";
const salt = await randomBytes(16);
const key = await scrypt(password, salt, 32);

const cipher = crypto.createCipheriv(algorithm, key, iv);

const ciphertext = Buffer.concat([
  cipher.update(plaintext),
```

h0pper

Cryptography Fundamentals

Key Derivation Functions

h0pper

Sp9IxQEzyzdZ94B4VyQoF+6e3EKgJPdoQDF99Ta

Password	Key
password	XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=
123456	jZae727K08KaOmKSgOaGzww/XVqGr/PKEgIMkjrcbJI=
qwerty	ZehL4zUy+3hMSBKWdfnv86aCsnFowOp0Syz1juAjN8U=
111111	vLFfghR5tNV3K9DKhmwArV+SbjWAcgZZzIDTnJ0JgCo=
ppppp	dylyfv5OoHljyL+JdqS6YiURDJgRwo+Rgdb/gRTptps=
1q2w3e4r	cquZT6LrQmwFHvWcrWF3UL/gbXz2MRKF/3nBnDKv0jY=
123123123	ky88G1YlfOhTmsJp16q0JVDaz4gY0HXwvfGZBWKq4+8=
aueuaueu	mzVJAUV6pKrLwUQ/+U+vcBAfkP4Vv9iSFzAWI7+eW60=

h0pper

oFzBqXkAn5TxjIvhWi

dPuYLRSLIr9w23egqkqAQP+z5rFz1DDYy8jcXTo

Notable mentions 🧐

- `crypto.hkdf`
  (HMAC-based Key Derivation Function)
- `crypto.pbkdf2`
  (Password-Based Key Derivation Function 2)

Cryptography Fundamentals

Randomness

Math.random()

The `Math.random()` static method returns a floating-point, pseudo-random number that's greater than or equal to 0 and less than 1, with approximately uniform distribution over that range — which you can then scale to your desired range. The implementation selects the initial seed to the random number generation algorithm; it cannot be chosen or reset by the user.

> **Note:** `Math.random()` *does not* provide cryptographically secure random numbers. Do not use them for anything related to security. Use the Web Crypto API instead, and more precisely the `window.crypto.getRandomValues()` method.

Try it

```
import crypto from "crypto";
import util from "util";

const randomBytes = util.promisify(crypto.randomBytes);
const randomFill = util.promisify(crypto.randomFill);
const randomInt = util.promisify(crypto.randomInt);

const buffer1 = await randomBytes(4);
console.log(buffer1.toString("hex"));
// 82e2e97d

const buffer2 = Buffer.alloc(4);
await randomFill(buffer2, 2, 1);
console.log(buffer2.toString("hex"));
// 00003500

const int = await randomInt(18, 180);
console.log(int);
// 137

const uuid = crypto.randomUUID();
console.log("uuid", uuid);
// 0748d0c6-3641-4858-876e-ec8420ba261d
```

```
import crypto from "crypto";
import util from "util";

const randomBytes = util.promisify(crypto.randomBytes);
const randomFill = util.promisify(crypto.randomFill);
const randomInt = util.promisify(crypto.randomInt);

const buffer1 = await randomBytes(4);
console.log(buffer1.toString("hex"));
// 82e2e97d

const buffer2 = Buffer.alloc(4);
await randomFill(buffer2, 2, 1);
console.log(buffer2.toString("hex"));
// 00003500
```

```
import crypto from "crypto";
import util from "util";

const randomBytes = util.promisify(crypto.randomBytes);
const randomFill = util.promisify(crypto.randomFill);
const randomInt = util.promisify(crypto.randomInt);

const buffer1 = await randomBytes(4);
console.log(buffer1.toString("hex"));
// 82e2e97d

const buffer2 = Buffer.alloc(4);
await randomFill(buffer2, 2, 1);
console.log(buffer2.toString("hex"));
// 00003500

const int = await randomInt(18, 180);
console.log(int);
// 127
```

```
const randomBytes = util.promisify(crypto.randomBytes);
const randomFill = util.promisify(crypto.randomFill);
const randomInt = util.promisify(crypto.randomInt);

const buffer1 = await randomBytes(4);
console.log(buffer1.toString("hex"));
// 82e2e97d

