

Project: Delivery #3.

Digital Signatures

David José León, Nicolás Parra Ramos,
Johnathan Andrés Leon, Dago Alexander Fonseca
Team C

10 Febrero, 2019

1 Descripción del diseño de la aplicación:

El patrón de arquitectura en el que está basada CypherApp es el de MVC, el front y el back se encuentran divididos. Para el Frontend se usó la tecnología de React, mientras que para el Backend se usó el framework de Rails.

Para cumplir con los requerimientos exigidos la idea que se llevó a cabo fue la de una aplicación cuya funcionalidad es la de servir como un medio seguro para la comunicación de mensajes privados.

Dentro nuestra aplicación un usuario tiene la posibilidad de crear un canal de comunicación a través del cual puede 'enviar' mensajes a otra persona. Los mensajes son cifrados y luego almacenados en la base de datos para que puedan luego ser consultados por el destinatario usando las llaves requeridas, por lo que solo la persona que conoce las claves puede acceder a la información del canal.

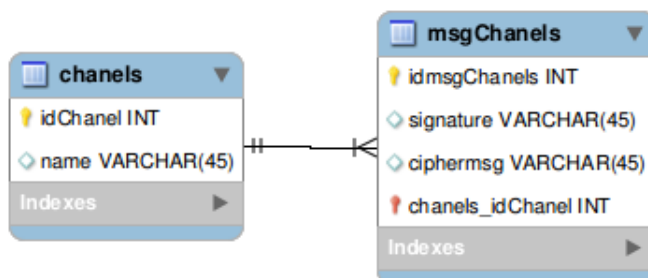
Otra de las razones por la que alguien usaría nuestro servicio, sería para enviar mensajes a otra persona, pero sin que esta pueda conocer su identidad o sus datos de contacto (lo que no pasaría si se comunicara por correo electrónico).

El acceso a todos los canales es público, por lo que cualquier persona puede entrar a cualquier canal y enviar un mensaje si conoce las claves.

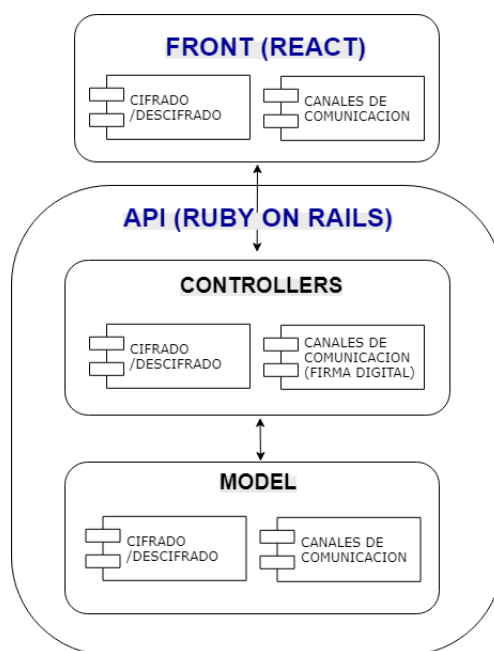
Debido a que el único medio de autenticación con el que se cuenta es la clave de cifrado y descifrado, se hace necesario otro medio de autenticación mejorar la seguridad y evitar el problema de suplantación de identidad que podría presentarse si una persona logra apoderarse de la clave y se hace pasar por la persona que envía los mensajes que espera el destinatario. Por lo que se identificó este como el escenario apropiado y se decidió usar firmas digitales para solucionar el problema.

La aplicación está compuesta por dos módulos: el modulo de cifrado/descifrado y el modulo de los canales de comunicación. Dentro del modo de cifrado/descifrado se encuentra la funcionalidad original sin el uso de firmas digitales, la cual es una versión mas simplificada de la versión completa que se encuentra dentro del modulo de canales de comunicación, la cual ofrece la funcionalidad de buscar canales y la que permite la autenticación del origen de los mensajes recibidos.

