# Implementación de ECDSA y ECDH para Comunicación Segura Cliente-Servidor

Norman Cortés, Yohanns Jara, Kevin Lucas, Juan Villalón
Equipo 4: "Cryptoknights"
Departamento de Eletrónica
Ingeniería Civil Telemática
Universidad Técnica Federico Santa María
Valparaíso, Chile
*GitHub del proyecto.*
*Video demostrativo youtube*
*norman.cortes@usm.cl*, *yohanns.jara@sansano.usm.cl*,
*kevin.lucas@usm.cl*, *juan.villalon@sansano.usm.cl*

***Resumen*—Este proyecto explora la implementación de ECDSA, ECDH y AES en un sistema de comunicación cliente-servidor para garantizar confidencialidad, integridad y autenticidad. La integración con bases de datos y técnicas avanzadas de seguridad como contraseñas cifradas con SHA256 y mitigación de inyecciones SQL asegura la escalabilidad y robustez del sistema.**

## I. Introducción

La comunicación segura en sistemas distribuidos es uno de los pilares fundamentales para el desarrollo de aplicaciones modernas en un entorno digitalizado. En un mundo donde los datos se han convertido en uno de los activos más valiosos, la necesidad de garantizar su integridad, confidencialidad y autenticidad es esencial, especialmente en áreas críticas como el Internet de las Cosas (IoT), sistemas financieros, redes de telecomunicaciones y aplicaciones de control industrial.

Este proyecto busca implementar una solución robusta de comunicación segura entre cliente y servidor mediante el uso de algoritmos criptográficos avanzados como ECDSA (Elliptic Curve Digital Signature Algorithm), ECDH (Elliptic Curve Diffie-Hellman) y AES (Advanced Encryption Standard). Cada uno de estos algoritmos juega un papel clave en la protección de los datos transmitidos: ECDSA asegura la integridad y autenticidad de la información, ECDH permite un intercambio seguro de claves, y AES protege los datos mediante cifrado simétrico. La integración de estas tecnologías garantiza un enfoque integral que cumple con las demandas de seguridad actuales.

El sistema desarrollado se implementó en un entorno local, lo que permitió probar y evaluar todos los componentes en un ambiente controlado. Aunque el alcance inicial se limitó a un entorno de pruebas, este proyecto establece las bases para futuras implementaciones en escenarios reales que involucran sensores distribuidos, conectividad a la nube y requisitos de alto rendimiento.

### *I-A. Contexto de Aplicación*

El sistema diseñado tiene múltiples aplicaciones en diversos escenarios donde la comunicación segura es fundamental. Algunos de estos incluyen:

- **Sensores IoT:** Dispositivos conectados que recopilan datos críticos como temperatura, humedad o ubicación geográfica y los transmiten a un servidor central para análisis. Por ejemplo, en la agricultura de precisión, sensores distribuidos monitorean las condiciones del suelo y el clima para optimizar los cultivos.
- **Sistemas Financieros:** Transacciones bancarias y datos de usuarios protegidos mediante firmas digitales y cifrado avanzado para evitar fraudes y accesos no autorizados.
- **Redes de Telecomunicaciones:** Intercambio seguro de información entre nodos de red, asegurando que los datos no sean interceptados ni manipulados durante la transmisión.
- **Monitoreo Industrial:** Sensores que supervisan maquinaria crítica en tiempo real, transmitiendo datos cifrados a servidores centrales para prevenir fallos y mejorar la eficiencia.

### *I-B. Motivación del Proyecto*

Con el crecimiento exponencial del IoT y la cantidad de dispositivos conectados, la protección de la información transmitida entre sensores y servidores se ha vuelto una preocupación prioritaria. Este proyecto busca demostrar que es posible implementar un sistema seguro, escalable y eficiente utilizando tecnologías modernas de criptografía. Adicionalmente, la elección de tecnologías como Vite-React para el cliente, Node.js Express para el servidor y una base de datos sql refleja la intención de utilizar herramientas ampliamente adoptadas en la industria, facilitando su integración y mantenimiento.

