NODE.JS KRIPTOGRAFIYASI: KRIPTOGRAFIK AMALLARNI BAJARISH

Bahramov Muhammadali Eraliyevich
Alfraganus university
bahromov1997.04.09@gmail.com

Annotatsiya

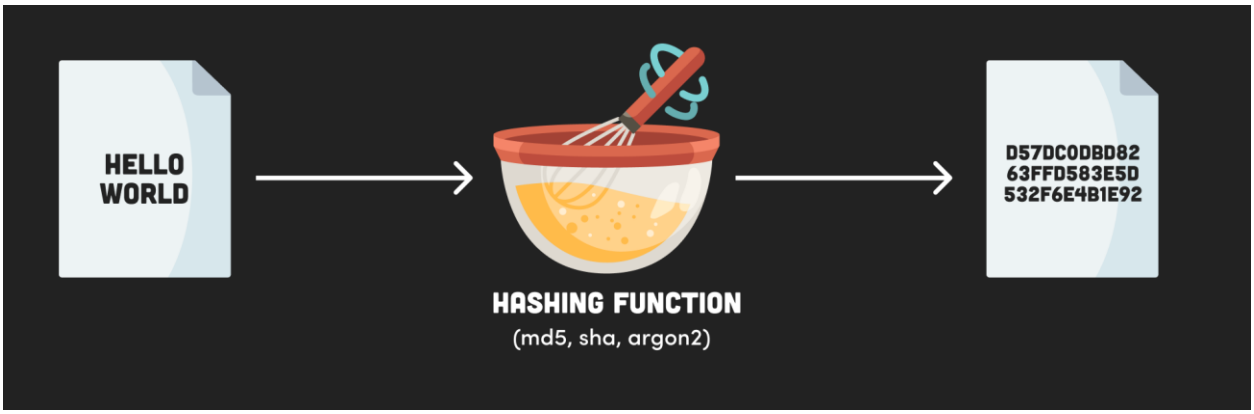
Ushbu maqola Node.jsda kriptografik amallarni bajarishning asosiy usullari va vositalarini batafsil tushuntiradi. Maqolada xashlash, shifrlash, HMAC (Hash-based Message Authentication Code), raqamli imzolar, salting (tuzlash) va xavfsiz tasodifiy sonlar yaratish kabi kriptografik amallar ko'rib chiqiladi. Har bir mavzu bo'yicha misollar keltirilib, Node.js crypto modulidan foydalanish usullari tushuntirilgan. Maqola dasturchilar uchun xavfsiz ma'lumotlarni saqlash va uzatish, shuningdek tizimlar va web-ilovalar xavfsizligini ta'minlashda zarur bo'lgan kriptografik amallarni o'rganishga qiziqqanlarga mo'ljallangan. Kriptografiyaning zamonaviy usullari yordamida ma'lumotlar xavfsizligini oshirish va autentifikatsiyani ta'minlash mumkinligini ko'rsatadi.

Kalit so’zlar: Node.js, Kriptografiya, crypto moduli, Hashing, Shifrlash (Encryption), Deshifrlash (Decryption), HMAC (Hash-based Message Authentication Code), Raqamli imzolar (Digital Signatures), Salting (Tuzlash), AES-256-CBC, SHA-256, Xavfsiz ma'lumotlar uzatish, Xavfsiz tasodifiy sonlar (Secure Random Numbers),Parolni himoyalash, Kriptografik xavfsizlik, Web xavfsizligi,Xavfsiz saqlash, Xavfsiz kalitlar. Digital Authentication, Data Integrity

Node.js Kriptografiyasi: Kriptografik Amallarni Bajarish

Kriptografiya veb-ilovalar va tizimlar xavfsizligini ta'minlashda muhim ahamiyatga ega. Bu turdagi texnologiyalar yordamida ma'lumotlarni shifrlash, hash qilish, raqamli imzolar yaratish va tasodifiy sonlar ishlab chiqarish mumkin. Node.js crypto moduli bu jarayonlarni sodda va samarali tarzda amalga oshirishga imkon beradi. Quyidagi maqolada siz Node.js yordamida kriptografik amallarni qanday bajarishni o'rganasiz.