1.1 Modelo relacional



1.2 Arquitectura



2 Descripción del algoritmo implementado

La presente aplicación presenta una completa descripción del algoritmo creado e implementado por el equipo en la aplicación. Este algoritmo se ve fuertemente influenciado por el cifrado Hill.

2.1 Nombre del algoritmo: Criptex

Desde una vista general, el algoritmo Criptex se puede describir como un algoritmo de sustitución polialfabético, por bloques y simétrico. El modo de operación que utiliza el algoritmo es el Electronic Code

Block. El tamaño de los bloques de este algortimo es de 5 caracteres. A continuación se explica de manera detallada los pasos del algortimo.

2.2 Pasos para el proceso de cifrado:

- Se recibe en lenguaje natural el mensaje a cifrar y la clave. La clave puede tener como mínimo un tamaño de 5 caracteres y como máximo 20 caracteres. El tamaño del mensaje debe ser como mínimo 5 caracteres y máximo 500 caracteres.
- Se calcula la llave a través de un entero que es generado por una función hash. El entero indica que llave usar de un arreglo de llaves predefinido. La función hash consiste en multiplicar los caracteres de la clave y al final realizar el módulo 17 al resultado de la multiplicación.
Nota = se realiza módulo 17 ya que es el número de llaves predefinidas. Cada llave es una matriz de tamaño 5x5.
- En caso de que el mensaje no tenga el tamaño adecuado para dividirlo en bloques de 5 caracteres, se le agrega al mismo el caracter '#' las veces que sea necesario.
- Se realiza una permutación que consiste en dividir en dos partes el mensaje e intercambiar la parte derecha con la izquierda.
- Se divide el mensaje permutado en bloques de 5 caracteres.
- Cada caracter del bloque se convierte a su código ASCII.
- Se multiplica cada bloque con la llave que se obtuvo en los pasos anteriores. Como resultado de cada iteración se obtiene un arreglo de tamaño 5 con elementos que van de 0 a 255.
- Se concatenan todos los bloques obtenidos.
- Se convierte cada elemento del mensaje cifrado en su respectiva representación en hexadecimal.
- Finalmente se agrega al final del mensaje cifrado el número de veces que fue necesario utilizar el comodín (#).

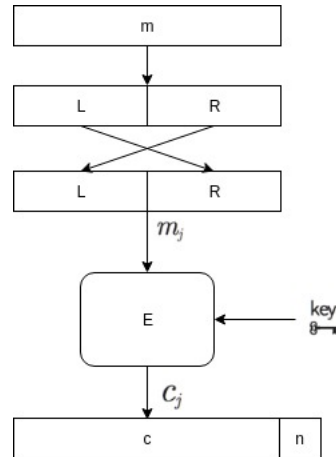
Nota:

El proceso explicado arriba solo se realiza una vez.

Las matrices predefinidas deben tener inversa en módulo 256.

Las matrices inversa se obtienen con el mismo procedimiento que en el algortimo Hill.

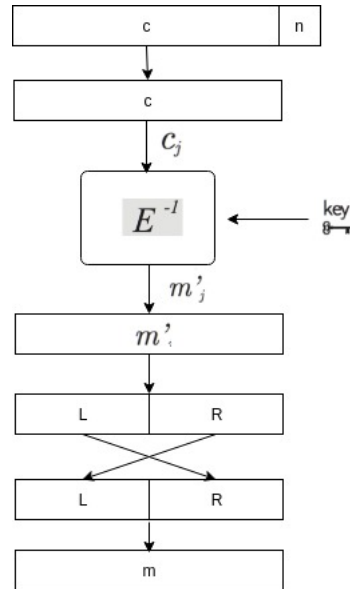
2.3 Diagrama proceso de cifrado.