En este proyecto se destacan dos componentes principales:

- **Pruebas Locales:** Permiten validar la funcionalidad del sistema en un entorno controlado, midiendo tiempos de ejecución, validación de firmas y rechazo de datos alterados.

- **Escalabilidad Potencial:** El diseño modular del sistema facilita su implementación en un entorno distribuido, donde múltiples sensores en diferentes ubicaciones transmiten datos a un servidor central, con capacidad de expansión a la nube para análisis en tiempo real.

## II. Estado del Arte

En el ámbito de la criptografía moderna, los algoritmos y herramientas utilizados en este proyecto representan estándares ampliamente adoptados para garantizar la seguridad en sistemas distribuidos. Esta sección detalla los fundamentos teóricos y las tecnologías empleadas para la implementación.

### *II-A. Fundamentos Criptográficos*

*II-A1. ECDSA: Elliptic Curve Digital Signature Algorithm:* ECDSA es un algoritmo basado en curvas elípticas que garantiza la integridad y autenticidad de los datos mediante firmas digitales [1]. Es ampliamente adoptado en sistemas como blockchain, donde asegura que las transacciones no sean manipuladas. En este proyecto, ECDSA se utilizó para firmar y verificar datos críticos en la comunicación cliente-servidor.

*II-A2. ECDH: Elliptic Curve Diffie-Hellman:* ECDH es un protocolo de intercambio de claves que permite a dos partes derivar un secreto compartido sin necesidad de un canal seguro inicial. Este secreto es empleado para cifrar los datos transmitidos. En el sistema desarrollado, tanto el cliente como el servidor generan pares de claves mediante la curva `secp256k1`, ampliamente utilizada por su seguridad y eficiencia.

*II-A3. AES: Advanced Encryption Standard:* AES es un algoritmo de cifrado simétrico que proporciona un alto rendimiento y un nivel robusto de seguridad [2]. En este proyecto, se utilizó para proteger los datos transmitidos entre cliente y servidor utilizando el secreto compartido derivado por ECDH.

### *II-B. Librerías y Herramientas Empleadas*

Para implementar estos algoritmos y construir un sistema funcional, se utilizaron múltiples librerías y herramientas tecnológicas, que se describen a continuación:

*II-B1. Backend: Node.js:* El servidor fue desarrollado utilizando Node.js, un entorno de ejecución de JavaScript conocido por su eficiencia en aplicaciones de red. Las siguientes librerías fueron esenciales para la implementación:

- `express`: Framework minimalista que facilita la creación de servidores web.
- `body-parser`: Middleware para analizar las solicitudes entrantes, permitiendo trabajar con datos JSON.
- `cors`: Librería para habilitar el intercambio de recursos de origen cruzado, asegurando que las solicitudes entre cliente y servidor sean permitidas.
- `elliptic`: Implementación robusta de curvas elípticas para generación de claves y firmas (ECDSA y ECDH).
- `crypto-js`: Herramienta para cifrado y descifrado AES en el servidor.
- `pg`: Cliente para interactuar con bases de datos PostgreSQL, aunque en pruebas locales se utilizó una configuración inicial con MySQL.

Ejemplo de inicialización del servidor y configuración de ECDH:

```
// Importar librerias
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { ec: EC } = require('elliptic');
const CryptoJS = require('crypto-js'); //
    Importar para cifrado AES

// Inicializar servidor y curva eliptica
const ec = new EC('secp256k1');
const app = express();

// Generar clave publica del servidor (ECDH)
const serverECDHKey = ec.genKeyPair();
const serverPublicKey = serverECDHKey.getPublic(
    'hex');

// Configurar CORS y body-parser
app.use(cors());
app.use(bodyParser.json());

// Endpoint: Clave publica del servidor
app.get('/ecdh/public-key', (req, res) => {
  res.json({ serverPublicKey });
});
```

