



**TEC.AZUAY**  
INSTITUTO UNIVERSITARIO

# **APLICACIONES SEGURAS**

## **IMPLEMENTACIÓN Y ANÁLISIS DE TÉCNICAS DE ENCRIPTACIÓN**

**Criptografía de Flujo (Stream Cipher)**

Prepared By :

- **Milton Avila**
- **Karla Mendez**
- **Jose Morocho**
- **Bryam Romero**
- **Luis Rey**
- **Oscar Zurita**

N6A

24/4/2025

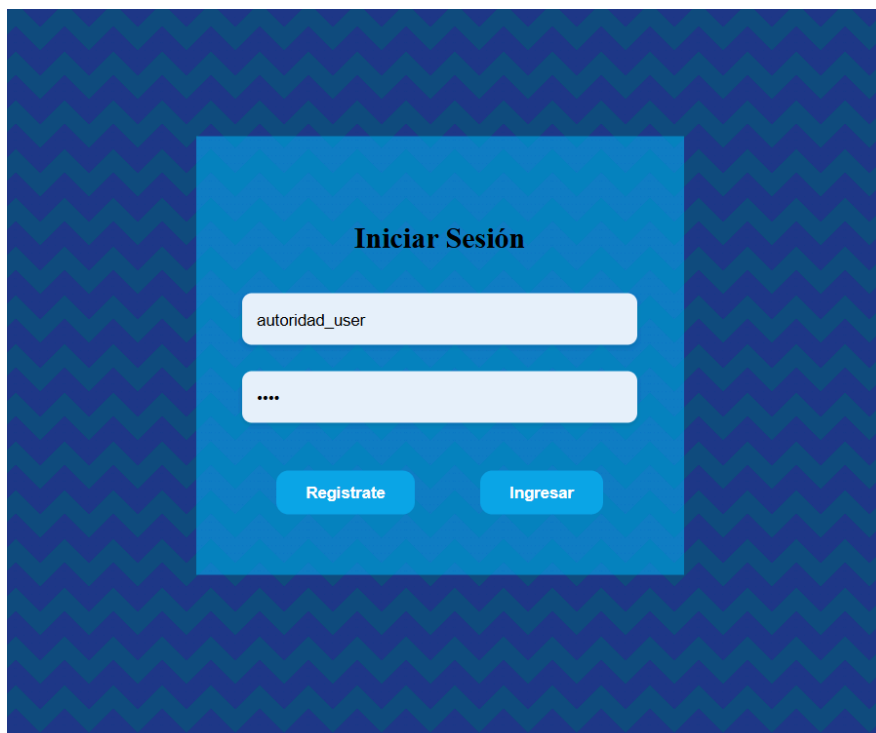
En este ejercicio implementamos una aplicación de mensajería segura utilizando el algoritmo de Criptografía de Flujo ChaCha20. Este tipo de cifrado se caracteriza por trabajar con datos en tiempo real, encriptando bit a bit o byte a byte, lo que lo hace ideal para proteger mensajes mientras son transmitidos.

El objetivo fue asegurar que los mensajes enviados entre usuarios estén cifrados antes de ser almacenados en la base de datos, y que solo puedan ser descifrados por el sistema cuando el receptor los consulta. Para lograrlo, se usó el algoritmo ChaCha20 en el backend, desarrollado con Node.js, Express y MongoDB.

Se construyó una API REST que permite:

- Registrar usuarios con autenticación mediante JWT.
- Enviar mensajes cifrados con ChaCha20.
- Consultar mensajes y descifrarlos correctamente.

Este ejercicio no solo demuestra la utilidad del cifrado de flujo en la protección de datos, sino que también refuerza buenas prácticas en la seguridad de aplicaciones, como la validación de entradas, autenticación y manejo de claves secretas.



```

import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { Router } from '@angular/router'; // Importar Router

@Component({
  selector: 'app-login',
  imports: [FormsModule, HttpClientModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'] // Corregí 'styleUrl' a 'styleUrls'
})
export class LoginComponent {
  username: string = '';
  password: string = '';
  publicKey: string = '';

  constructor(private http: HttpClient, private router: Router) {} // Inyectar Router

  login(): void {
    if (!this.username || !this.password) {
      console.error('El nombre de usuario y la contraseña son necesarios');
      return;
    }

    const payload = {
      username: this.username,
      password: this.password
    };

    this.http.post('http://localhost:3000/api/auth/login', payload).subscribe(
      response => {
        console.log('Inicio de sesión exitoso', response);
        this.router.navigate(['/mensaje']); // Aquí envía al form del mensaje
      },
      error => {
        console.error('Error al ingresar', error);
      }
    );
  }
}