const buffer2 = Buffer.alloc(4);
await randomFill(buffer2, 2, 1);
console.log(buffer2.toString("hex"));
// 00003500

const int = await randomInt(18, 180);
console.log(int);
// 137

const uuid = crypto.randomUUID();
console.log("uuid", uuid);
// 0748d0c6-3641-4858-876e-ec8420ba261d
```

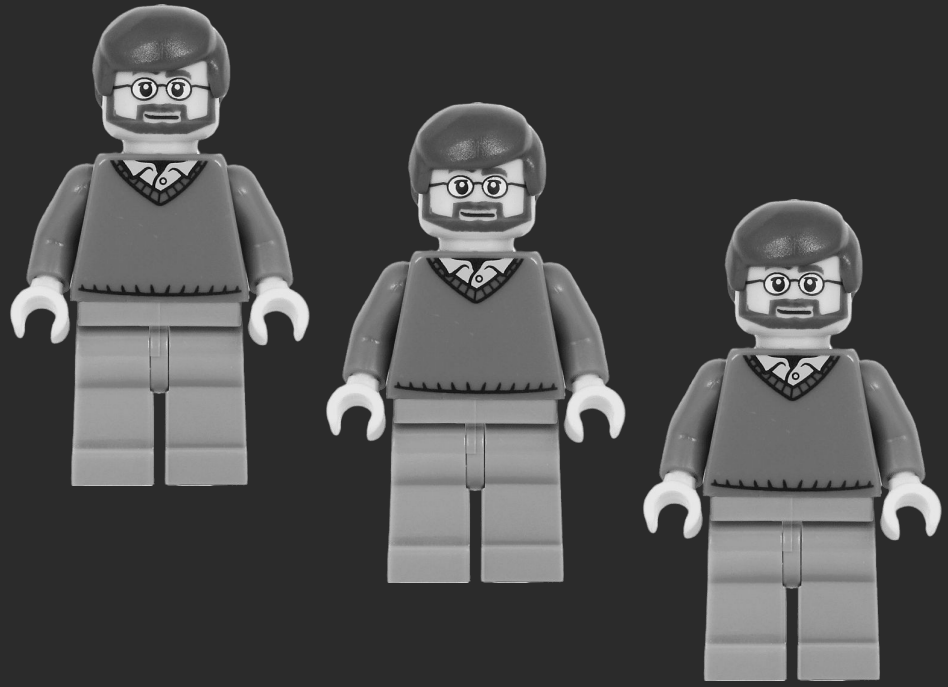
```
const buffer1 = await randomBytes(4);
console.log(buffer1.toString("hex"));
// 82e2e97d

const buffer2 = Buffer.alloc(4);
await randomFill(buffer2, 2, 1);
console.log(buffer2.toString("hex"));
// 00003500