2.4 Pasos para el proceso de descifrado:

- Se recibe la clave y el mensaje cifrado. La clave puede tener como mínimo un tamaño de 5 caracteres y como máximo 20 caracteres.
- Se calcula la llave inversa a través de un entero que es generado por la función hash mencionada anteriormente. El entero indica que llave usar de un arreglo de llaves inversas predefinido.
- Se remueve el ultimo caracter del mensaje cifrado y se guarda para su posterior utilización.
- Cada caracter del texto cifrado se convierte a su codigo ASCII.
- Se divide el texto cifrado en bloques de 5 caracteres.
- Se multiplica cada bloque con la llave inversa que se obtuvo en pasos anteriores. Como resultado de cada iteracion se obtiene un arreglo de tamaño 5 con elementos que van de 0 a 255.
- Se concatenan todos los bloques obtenidos.
- Se convierte cada elemento del resultado anterior en sus respectivos caracteres de acuerdo con su código ASCII.
- Se realiza una permutación que consiste en dividir en dos partes el mensaje e intercambiar la parte derecha con la izquierda.
- Finalmente se eliminan los comodines agrgados en el proceso de cifrado. Se sabe la cantidad que se agregaron gracias al ultimo caracter que se eliminó del texto cifrado y que fue guardado.

2.5 Diagrama proceso de descifrado.



3 Descripción de la implementación de la firma digital en la aplicación web

3.1 Algoritmo (RSA):

El algoritmo de la firma digital que se basa en el criptosistema RSA se desarrolló en lenguaje ruby y consta de tres procedimientos principales: Generación de llaves, Generación de la firma y Verificación de la firma:

- Generación de llaves: Los números primos p y q , a partir de los cuales se generan las llaves se obtienen de una un arreglo constante de números primos almacenados en memoria, es decir, los números que usamos no se generan, sino que se seleccionan aleatoriamente de ese conjunto previamente establecido. Dado que en el arreglo los números se encuentran ordenados de menor a mayor, lo que se hace es que luego asignar p a uno de los elementos del arreglo correspondiente a un número aleatorio dentro del rango de las posiciones del mismo, se elige q de tal forma que no puede haber mas de 5 posiciones entre la posición de p y la posición de q para garantizar que los dos números tengan un valor absoluto cercano.
Para generar e se genera un numero aleatorio desde 2 hasta $\phi-1$ hasta encontrar el primero para el que el MCD entre ϕ y e tentativo sea igual a 1.
Para hallar d se usa el algoritmo extendido de Euler con la salvedad necesaria para el caso en el que d es menor que cero, la cual consiste en sumar ϕ a d .
Para los procedimientos auxiliares como PowerMod y EEA se siguieron los pseudocódigos de las presentaciones del curso.
- Tanto el método generador de la firma como el verificador reciben el mensaje original como un string, el cual se convierte a un arreglo de códigos ASCII correspondientes a cada carácter para proceder a

calcular la función PowerMod con las respectivas llaves en cada caso.