```

```

<div class="contenedor">
  <div class="contenido">
    <h2 class="iniciar_sesion">Iniciar Sesión</h2>

    <input [(ngModel)]="username" placeholder="Correo electrónico" />
    <input [(ngModel)]="password" type="password" placeholder="Contraseña" />

    <div class="botones-horizontal">
      <!-- Boton de registrar -->
      <button
        (click)="registrar()"
        class="btn"
      >Registrate</button>
      <button
        (click)="login()"
        class="btn"
      >
        Ingresar
      </button>
    </div>
  </div>
</div>

```

```

export class RegistroComponent {
  register(): void {
    if (!this.publicKey) {
      console.error('La clave pública es necesaria');
      return;
    }

    const payload = {
      username: this.username,
      password: this.password,
      publicKey: this.publicKey // Incluye la clave pública en el payload
    };

    this.http.post('http://localhost:3000/api/auth/register', payload).subscribe(
      response => {
        console.log('Usuario registrado', response);
        this.router.navigate(['/login']); // Redirige al login después del registro
      },
      error => {
        console.error('Error en el registro', error);
      }
    );
  }
}

```

## Registro de Usuario

Completa los campos para crear tu cuenta

```

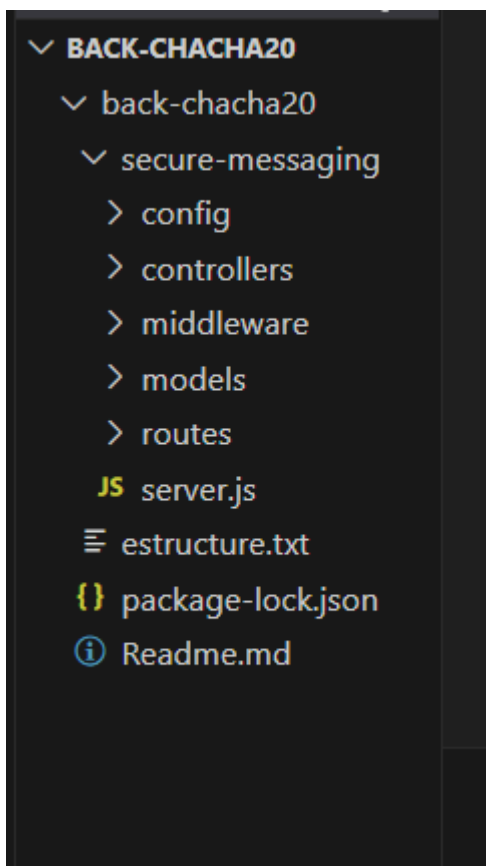
1  import { Component, OnInit } from '@angular/core';
2  import { MessageService } from '../../services/message.service';
3  import { CommonModule } from '@angular/common'; // Importar CommonModule
4  import { FormsModule } from '@angular/forms'; // Importar FormsModule
5
6  @Component({
7    selector: 'app-mensajes',
8    standalone: true, // Indicar que es un componente standalone
9    imports: [CommonModule, FormsModule], // Incluir CommonModule y FormsModule
10   templateUrl: './mensaje.component.html',
11   styleUrls: ['./mensaje.component.css']
12 })
13 export class MensajesComponent implements OnInit {
14   mensaje: string = '';
15   mensajes: any[] = [];
16
17   constructor(private MessageService: MessageService) {}
18
19   ngOnInit(): void {
20     this.obtenerMensajes();
21   }
22
23   enviar(): void {
24     if (!this.mensaje) return;
25     this.MessageService.sendMessage(this.mensaje).subscribe(() => {
26       this.mensaje = '';
27       this.obtenerMensajes();
28     });
29   }
30
31   obtenerMensajes(): void {
32     this.MessageService.getMessages().subscribe((data: any) => {
33       this.mensajes = data;
34     });
35   }
36 }

```

## BACK END

Tecnologías utilizadas:

- Node.js + Express (servidor backend)
- MongoDB (almacenamiento de usuarios y mensajes)
- Mongoose (modelado de datos)
- @stablelib/chacha20poly1305 (cifrado seguro ChaCha20)
- JWT (autenticación de usuarios)
- bcryptjs (hashing de contraseñas)