1. Hashing (Xashlash)

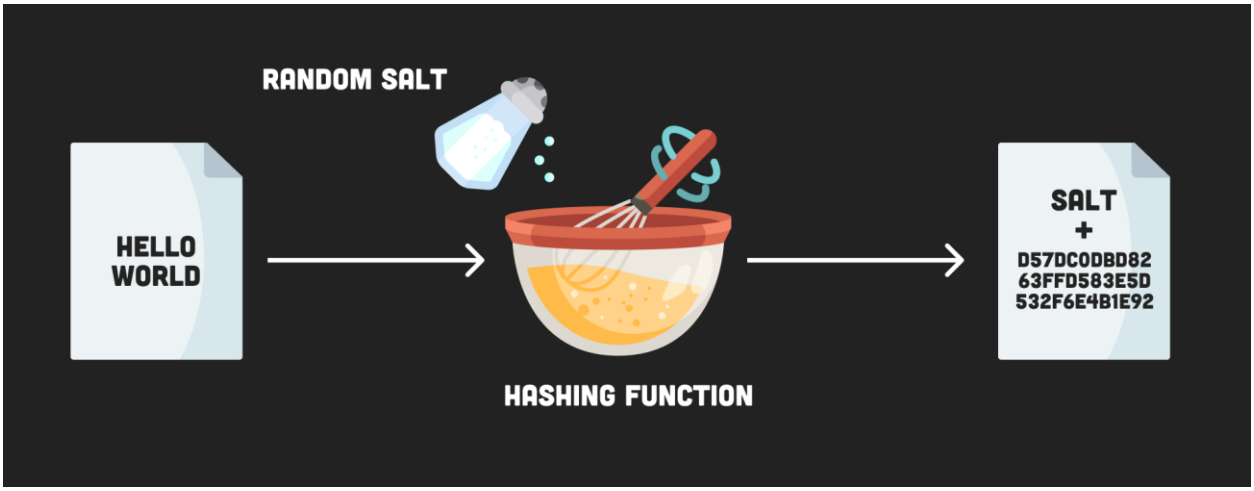


Hashing - bu ma'lumotni bir tomonlama kodlash jarayonidir. Hashing jarayonida, kiruvchi ma'lumot (masalan, parol) o'zgartirilgan holatda qaytariladi va original ma'lumotni qayta olish imkonsizdir. Bu jarayon odatda parollarni saqlashda ishlatiladi.

```
const crypto = require('crypto');
// Parolni xashlash
const password = 'mySecretPassword';
const hash = crypto.createHash('sha256').update(password).digest('hex');
console.log('Xashlangan parol:', hash);
```

Yuqoridagi kodda crypto.createHash() yordamida SHA-256 algoritmi bilan xashlash amalga oshiriladi. digest('hex') bu natijani o'qilishi oson bo'lgan formatda beradi.

2. Salting (Tuzlash)



Salting (tuzlash) - bu xashlash jarayonini kuchaytirish uchun qo'llaniladi. Tuzlash, hashning o'ziga xos noyob qiymatini qo'shib, xashning xavfsizligini oshiradi. Tuzlash natijasida har bir foydalanuvchi uchun yagona hash hosil qilinadi, hatto ular bir xil parolni ishlatgan bo'lsa ham.

```
const crypto = require('crypto');
// Tuzlash
const password = 'mySecretPassword';
const salt = crypto.randomBytes(16).toString('hex');  // 16 baytli tasodifiy tuz
const hash = crypto.createHash('sha256').update(password + salt).digest('hex');
console.log('Tuzlangan parol xashi:', hash);
console.log('Tuz:', salt);
```

Bu yerda biz avval tuz (salt) yaratamiz va uni parolga qo'shamiz. Tuzlangan parolni xashlash orqali xavfsizlikni kuchaytiramiz.