const int = await randomInt(18, 180);
console.log(int);
// 137

const uuid = crypto.randomUUID();
console.log("uuid", uuid);
// 0748d0c6-3641-4858-876e-ec8420ba261d
```

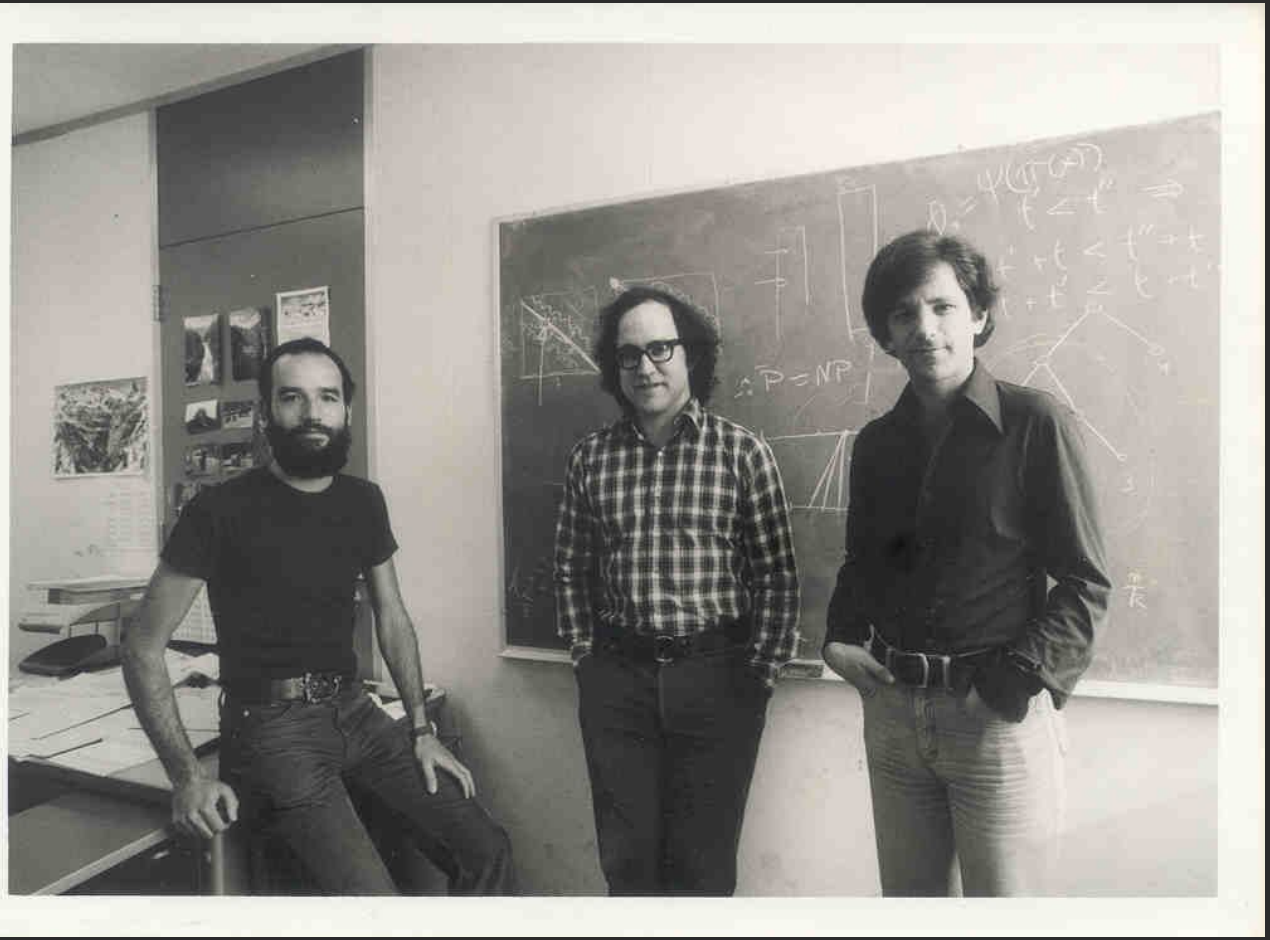
```
const buffer2 = Buffer.alloc(4);
await randomFill(buffer2, 2, 1);
console.log(buffer2.toString("hex"));
// 00003500

const int = await randomInt(18, 180);
console.log(int);
// 137

const uuid = crypto.randomUUID();
console.log("uuid", uuid);
// 0748d0c6-3641-4858-876e-ec8420ba261d
```


01001100
01000101
01000111
01001111

Cryptography
Fundamentals

Key Distribution Problem




P=NP

bob's private key → bob's public key

bob's public key

bob's public key

plaintext

ciphertext

ciphertext

bob's public key

plaintext

ciphertext

bob's private key

bob's public key

ciphertext

plaintext

```
import crypto from "crypto";
import util from "util";
import { writeFile } from "fs/promises";

const generateKeyPair = util.promisify(crypto.generateKeyPair);

const keyPair = await generateKeyPair("rsa", {
  modulusLength: 4096,
});

const publicKey = keyPair.publicKey.export({
  type: "spki",
  format: "pem",
});
await writeFile("bob-public.pem", publicKey);

const privateKey = keyPair.privateKey.export({
  type: "pkcs8",
  format: "pem",
});
await writeFile("bob-private.pem", privateKey);
```