Componentes desarrollados:

1. Modelo de usuario (models/User.js)

- username: único
- password: encriptada con bcrypt
- publicKey: (campo opcional para firma/asimetría futura)

```
back-chacha20 > secure-messaging > models > JS User.js > ...
1  const mongoose = require('mongoose');
2
3  const UserSchema = new mongoose.Schema({
4    username: { type: String, required: true, unique: true },
5    password: { type: String, required: true },
6    publicKey: { type: String, required: true },
7  });
8
9  module.exports = mongoose.model('User', UserSchema);
10
```

2. Modelo de mensaje (models/Message.js)

- sender: usuario emisor
- receiver: usuario receptor
- content: mensaje cifrado
- nonce: valor aleatorio necesario para descifrar
- timestamp: fecha de envío

```
back-chacha20 > secure-messaging > models > JS Message.js > ...
1  const mongoose = require('mongoose');
2
3  const MessageSchema = new mongoose.Schema({
4    sender: { type: String, required: true },
5    receiver: { type: String, required: true },
6    content: { type: String, required: true }, // mensaje cifrado
7    nonce: { type: String, required: true }, // nonce para descifrar con ChaCha20
8    timestamp: { type: Date, default: Date.now },
9  });
10
11  module.exports = mongoose.model('Message', MessageSchema);
12
```

### 3. Controlador de autenticación (controllers/AuthController.js)

- Registro:

- Valida que el nombre de usuario no exista
- Cifra la contraseña con bcrypt
- Guarda el usuario

- Login:

- Verifica credenciales
- Devuelve un token JWT válido si la contraseña coincide



```

back-chacha20 > secure-messaging > controllers > JS AuthController.js > register > register > newUser
1  const bcrypt = require('bcryptjs');
2  const jwt = require('jsonwebtoken');
3  const User = require('../models/User');
4
5  // Registro de usuario
6  exports.register = async (req, res) => {
7    try {
8      const { username, password, publicKey } = req.body;
9
10     // Validar que todos los campos estén presentes
11     if (!username || !password || !publicKey) {
12       return res.status(400).json({ error: 'Todos los campos son obligatorios' });
13     }
14
15     // Verificar si el usuario ya existe
16     const existingUser = await User.findOne({ username });
17     if (existingUser) {
18       return res.status(400).json({ error: 'El nombre de usuario ya existe' });
19     }
20
21     // Encriptar la contraseña
22     const hashedPassword = await bcrypt.hash(password, 10);
23
24     // Crear nuevo usuario
25     const newUser = new User({
26       username,
27       password: hashedPassword,
28       publicKey,
29     });

```

#### 4. Controlador de mensajes (controllers/MessageController.js)

- sendMessage:
  - Verifica que el receptor exista
  - Genera nonce aleatorio
  - Cifra el mensaje con ChaCha20 usando key fija y el nonce
  - Guarda el mensaje cifrado y el nonce
- getMessages:
  - Recupera los mensajes de un usuario
  - Descifra cada mensaje con su nonce correspondiente
  - Devuelve los mensajes en texto plano al usuario autenticado

```

back-chacha20 > secure-messaging > controllers > JS MessageController.js > decryptMessage
1  const Message = require('../models/Message'); // ← Corregido aquí
2  const User = require('../models/User');
3  const { ChaCha20Poly1305 } = require('@stablelib/chacha20poly1305');
4  const { randomBytes } = require('crypto');
5
6  // Encriptar mensaje con ChaCha20-Poly1305
7  function encryptMessage(plainText, key) {
8      const nonce = randomBytes(12); // 96 bits recomendados
9      const chacha = new ChaCha20Poly1305(key);
10     const ciphertext = chacha.seal(nonce, Buffer.from(plainText));
11     return {
12         nonce: nonce.toString('hex'),
13         encrypted: Buffer.from(ciphertext).toString('hex')
14     };
15 }
16
17 // Desencriptar mensaje
18 function decryptMessage(encryptedMessageHex, nonceHex, key) {
19     const chacha = new ChaCha20Poly1305(key);
20     const decrypted = chacha.open(
21         Buffer.from(nonceHex, 'hex'),
22         Buffer.from(encryptedMessageHex, 'hex')
23     );
24     return decrypted ? Buffer.from(decrypted).toString() : null;
25 }
26
27 // Enviar mensaje
28 exports.sendMessage = async (req, res) => {
29     try {
30         const { sender, receiver, message } = req.body;