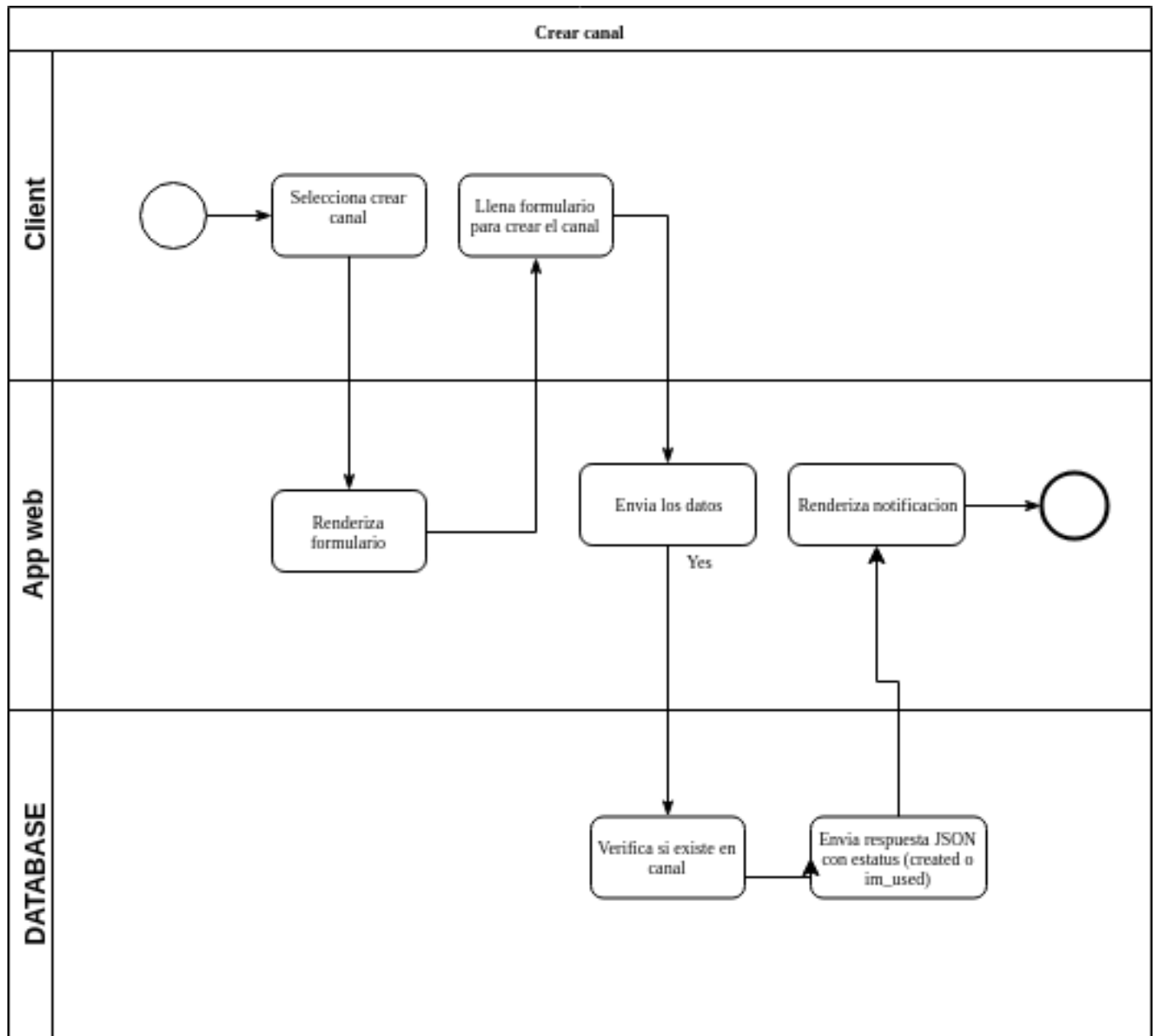
3.2 Modo de operación de la firma dentro de la aplicación:

El funcionamiento de la implementación se puede explicar a través de tres casos de uso:

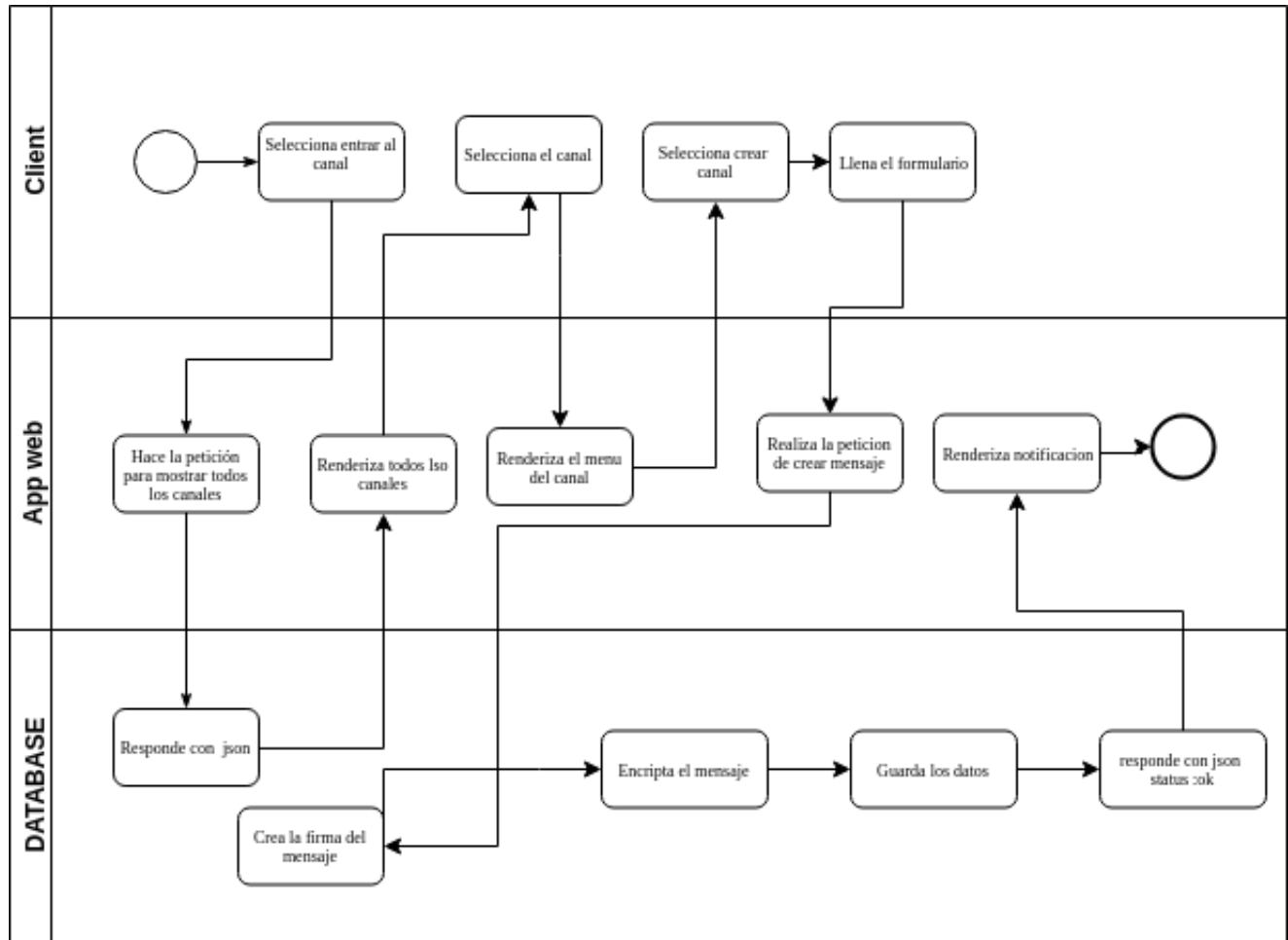
- Creación de un canal: Para crear el canal el usuario debe ingresar el nombre del canal, su correo electrónico y el del destinatario. Una vez los datos son validados, en el controlador cuando se crea el nuevo canal en la base de datos, se generan las llaves y se envía un correo con la llave privada para el emisor de los mensajes y otro con la llave publica para el receptor. Cada canal tiene una sola llave privada y una sola llave pública.
- - Creación de un mensaje: La operación de la creación de un mensaje requiere de los siguientes datos de entrada que el usuario que envía el mensaje debe proporcionar:// La llave de cifrado, que es la que se usa junto con el algoritmo de cifrado elegido por el usuario para cifrar el mensaje y almacenarlo en la base de datos; la clave secreta del RSA y el mensaje, con los que se genera la firma digital que es también guardada en la base datos para uso posterior. Cada mensaje dentro de un canal tiene su propia firma digital asociada.
- Consulta del mensaje: Cuando el destinatario quiera obtener el mensaje deberá primero seleccionar o buscar el canal de comunicación establecido por el emisor de los mensajes, luego deberá suministrar la clave de cifrado en adición a la clave publica de la firma y el id del mensaje que desea consultar. // El sistema descifra el mensaje cifrado que se encuentra en la base de datos con la llave suministrada y lo compara con la firma del mensaje descifrada usando la clave publica y RSA. Si concuerdan se envía una notificación al usuario diciendo que se comprobó que el mensaje recibido fue enviado por la persona que creo el canal y por lo tanto, de quien espera recibir el mensaje; de lo contrario se informara que no se puede confirmar el origen del mensaje.