Listing 1. Inicializacion del Servidor con Node.js y express

*II-B2. Frontend: React y Vite:* El cliente fue desarrollado utilizando React, una biblioteca de JavaScript para construir interfaces de usuario [3], [4]. Se configuró con Vite como herramienta de desarrollo, conocida por su velocidad en la construcción de proyectos modernos. Las tecnologías clave incluyeron:

- `React`: Para la construcción de componentes reutilizables y dinámicos.
- `Vite`: Herramienta rápida para el empaquetado y desarrollo de proyectos React.
- `JSX`: Sintaxis que combina JavaScript y HTML, utilizada para definir la interfaz de usuario.
- `Tailwind CSS`: Framework de CSS utilizado para estilizar la aplicación con una sintaxis limpia y eficiente.

Ejemplo de configuración de claves y comunicación desde el cliente:

```
// Importar React y elliptic
import React, { useState } from 'react';
import { ec as EC } from 'elliptic';

const ec = new EC('secp256k1');

function App() {
  const [clientKey, setClientKey] = useState(
    null);
  const [serverPublicKey, setServerPublicKey] =
    useState('');

  // Generar clave ECDH
  const generateKeyPair = () => {
    const keyPair = ec.genKeyPair();
    setClientKey(keyPair);
    console.log('Clave Publica Cliente:',
    keyPair.getPublic('hex'));
  };

  // Solicitar clave publica del servidor
  const fetchServerPublicKey = async () => {
```

```
    const response = await fetch('http://localhost:3000/ecdh/public-key');
    const data = await response.json();
    setServerPublicKey(data.serverPublicKey);
    console.log('Clave Publica Servidor:', data.serverPublicKey);
  };

  return (
    <div>
      <button onClick={generateKeyPair}>Generar Clave Cliente</button>
      <button onClick={fetchServerPublicKey}>Obtener Clave Servidor</button>
    </div>
  );
}
```

Listing 2. Generacion de claves ECDH y comunicacion con el servidor

*II-B3. Base de Datos y Seguridad:* Aunque en pruebas locales se utilizó MySQL, la estructura del proyecto permite la integración con bases de datos como PostgreSQL. Además, se implementaron las siguientes prácticas de seguridad:

- `bcrypt`: Para la generación de hashes seguros de contraseñas.
- `JWT (JSON Web Token)`: Para autenticación basada en tokens, permitiendo sesiones seguras y evitando el almacenamiento de credenciales sensibles en el cliente.

### *II-C. Justificación de las Tecnologías*

La combinación de estas herramientas y librerías se eligió por las siguientes razones:

- **Eficiencia:** React y Vite ofrecen un entorno de desarrollo rápido y flexible, mientras que Node.js asegura un backend no bloqueante y escalable.
- **Compatibilidad:** Las librerías como `express` y `elliptic` son estándares en la industria, facilitando futuras integraciones.
- **Seguridad:** La implementación de algoritmos avanzados como ECDSA, ECDH y AES, junto con herramientas de seguridad como `bcrypt` y `JWT`, asegura un sistema robusto y protegido contra vulnerabilidades comunes.