```
import crypto from "crypto";
import util from "util";
import { writeFile } from "fs/promises";

const generateKeyPair = util.promisify(crypto.generateKeyPair);

const keyPair = await generateKeyPair("rsa", {
  modulusLength: 4096,
});

const publicKey = keyPair.publicKey.export({
  type: "spki",
  format: "pem",
});
await writeFile("bob-public.pem", publicKey);

const privateKey = keyPair.privateKey.export({
  type: "pkcs8",
```

```
import util from "util";
import { writeFile } from "fs/promises";

const generateKeyPair = util.promisify(crypto.generateKeyPair);

const keyPair = await generateKeyPair("rsa", {
  modulusLength: 4096,
});

const publicKey = keyPair.publicKey.export({
  type: "spki",
  format: "pem",
});
await writeFile("bob-public.pem", publicKey);

const privateKey = keyPair.privateKey.export({
  type: "pkcs8",
  format: "pem",
});
await writeFile("bob-private.pem", privateKey);
```

```
  modulusLength: 4096,
});

const publicKey = keyPair.publicKey.export({
  type: "spki",
  format: "pem",
});
await writeFile("bob-public.pem", publicKey);

const privateKey = keyPair.privateKey.export({
  type: "pkcs8",
  format: "pem",
});
await writeFile("bob-private.pem", privateKey);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

const bobPublicKey = crypto.createPublicKey(
  await readFile("bob-public.pem")
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
  plaintext
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

const bobPublicKey = crypto.createPublicKey(
  await readFile("bob-public.pem")
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

const bobPublicKey = crypto.createPublicKey(
  await readFile("bob-public.pem")
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
  plaintext
);
```

```
const bobPublicKey = crypto.createPublicKey(
  await readFile("bob-public.pem")
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
  plaintext
);
```

```
const bobPublicKey = crypto.createPublicKey(
  await readFile("bob-public.pem")
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
  plaintext
);
```

```
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
  plaintext
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

const bobPublicKey = crypto.createPublicKey(
  await readFile("bob-public.pem")
);

const plaintext = Buffer.from("Hello world!", "utf8");

const ciphertext = crypto.publicEncrypt(
  {
    key: bobPublicKey,
    padding: RSA_PKCS1_OAEP_PADDING,
  },
  plaintext
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

async function receive(ciphertext) {
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );

  console.log(plaintext.toString("utf8"));
  // Hello world!
}
```

```
const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

async function receive(ciphertext) {
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );

  console.log(plaintext.toString("utf8"));
  // Hello world!
}
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

async function receive(ciphertext) {
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );
```

```
async function receive(ciphertext) {
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );

  console.log(plaintext.toString("utf8"));
  // Hello world!
}
```

```
const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

async function receive(ciphertext) {
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );

  console.log(plaintext.toString("utf8"));
  // Hello world!
}
```

```
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );

  console.log(plaintext.toString("utf8"));
  // Hello world!
}
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const { RSA_PKCS1_OAEP_PADDING } = crypto.constants;

async function receive(ciphertext) {
  const bobPrivateKey = crypto.createPrivateKey(
    await readFile("bob-private.pem")
  );

  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );
```

```
  const plaintext = crypto.privateDecrypt(
    {
      key: bobPrivateKey,
      padding: RSA_PKCS1_OAEP_PADDING,
    },
    ciphertext
  );

  console.log(plaintext.toString("utf8"));
  // Hello world!
}
```

Notable mentions 🧐

- `crypto.createDiffieHellman`
  (Diffie-Hellman key exchange)
- `crypto.createECDH`
  (Elliptic Curve Diffie-Hellman)

Cryptography Fundamentals

Signing and Verifying

alice's private key

alice's public key

message

signature

alice's public key

message
signature

alice's private key

alice's public key

message

signature

alice's public key

message

signature

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```


```
import crypto from "crypto";
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```


```
import crypto from "crypto";
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```

```
import { readFile } from "fs/promises";

const alicePrivateKey = crypto.createPrivateKey(
  await readFile("alice-private.pem")
);

const message = Buffer.from("Bob has a ticket to JSConf JP", "utf8");

const signature = crypto.sign(
  "sha256",
  message,
  {
    key: alicePrivateKey,
  }
);
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

async function receive(message, signature) {
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
  // true
}
```

```
async function receive(message, signature) {
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
  // true
}
```

```
import crypto from "crypto";
import { readFile } from "fs/promises";

async function receive(message, signature) {
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
```

```
async function receive(message, signature) {
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
  // true
}
```

```
import { readFile } from "fs/promises";

async function receive(message, signature) {
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
```

```
async function receive(message, signature) {
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
  // true
```