3.3 Flujos

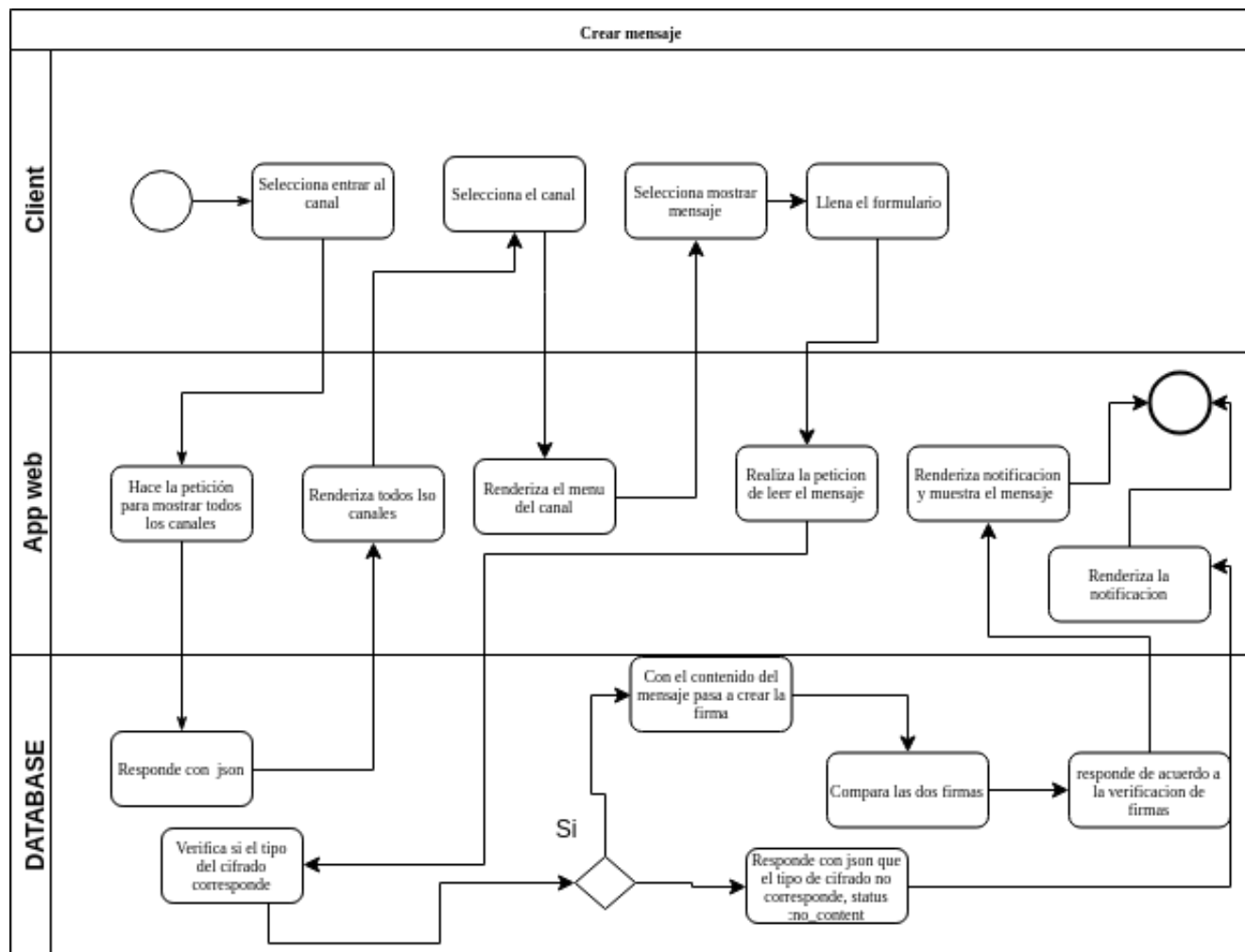
3.3.1 Crear canal



3.3.2 Crear mensaje



3.3.3 Mostrar mensaje



4 Plan de acción

A continuación se describen los arreglos realizados para contrarrestar las vulnerabilidades que el equipo *D* encontró en la aplicación

4.1 Variables de entorno

Se configura el archivo *.gitignore* para las variables de entorno que se encuentran ubicadas en */config/local_env.yml*.

```
73 config.action_mailer.raise_delivery_errors = true