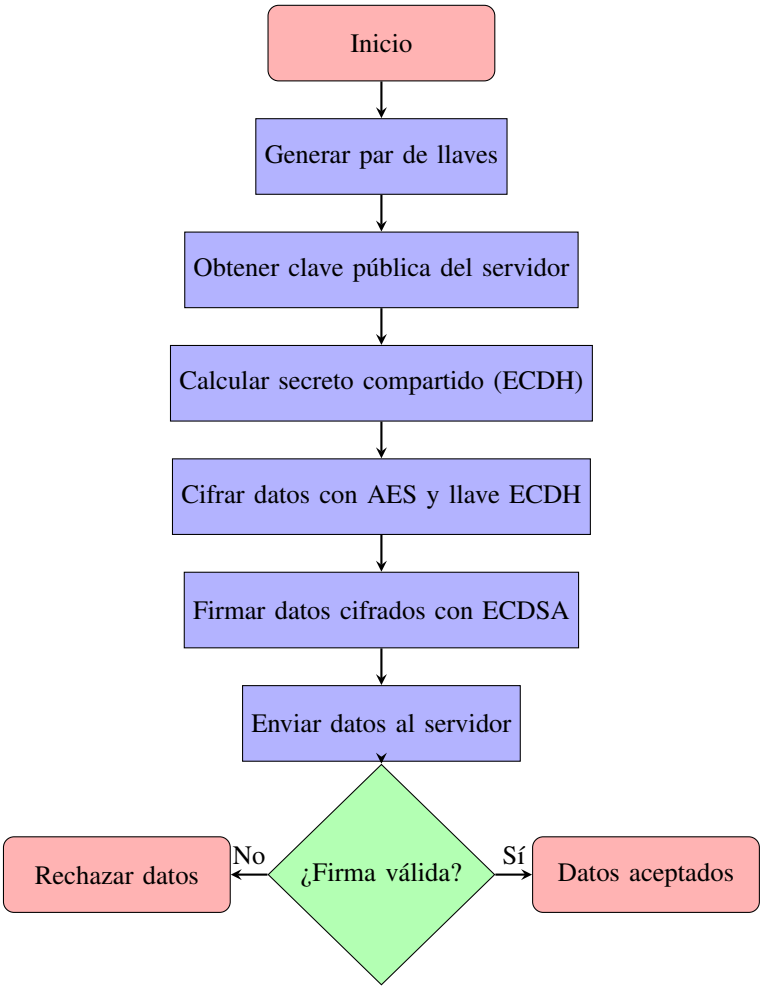
## III. Metodología

El sistema cliente-servidor se desarrolló bajo un enfoque modular, donde cada componente cumple una función específica en la arquitectura general. La metodología implementada se divide en las siguientes etapas principales:

### *III-A. Arquitectura General*

La arquitectura del sistema consta de un cliente desarrollado en React y un servidor desarrollado en Node.js [5]. La interacción entre ambos se basa en la comunicación HTTP mediante endpoints RESTful. El sistema fue diseñado para ser escalable y adaptable a entornos distribuidos más complejos, como IoT o aplicaciones en la nube.

La Figura muestra el flujo de trabajo, desde la generación de claves hasta el almacenamiento de los datos en la base de datos.




### *III-B. Etapas del Flujo de Trabajo*

El flujo de trabajo del sistema se divide en las siguientes etapas:

1. **Generación de Claves:** Tanto el cliente como el servidor generan pares de llaves utilizando la curva `secp256k1`.
2. **Intercambio de Claves:** El cliente solicita la clave pública del servidor mediante un endpoint REST.
3. **Cálculo del Secreto Compartido:** Ambos componentes derivan una llave simetrica compartida a partir de sus claves privadas propias y las claves públicas del otro.
4. **Cifrado de Datos:** Los datos del sensor se cifran utilizando AES con el secreto compartido como clave.
5. **Firma Digital:** Los datos cifrados se firman utilizando ECDSA con la llave privada para garantizar su integridad y autenticidad.
6. **Validación en el Servidor:** El servidor valida la firma con la llave publica, descifra los datos con la llave simetrica ECDH y el vector iv para luego almacenar en la base de datos.

### III-C. *Módulos y Componentes*

*III-C1. Módulo Cliente (Frontend):* El cliente, desarrollado en React, genera claves ECDH, solicita la clave pública del servidor y envía datos cifrados y firmados. El uso de Tailwind CSS permite una interfaz limpia y funcional.

Ejemplo de generación de claves y cálculo del secreto compartido:

```
const ec = new EC('secp256k1');

// Generar clave del cliente
const clientKey = ec.genKeyPair();
console.log(`Clave Publica Cliente: ${clientKey.
    getPublic('hex')}`);

// Derivar el secreto compartido tras recibir la
     clave publica del servidor
const sharedSecret = clientKey.derive(
    serverPublicKey).toString(16);
console.log(`Secreto Compartido: ${sharedSecret
    }`);
```

Listing 3. Generación de claves y cálculo del secreto compartido

*III-C2. Módulo Servidor (Backend):* El servidor, desarrollado en Node.js, maneja las solicitudes de los clientes, valida firmas digitales y descifra los datos utilizando el secreto compartido. Además, el servidor expone un endpoint para proporcionar su clave pública.