```
  const alicePublicKey = crypto.createPublicKey(
    await readFile("alice-public.pem")
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
  // true
}
```

```
  );

  const isVerified = crypto.verify(
    "sha256",
    message,
    {
      key: alicePublicKey,
    },
    signature
  );

  console.log(isVerified);
  // true
}
```

```
      'SHA256',
      message,
      {
        key: alicePublicKey,
      },
      signature
    );

    console.log(isVerified);
    // true
}
```

Notable mentions 🧐

- `crypto.createHmac`
  (Hash-Based Message Authentication Code)

Cryptography
Fundamentals

Public Key Certificates

CERTIFICATE

OF ACHIEVEMENT

PROUDLY PRESENTED TO

Yonatan Mevorach

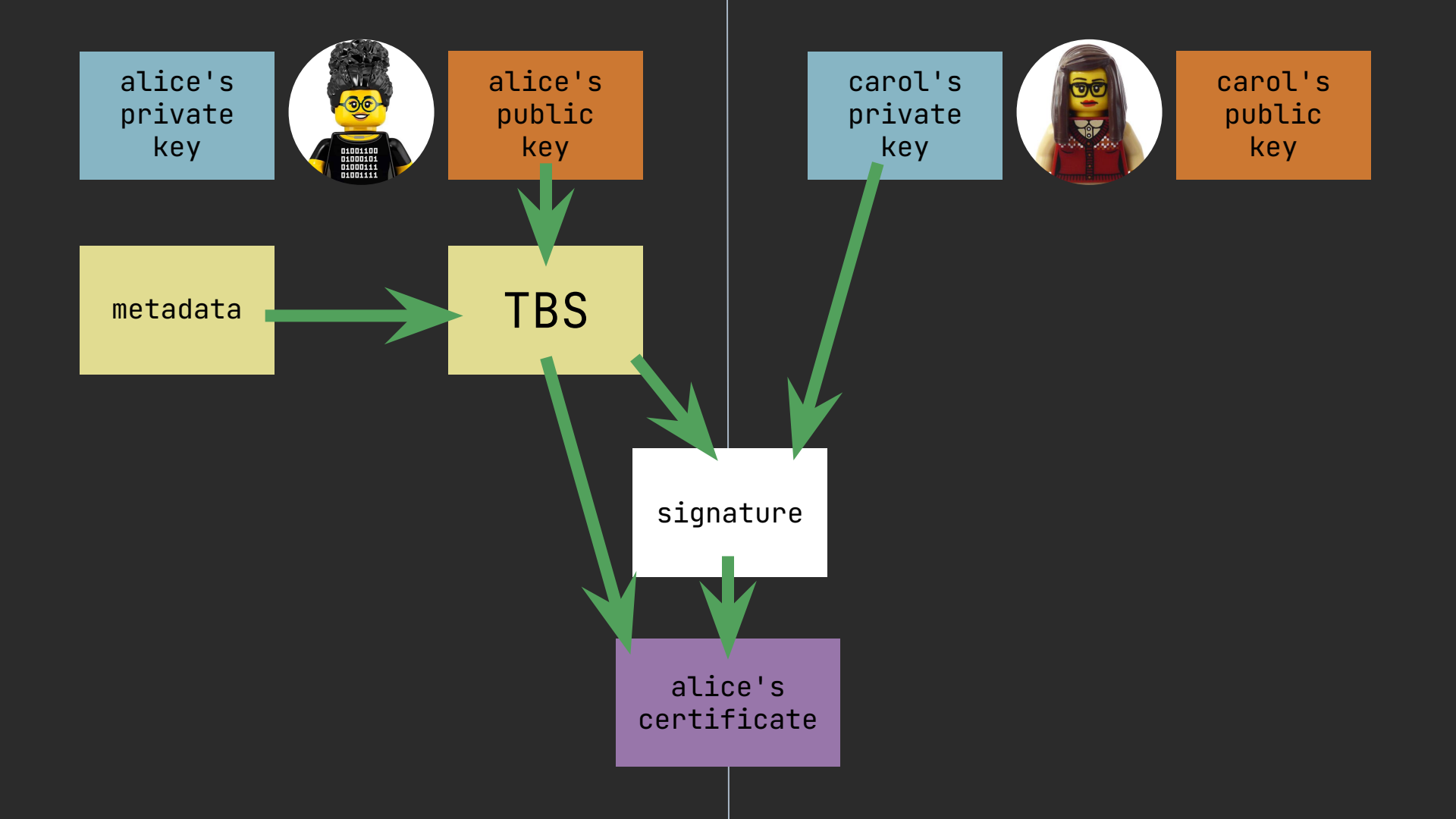
For being the absolute best at remembering Git commands,
without ever needing to consult documentation

Linus Torvalds
Creator of Git

Alice

Carol

Bob


alice's private key
alice's public key
carol's private key
carol's public key
metadata
TBS
signature
alice's certificate

carol's private key

carol's public key

alice's certificate

signature

metadata

alice's public key

✅

❌