```

Iniciando Mongo

```
20 | USER Name,
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20> cd back-chacha20
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20> cd secure-messaging
>>
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20\secure-messaging> mongod
{"t":{"$date":"2025-04-24T14:49:27.736-05:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"thread1","msg":"Automatically disabling TLS 1.0, to force-enable
S 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2025-04-24T14:49:30.738-05:00"},"s":"I", "c":"CONTROL", "id":5945603, "ctx":"thread1","msg":"Multi threading initialized"}
{"t":{"$date":"2025-04-24T14:49:30.758-05:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"thread1","msg":"Implicit TCP FastOpen unavailable. If TCP FastOp
is required, set at least one of the related parameters", "attr":{"relatedParameters":{"tcpFastOpenServer","tcpFastOpenClient","tcpFastOpenQueueSize"}}}
{"t":{"$date":"2025-04-24T14:49:30.789-05:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"thread1","msg":"Initialized wire specification", "attr":{"spec":{"
comingExternalClient":{"minWireVersion":0,"maxWireVersion":25},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":25},"outgoing":{"minWireVersion":
maxWireVersion":25},"isInternalClient":true}}}
{"t":{"$date":"2025-04-24T14:49:30.797-05:00"},"s":"I", "c":"TEVANT_M", "id":7091600, "ctx":"thread1","msg":"Starting TenantMigrationAccessBlockerRegistry"}
{"t":{"$date":"2025-04-24T14:49:30.798-05:00"},"s":"I", "c":"CONTROL", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting", "attr":{"pid":26748,"por
27017,"dbPath":"/data/db","architecture":"64-bit","host":"DESKTOP-9JUT0AH"}}
{"t":{"$date":"2025-04-24T14:49:30.798-05:00"},"s":"I", "c":"CONTROL", "id":23398, "ctx":"initandlisten","msg":"Target operating system minimum version", "
r":{"targetMinOS":"Windows 7/Windows Server 2008 R2"}}
{"t":{"$date":"2025-04-24T14:49:30.799-05:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build Info", "attr":{"buildInfo":{"version"
.0.4","gitVersion":"bc35ab4305d9920d9d0491c1c9ef9b72383d31f9","modules":[],"allocator":"tcmalloc-gperf","environment":{"distmod":"windows","distarch":"x86_64"
arget_arch":"x86_64"}}}}
{"t":{"$date":"2025-04-24T14:49:30.799-05:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System", "attr":{"os":{"name":"Mi
soft Windows 10","version":"10.0 (build 26100)}}}
{"t":{"$date":"2025-04-24T14:49:30.799-05:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command line", "attr":{"opti
":{}}}
{"t":{"$date":"2025-04-24T14:49:30.808-05:00"},"s":"I", "c":"STORAGE", "id":22270, "ctx":"initandlisten","msg":"Storage engine to use detected by data fil

ckpoint timestamp: (0, 0) base write gen: 1017}}}}
{"t":{"$date":"2025-04-24T16:43:34.428-05:00"},"s":"I", "c":"WCHKPT", "id":22430, "ctx":"Checkpointner","msg":"WiredTiger message", "attr":{"message":{"ts sec
":1745531014,"ts_usec":428168,"thread":"26748:140731206305312","session name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"v
erbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 117, snapshot max: 117 snapshot count: 0, oldest timestamp: (0, 0) , meta che
ckpoint timestamp: (0, 0) base write gen: 1017}}}}
{"t":{"$date":"2025-04-24T16:44:34.466-05:00"},"s":"I", "c":"WCHKPT", "id":22430, "ctx":"Checkpointner","msg":"WiredTiger message", "attr":{"message":{"ts sec
":1745531074,"ts_usec":466068,"thread":"26748:140731206305312","session name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"v
erbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 118, snapshot max: 118 snapshot count: 0, oldest timestamp: (0, 0) , meta che
ckpoint timestamp: (0, 0) base write gen: 1017}}}}
{"t":{"$date":"2025-04-24T16:45:34.484-05:00"},"s":"I", "c":"WCHKPT", "id":22430, "ctx":"Checkpointner","msg":"WiredTiger message", "attr":{"message":{"ts sec
":1745531134,"ts_usec":483952,"thread":"26748:140731206305312","session name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"v
erbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 119, snapshot max: 119 snapshot count: 0, oldest timestamp: (0, 0) , meta che
ckpoint timestamp: (0, 0) base write gen: 1017}}}}
{"t":{"$date":"2025-04-24T16:46:34.509-05:00"},"s":"I", "c":"WCHKPT", "id":22430, "ctx":"Checkpointner","msg":"WiredTiger message", "attr":{"message":{"ts sec
":1745531194,"ts_usec":508946,"thread":"26748:140731206305312","session name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"v
erbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 120, snapshot max: 120 snapshot count: 0, oldest timestamp: (0, 0) , meta che
ckpoint timestamp: (0, 0) base write gen: 1017}}}}
{"t":{"$date":"2025-04-24T16:47:34.534-05:00"},"s":"I", "c":"WCHKPT", "id":22430, "ctx":"Checkpointner","msg":"WiredTiger message", "attr":{"message":{"ts sec
":1745531254,"ts_usec":533969,"thread":"26748:140731206305312","session name":"WT_SESSION.checkpoint","category":"WT_VERB_CHECKPOINT_PROGRESS","category_id":7,"v
erbose_level":"DEBUG_1","verbose_level_id":1,"msg":"saving checkpoint snapshot min: 121, snapshot max: 121 snapshot count: 0, oldest timestamp: (0, 0) , meta che
ckpoint timestamp: (0, 0) base write gen: 1017}}}}
[]
```