Configuración básica del servidor:

```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const { ec: EC } = require('elliptic');

const ec = new EC('secp256k1');
const app = express();

// Generar clave del servidor
const serverECDHKey = ec.genKeyPair();
const serverPublicKey = serverECDHKey.getPublic(
    'hex');

// Configurar middlewares
app.use(cors());
app.use(bodyParser.json());

// Endpoint para obtener la clave publica del
    servidor
app.get('/ecdh/public-key', (req, res) => {
  res.json({ serverPublicKey });
});
```

Listing 4. Configuración inicial del servidor con Express

*III-C3. Cifrado y Descifrado con AES:* El cifrado y descifrado se realizaron utilizando la librería `CryptoJS`, tanto en el cliente como en el servidor. El secreto compartido derivado de ECDH sirve como clave para cifrar los datos del sensor.

Ejemplo de cifrado:

```
const CryptoJS = require('crypto-js');
const data = JSON.stringify({ id: 1, temperatura
    : 22, latitud: -33.4489, altitud: 500 });

// Cifrar datos
const ciphertext = CryptoJS.AES.encrypt(data,
    sharedSecret).toString();
console.log(`Datos Cifrados: ${ciphertext}`);
```

Listing 5. Cifrado de datos con AES

Ejemplo de descifrado en el servidor:

```
const decryptedData = CryptoJS.AES.decrypt(
    ciphertext, sharedSecret).toString(CryptoJS.
    enc.Utf8);
console.log(`Datos Descifrados: ${decryptedData
    }`);
```

Listing 6. Descifrado de datos con AES

*III-C4. Base de Datos:* La información descifrada se almacena en una base de datos MySQL en la prueba local, pero la arquitectura permite su integración con otros sistemas como PostgreSQL. Además, se implementaron medidas de seguridad para las contraseñas utilizando `bcrypt`.

Ejemplo de hash de contraseñas con `bcrypt`:

```
const bcrypt = require('bcrypt');
const password = '
    miContrasenaSeguraCriptoKnights';

bcrypt.hash(password, 10, (err, hash) => {
  if (err) throw err;
  console.log(`Hash generado: ${hash}`);
});
```

Listing 7. Hash de contraseñas con bcrypt

### III-D. *Validación y Almacenamiento de Datos*

En el servidor, la firma digital generada por el cliente se valida para garantizar la autenticidad de los datos. Una vez validada, la información se descifra y se almacena en la base de datos.

Ejemplo de validación de firma en el servidor:

```
const isSignatureValid = serverKey.verify(
    encryptedData, clientSignature);
if (isSignatureValid) {
  console.log('Firma valida. Los datos son
    autenticos.');
} else {
  console.log('Firma invalida. Posible
    manipulacion.');
}
```

Listing 8. Validación de firma con ECDSA

## IV. Resultados y Análisis

### IV-A. *Pruebas Locales*

El sistema se implementó y probó en un entorno local controlado para evaluar su funcionalidad y rendimiento. Las pruebas realizadas incluyeron la generación de claves, el cifrado y descifrado de datos, la firma y validación de datos, así como el rechazo de datos manipulados. Los resultados obtenidos demostraron que el sistema cumple con los objetivos establecidos de garantizar confidencialidad, integridad y autenticidad.

*IV-A1. Tiempos de Ejecución:* Las operaciones realizadas durante las pruebas locales mostraron tiempos de ejecución óptimos:

- **Generación de Claves ECDH:** Promedio de 10ms.
- **Cálculo del Secreto Compartido:** Promedio de 5ms.
- **Cifrado AES:** Promedio de 8ms por mensaje.
- **Firma ECDSA:** Promedio de 12ms.
- **Validación de Firmas:** Promedio de 15ms.