```
import { X509Certificate } from "crypto";
import { readFile } from "fs/promises";

const aliceCert = new X509Certificate(await readFile("alice.cer"));

console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);

console.log(isVerified);
// true
```

```ts
import { X509Certificate } from "crypto";
import { readFile } from "fs/promises";

const aliceCert = new X509Certificate(await readFile("alice.cer"));

console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));
```

```
import { X509Certificate } from "crypto";
import { readFile } from "fs/promises";

const aliceCert = new X509Certificate(await readFile("alice.cer"));

console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);
```

```
import { X509Certificate } from "crypto";
import { readFile } from "fs/promises";

const aliceCert = new X509Certificate(await readFile("alice.cer"));

console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);

console.log(isVerified);
// true
```

```
import { X509Certificate } from "crypto";
import { readFile } from "fs/promises";

const aliceCert = new X509Certificate(await readFile("alice.cer"));

console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);

console.log(isVerified);
// true
```

```
const aliceCert = new X509Certificate(await readFile("alice.cer"));

console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);

console.log(isVerified);
// true
```

```
console.log(aliceCert.subject);
// CN=alice

console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);

console.log(isVerified);
// true
```

```
console.log(aliceCert.issuer);
// CN=carol

console.log(`${aliceCert.validFrom} - ${aliceCert.validTo}`);
// May  2 19:00:13 2023 GMT - Jul 31 19:00:12 2023 GMT

const carolCert = new X509Certificate(await readFile("carol.cer"));

const isVerified = aliceCert.verify(carolCert.publicKey);