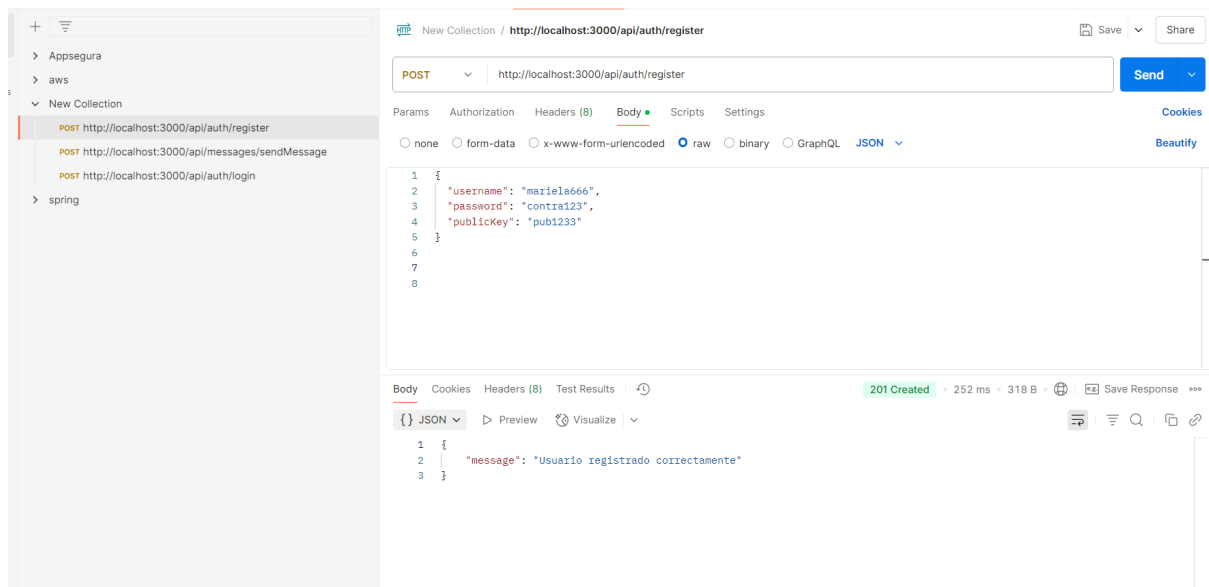
## Iniciando node

```
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20> cd back-chacha20
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20> cd secure-messaging
>>
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20\secure-messaging> node server.js
Server running on port 3000
(node:15280) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
MongoDB connected
[]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20> cd back-chacha20
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20> cd secure-messaging
>>
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20\secure-messaging> node server.js
Server running on port 3000
(node:15280) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
MongoDB connected
█
```

## PRUEBA DE ENDPOINTS EN POSTMAN - registro - login - mensajes





- Los mensajes no quedan en texto plano en la base de datos.
- Cada mensaje tiene su propia clave de flujo (nonce).
- Solo usuarios autenticados pueden enviar y leer mensajes.
- Técnica usada en protocolos como TLS 1.3.

#### **LINK DE LOS REPOSITORIOS**

<https://github.com/RomaRomero/encrypt-chacha20.git>

<https://github.com/Oscar2876/frontappsegura.git>