Los tiempos obtenidos son adecuados para sistemas en tiempo real, como IoT, donde la latencia es un factor crítico.

*IV-A2. Validación de Datos No Alterados:* En todas las pruebas realizadas con datos no alterados, el servidor validó exitosamente las firmas generadas por el cliente y descifró los datos correctamente. Esto confirma que la implementación de ECDSA es funcional y efectiva.

*IV-A3. Rechazo de Datos Alterados:* En simulaciones de ataques de manipulación (tampering), donde los datos cifrados fueron alterados antes de llegar al servidor, el sistema rechazó los datos debido a firmas inválidas. Este resultado demuestra la robustez del sistema frente a ataques de integridad.

### *IV-B. Análisis de Escalabilidad*

Aunque el sistema fue probado localmente, su diseño modular permite su implementación en escenarios reales con múltiples sensores distribuidos. Algunas mejoras necesarias para la escalabilidad incluyen:

- **Integración con la Nube:** Utilizar servicios en la nube como AWS o Azure para manejar grandes volúmenes de datos en tiempo real.
- **Redundancia de Claves:** Implementar mecanismos de rotación de claves para aumentar la seguridad en sistemas a largo plazo.
- **Canal Seguro:** Integrar HTTPS/TLS para proteger la transmisión de datos entre cliente y servidor.

### *IV-C. Limitaciones Identificadas*

A pesar de los resultados positivos, se identificaron ciertas limitaciones durante las pruebas:

- **Pruebas en Entornos Reales:** El sistema no fue probado con sensores físicos o en entornos distribuidos con múltiples clientes conectados simultáneamente.
- **Gestión de Claves:** Aunque la implementación actual utiliza ECDH para derivar secretos compartidos, no se exploraron mecanismos avanzados para la gestión y rotación de claves.
- **Canal de Comunicación:** En la versión actual, la comunicación cliente-servidor no utiliza un canal seguro como HTTPS, lo que podría ser un riesgo en entornos de red no confiables.

## V. Escenarios Potenciales de Uso

La arquitectura desarrollada tiene aplicaciones prácticas en varios escenarios. Algunos ejemplos incluyen:

- **Monitoreo Ambiental:** Sensores distribuidos recopilan datos de temperatura, humedad y calidad del aire, enviándolos cifrados a un servidor central.
- **Redes Industriales:** Comunicación segura entre sensores y actuadores en fábricas inteligentes, garantizando la integridad de los datos de control.
- **Sistemas de Salud:** Transmisión segura de datos de pacientes desde dispositivos médicos a sistemas hospitalarios.

## VI. Discusión

La implementación del sistema presentó diversos desafíos y oportunidades que permiten analizar tanto el estado actual como las posibles extensiones futuras del proyecto. A continuación, se discuten las principales decisiones tecnológicas, los beneficios obtenidos y las limitaciones identificadas.

### *VI-A. Análisis de las Tecnologías Utilizadas*

El proyecto se construyó sobre una base tecnológica moderna que incluye ECDSA, ECDH y AES para la seguridad criptográfica, junto con herramientas como React, Node.js y MySQL para la implementación de cliente, servidor y almacenamiento de datos. Cada una de estas tecnologías fue seleccionada por razones específicas:

*VI-A1. ECDSA:* fue elegido para la autenticidad e integridad de los datos debido a su alta eficiencia y menor tamaño de claves en comparación con RSA. Por ejemplo, una clave de 256 bits de ECDSA ofrece un nivel de seguridad equivalente al de una clave RSA de 3072 bits, lo que reduce significativamente el uso de recursos. Sin embargo, EdDSA, basado en curvas modernas como `ed25519`, podría ser una opción futura, ya que es más rápido y menos propenso a errores de implementación, aunque presenta menor soporte en sistemas heredados.