3. HMAC (Hash-based Message Authentication Code)

HMAC — bu xabarni autentifikatsiyalash va uning butunligini tekshirish uchun ishlatiladigan usul. HMAC ma'lum bir kalit va xash algoritmi yordamida xabarni autentifikatsiyalashni ta'minlaydi.

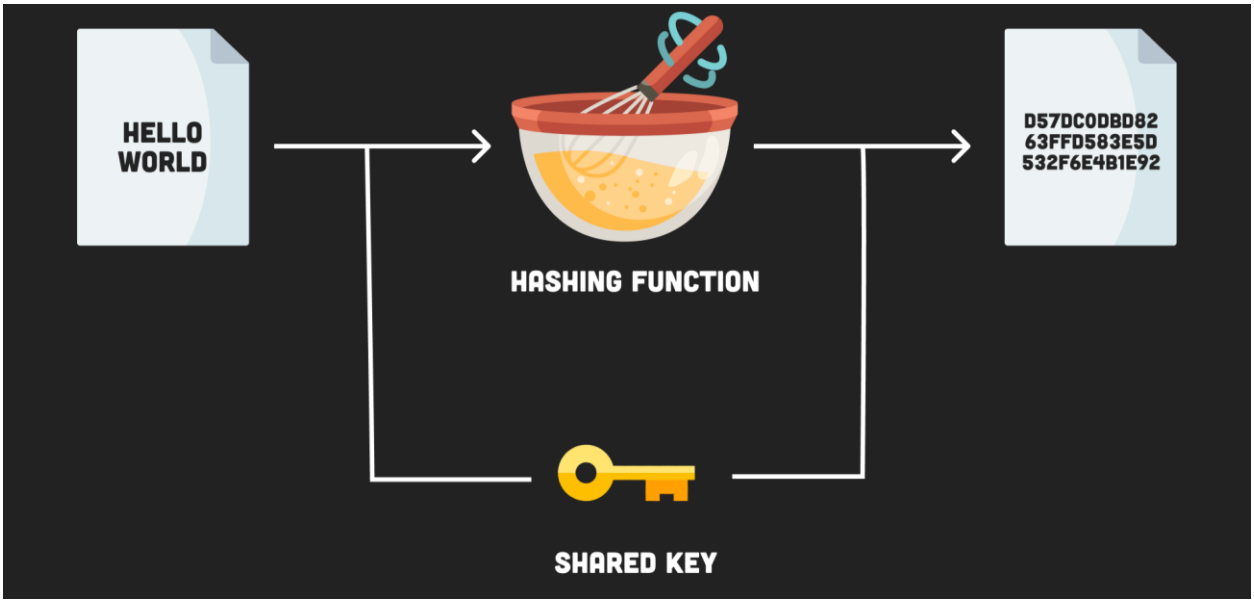
```
const crypto = require('crypto');

// Xabar va kalit
const message = 'Hello, World!';
const secretKey = 'mySecretKey';

// HMAC yaratish
const hmac = crypto.createHmac('sha256', secretKey).update(message).digest('hex');

console.log('HMAC:', hmac);
```

HMAC usuli xabarni va maxfiy kalitni birlashtirib, xabarni autentifikatsiyalash uchun ishlatiladi. Bu texnologiya ma'lumotlar uzatilganda, ularning butunligini tekshirish uchun juda foydalidir.



4. Shifrlash va Deshifrlash (Encryption va Decryption)

Shifrlash — bu ma'lumotlarni faqat maxsus kalit bilan o'qilishi mumkin bo'lgan tarzda kodlash jarayonidir. Node.jsda shifrlash va deshifrlashni amalga oshirish uchun crypto` moduli ishlatiladi.

```
const crypto = require('crypto');

// Shifrlash algoritmi va kalit
const algorithm = 'aes-256-cbc';
const key = crypto.randomBytes(32); // 32 baytli kalit
const iv = crypto.randomBytes(16);  // 16 baytli boshlang'ich vektor
// Shifrlash
const cipher = crypto.createCipheriv(algorithm, key, iv);
let encrypted = cipher.update('Secret Message', 'utf8', 'hex');
encrypted += cipher.final('hex');
console.log('Shifrlangan xabar:', encrypted);
// Deshifrlash
const decipher = crypto.createDecipheriv(algorithm, key, iv);
```

```
let decrypted = decipher.update(encrypted, 'hex', 'utf8');
decrypted += decipher.final('utf8');

console.log('Deshifrlangan xabar:', decrypted);
```