console.log(isVerified);
// true
```

Alice

Carol

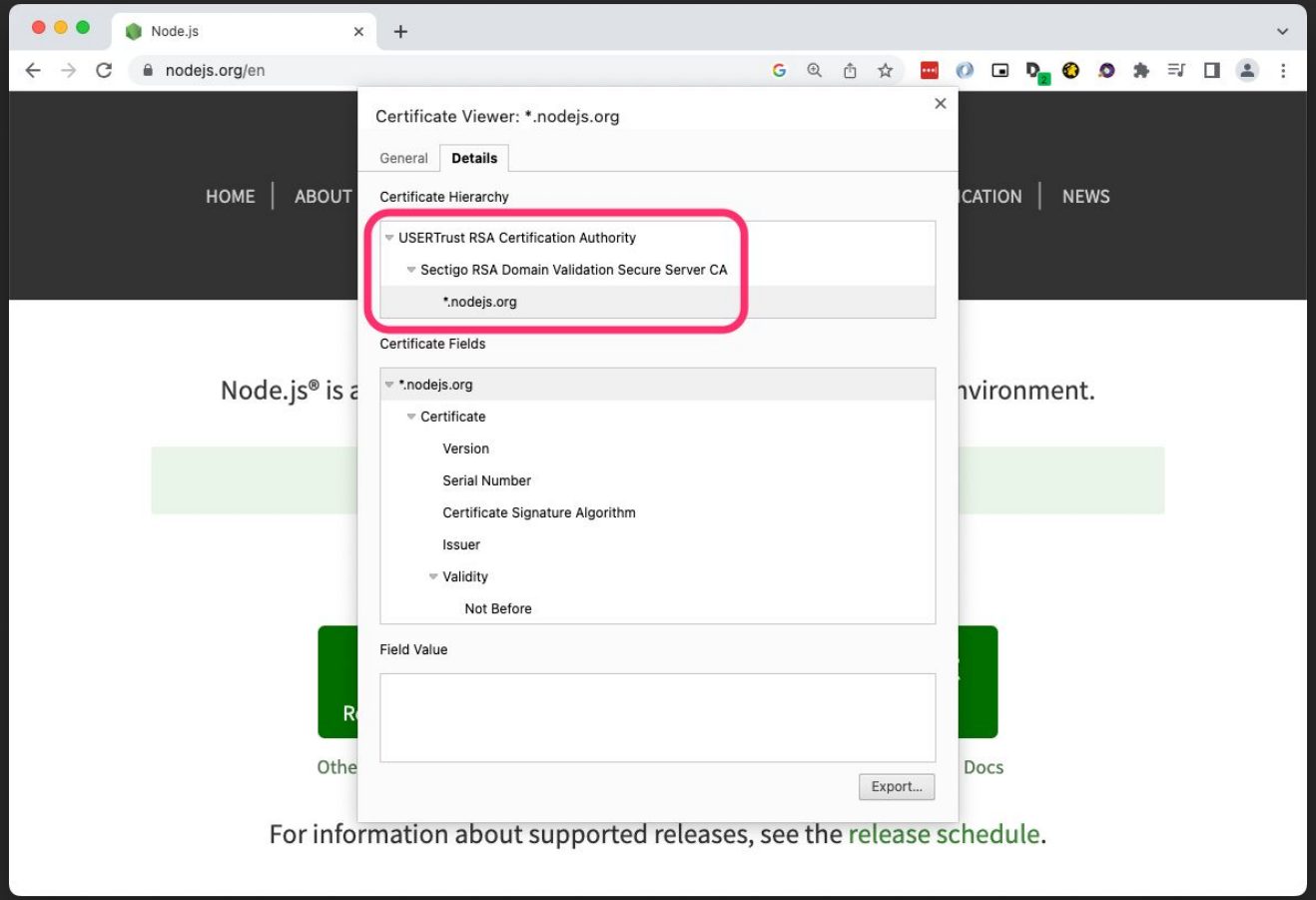
Bob

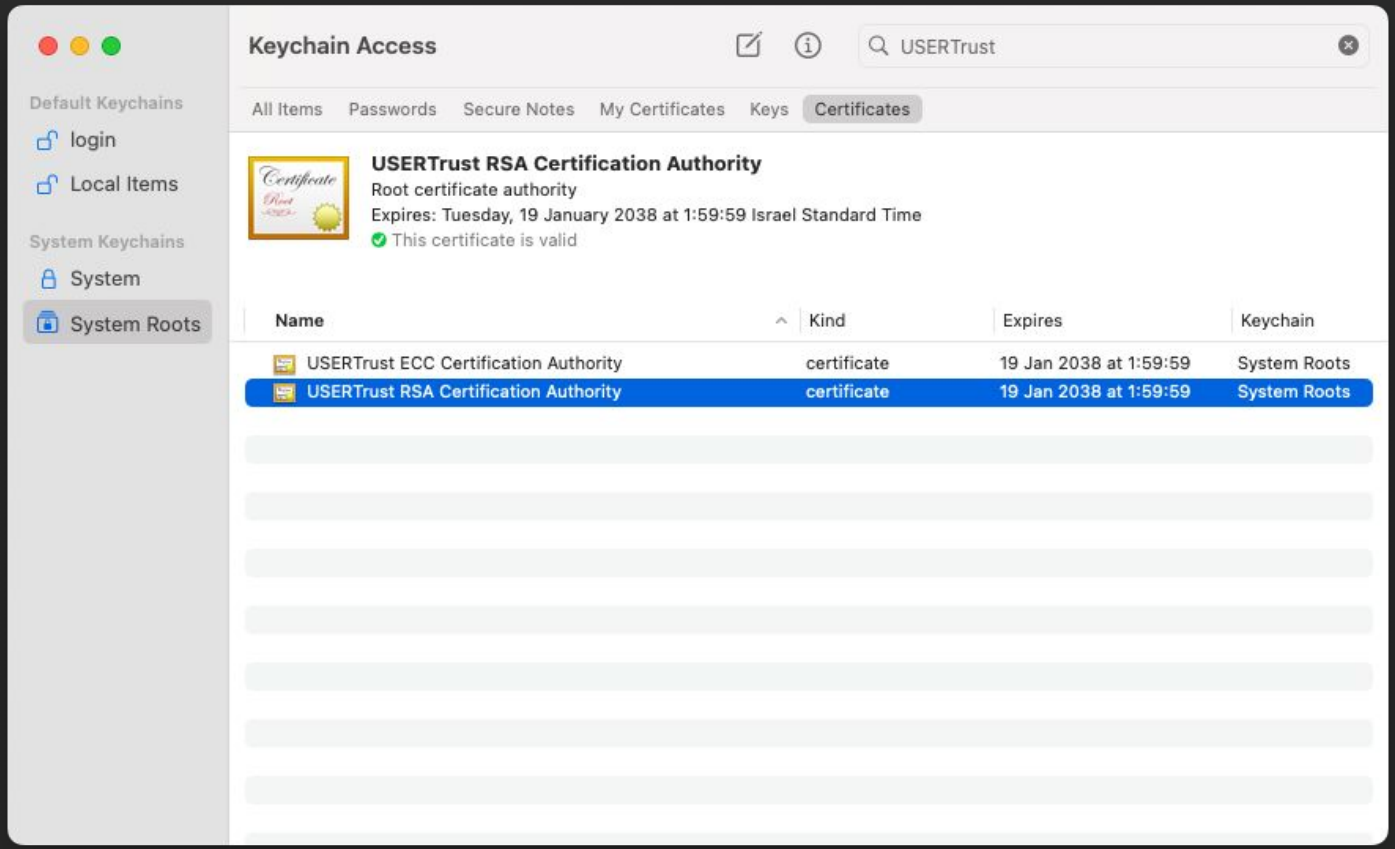
Alice

Carol

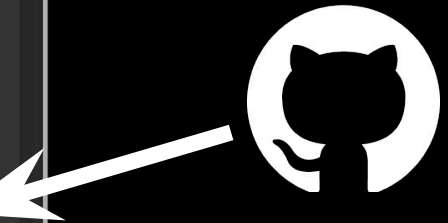
Doris

Bob


Node.js
nodejs.org/en
HOME
ABOUT
CATION
NEWS
Node.js® is a
nvironment.
Othe
Docs
For information about supported releases, see the release schedule.
Certificate Viewer: *.nodejs.org
General
Details
Certificate Hierarchy
USERTrust RSA Certification Authority
Sectigo RSA Domain Validation Secure Server CA
*.nodejs.org
Certificate Fields
*.nodejs.org
Certificate
Version
Serial Number
Certificate Signature Algorithm
Issuer
Validity
Not Before
Field Value
Export...


Keychain Access
USERTrust
Default Keychains
login
Local Items
System Keychains
System
System Roots
All Items
Passwords
Secure Notes
My Certificates
Keys
Certificates
USERTrust RSA Certification Authority
Root certificate authority
Expires: Tuesday, 19 January 2038 at 1:59:59 Israel Standard Time
This certificate is valid
Name
Kind
Expires
Keychain
USERTrust ECC Certification Authority
certificate
19 Jan 2038 at 1:59:59
System Roots
USERTrust RSA Certification Authority
certificate
19 Jan 2038 at 1:59:59
System Roots


Join us on the WiXapp
Mastering Cryptography Fundamentals with Node's crypto module
JSConf Japan 2023
Enter your Name
Generate your certificate
Code Samples
My blog
My Twitter



Cryptography Fundamentals

- Encryption
- Key Derivation Functions
- Randomness
- Key Distribution Problem \ Asymmetric Encryption
- Signing and Verifying
- Public Key Certificates


Join us on the WIXapp
Mastering Cryptography Fundamentals with Node's crypto module
JSConf Japan 2023
ジョン・ドウ
Generate your certificate
Code Samples
My blog
My Twitter

CERTIFICATE

OF ACHIEVEMENT

PROUDLY PRESENTED TO

ジョン・ドウ

For becoming a Cryptography expert

Public Key
for Verification

Personalized Cryptographic
Signature Based on Your Name

Verification
Instructions


Join us on the WiXapp
Mastering Cryptography Fundamentals with Node's crypto module
JSConf Japan 2023
Enter your Name
Generate your certificate
Code Samples
My blog
My Twitter