*VI-A2. ECDH:* Para el intercambio seguro de claves, se utilizó **ECDH**, que garantiza un mecanismo eficiente sin necesidad de un canal seguro inicial. La curva `secp256k1` fue elegida por su amplia adopción y rendimiento comprobado. Alternativamente, `X25519` podría ofrecer una mejor resistencia a errores de implementación y mayor velocidad, pero requiere evaluar su soporte en entornos específicos.

*VI-A3. AES:* **AES** se empleó para garantizar la confidencialidad de los datos, utilizando claves de 256 bits. Aunque es un estándar ampliamente aceptado, `ChaCha20` podría ser una alternativa en aplicaciones móviles o sistemas sin aceleración por hardware, ya que ofrece un rendimiento más uniforme en dichos entornos. Sin embargo, el soporte de AES en hardware lo mantiene como la elección más eficiente en términos generales.

*VI-A4. Herramientas del Backend:* El servidor fue desarrollado en Node.js utilizando librerías como:

- `express`: Simplifica la creación de endpoints RESTful [6].
- `crypto-js`: Proporciona una solución estándar para el cifrado y descifrado de datos.
- `bcrypt`: Garantiza la seguridad de las contraseñas almacenadas mediante el uso de hashes y sal para evitar ataques de fuerza bruta [7].
- `JWT`: Permite la autenticación segura mediante tokens firmados digitalmente.

Estas herramientas no solo garantizaron la seguridad, sino que también facilitaron el desarrollo modular del sistema.

*VI-A5. Herramientas del Frontend:* El cliente se desarrolló en React con el apoyo de Vite para optimizar el proceso de desarrollo. Además, se emplearon librerías como:

- `Tailwind CSS`: Simplificó el diseño de la interfaz, haciendo que esta sea visualmente atractiva y funcional.
- `elliptic`: Implementó las operaciones de ECDSA y ECDH directamente en el cliente.

### *VI-B. Beneficios del Sistema*

El sistema desarrollado ofrece múltiples ventajas:

- **Robustez Criptográfica:** La combinación de ECDSA, ECDH y AES garantiza que los datos sean confidenciales, íntegros y auténticos.
- **Escalabilidad:** El diseño modular permite extender el sistema para manejar múltiples sensores en entornos distribuidos.
- **Eficiencia:** Los tiempos de ejecución observados son ideales para aplicaciones en tiempo real como IoT.
- **Facilidad de Integración:** Tecnologías como Node.js y React son ampliamente utilizadas en la industria, facilitando la adopción y mantenimiento del sistema.

### *VI-C. Limitaciones del Sistema*

A pesar de los resultados positivos, se identificaron ciertas áreas de mejora:

- **Falta de Canal Seguro:** La implementación actual no utiliza HTTPS/TLS, lo que podría exponer los datos a ataques de red.
- **Pruebas Limitadas:** Las pruebas se realizaron en un entorno local, por lo que no se evaluaron las capacidades del sistema bajo condiciones reales con múltiples sensores distribuidos.
- **Gestión de Claves:** No se implementaron estrategias avanzadas de rotación o almacenamiento seguro de claves privadas.

### *VI-D. Mejoras Futuras*

Con base en las limitaciones identificadas, se proponen las siguientes mejoras:

- **Integración con HTTPS/TLS:** Proteger la comunicación cliente-servidor mediante un canal seguro.
- **Pruebas en Entornos Reales:** Ampliar las pruebas a escenarios con sensores físicos conectados a una nube centralizada.
- **Rotación de Claves:** Implementar mecanismos para renovar claves periódicamente y mitigar posibles riesgos de seguridad.
- **Optimización de Recursos:** Explorar librerías más eficientes para el cifrado y descifrado en dispositivos con recursos limitados.

### *VI-E. Reflexión General*

Este proyecto demostró que es posible construir un sistema de comunicación seguro y eficiente utilizando herramientas y algoritmos modernos. Si bien se lograron resultados prometedores en el entorno de prueba, es evidente que su implementación en un entorno real requeriría ajustes adicionales para garantizar su robustez frente a los desafíos del mundo real. A pesar de las limitaciones, el diseño modular del sistema asegura su escalabilidad, haciendo posible su aplicación en sectores críticos como IoT, telecomunicaciones y monitoreo industrial.