Bu misolda aes-256-cbc algoritmi yordamida xabar shifrlanadi va keyin deshifrlanadi. Kalit va boshlang'ich vektor (IV) har doim xavfsiz bo'lishi kerak.

5. Raqamli Imzolar (Digital Signatures)

Raqamli imzolar — bu ma'lumotlarni autentifikatsiyalash va ularning butunligini tasdiqlash uchun ishlatiladi. Raqamli imzolar xabarni shaxsiy kalit bilan imzolashni va umumiy kalit bilan tekshirishni ta'minlaydi.

```
const crypto = require('crypto');

// Asl xabar
const message = 'This is a secure message';

// Shaxsiy kalit
const privateKey = `-----BEGIN PRIVATE KEY-----
MIIBVwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSkAgEAAoIBAQ
Cbbk/5hMYhVHGz
...
-----END PRIVATE KEY-----`;
// Xabarni raqamli imzola
const sign = crypto.createSign('SHA256');
sign.update(message);
const signature = sign.sign(privateKey, 'hex');
console.log('Raqamli imzo:', signature);
// Imzoni tekshirish
const publicKey = `-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAASCAQ6gBWFlC9ecQUmti3ztv5fjw6+
5m5uDjqmf
...
-----END PUBLIC KEY-----`;
```

```
const verify = crypto.createVerify('SHA256');
verify.update(message);
const isVerified = verify.verify(publicKey, signature, 'hex');

console.log('Imzo tasdiqlangan:', isVerified);
```

Yuqoridagi kodda xabar shaxsiy kalit yordamida imzolanadi va umumiy kalit yordamida imzo tasdiqlanadi. Agar imzo to'g'ri bo'lsa, xabarni tasdiqlash mumkin.

6. Tasodifiy Sonlar (Random Numbers)

Kriptografiyada tasodifiy sonlar yaratish juda muhim. Node.jsda crypto` moduli yordamida xavfsiz tasodifiy sonlar yaratish mumkin. Bu sonlar xavfsizlikka bog'liq jarayonlarda, masalan, kalitlar yoki tuzlarni yaratishda ishlatiladi.

```
const crypto = require('crypto');

// Xavfsiz tasodifiy son
const randomBytes = crypto.randomBytes(16);  // 16 baytli tasodifiy son
console.log('Xavfsiz tasodifiy sonlar:', randomBytes.toString('hex'));
```

Bu yerda crypto.randomBytes() metodi yordamida xavfsiz tasodifiy sonlar hosil qilinadi. Ushbu sonlar masalan, xavfsiz kalitlar yoki session token'larini yaratish uchun ishlatiladi.

Xulosa

Node.js kriptografiya moduli yordamida ma'lumotlarni himoyalash va autentifikatsiya qilish uchun kerakli vositalar taqdim etadi. Yashirin ma'lumotlarni saqlash va uzatish jarayonida, kriptografik amallarni (xashlash, shifrlash, HMAC, raqamli imzolar va boshqalar) ishlatish juda muhimdir. Yuqoridagi misollar yordamida siz Node.jsda eng ko'p ishlatiladigan kriptografik amallarni o'rganishingiz mumkin.

Foydalanilgan adabiyotlar

1. Randell D. Wallace and Don F.Dagenais.The changing world of electronic signatures // https: www.library.lp.findlaw.com.
2. Craig Buckler Node.js: Novice to Ninja
3. Google.com
4. https://nodejs.org/en
5. https://itoolkit.co/blog/2023/08/what-are-hashing-salting-algorithms/#:~:text=In%20cryptography%2C%20salting%20refers%20to,number%20generator%20for%20creating%20salts