## VII. Conclusiones

### *VII-A. Principales Resultados*

El proyecto demostró que es posible implementar un sistema seguro y eficiente utilizando tecnologías modernas de criptografía. Las pruebas realizadas validaron que el sistema protege la confidencialidad, integridad y autenticidad de los datos transmitidos. Además, la implementación modular asegura que el sistema puede adaptarse a diferentes entornos y escalas.

### *VII-B. Mejoras Futuras*

Para llevar este proyecto a un entorno de producción, se proponen las siguientes mejoras:

- Integrar un canal de comunicación seguro utilizando HTTPS/TLS.
- Implementar un sistema de monitoreo en tiempo real para supervisar el estado de los sensores y la integridad de los datos.
- Expandir las pruebas a entornos reales con sensores físicos y múltiples nodos conectados.

**Agregar aquí una tabla resumen con los resultados obtenidos y las propuestas de mejora.**

## VIII. Feedback

A continuación se verá un resumen con el feedback recibido de 2 grupos, grupo 7 (D.FERNANDEZ, M.FLORES, C.MUJICA y C.TRONCOSO.) y Grupo 8 (T.CAMACHO y N.GOROSTIDI.).
Cosas que encontraron positivas: Utilizar ECDH, ambos grupos utilizaron ECDH y encontraron que es el mejor punto entre seguridad y bajo computo. Utilizar la curva elíptica SECP256K1, tiene menor tamaño y computo que otras curvas, encontraron que es correcto para el problema. Haber armado todo el sistema, la base de datos, la API, además de los sensores. Hay ventajas de firmar el mensaje encriptado, encontramos que está correcto que lo hayan hecho.
Cosas que encontraron que podríamos mejorar:

Utilizar AES puede ser complicado para hardware limitado, ¿Se consideró eso? ¿Cuál es la justificación?
*R: Se consideró el amplio soporte y optimización en hardware, ya que bastante dispositivos tienen optimización criptográfica para AES.*
¿Qué algoritmo utilizaron para firmar? ¿Consideraron el tamaño de la firma?
*R: Se utilizó ECDSA por su alta eficiencia y seguridad, permite generar firmas más pequeñas comparadas con RSA, manteniendo el sistema ligero y seguro.*
¿Cuál es el tamaño del mensaje enviado por el sensor?
*R: Es importante ver si hay sobrecarga de la bandwith. Consideramos:*
*IV: 32 bytes, Signature: 64 bytes, Public Key: 130 bytes, Mensaje: n bytes*

¿Cuál es el esquema de descripción?
*Teniendo ya la llave pública, la firma y el vector de inicialización, se verifica la firma con la llave pública del cliente, luego se desencripta el mensaje con la llave simétrica ECDH (ClientPublicKey y ServerPrivateKey) y el vector de inicialización.*

## IX. Agradecimientos

Agradecemos a la profesora Berioska Contreras y al ayudante Vicente Tejos por su guía durante el desarrollo de este proyecto. Junto con nuestros compañeros que tuvieron la disposición de darnos feedback y ofrecernos mejoras a futuro.

## Índice



## Referencias

[1] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.
[2] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
[3] I. Meta Platforms, *React Documentation*, 2024, Última consulta: noviembre 2024. [Online]. Available: https://react.dev
[4] E. You, *Vite Documentation*, 2024, Última consulta: noviembre 2024. [Online]. Available: https://vitejs.dev
[5] O. Foundation, *Node.js Documentation*, 2024, Última consulta: noviembre 2024. [Online]. Available: https://nodejs.org
[6] ——, *Express Documentation*, 2024, Última consulta: noviembre 2024. [Online]. Available: https://expressjs.com
[7] C. Hale, *bcrypt: A Modern Password Hashing Library*, 2024, Última consulta: noviembre 2024. [Online]. Available: https://github.com/pyca/bcrypt