74 config.action_mailer.delivery_method = :smtp
75 config.action_mailer.smtp_settings = {
76   address:           'smtp.gmail.com',
77   port:              587,
78   domain:            ENV["MAIL_DOMAIN"],
79   user_name:         ENV["MAIL_USERNAME"],
80   password:          ENV["MAIL_PASSWORD"],
81   authentication:    :plain,
82   enable_starttls_auto: true
83 }
```

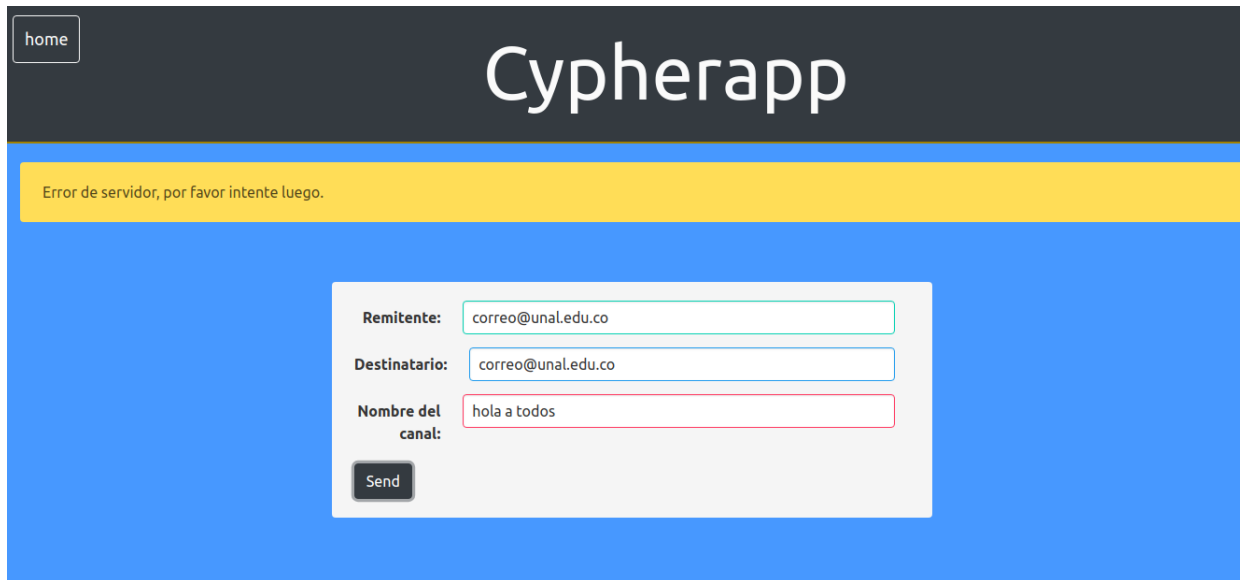
Variables de entorno en development.rb

Por otro lado se configuraron las variables de entorno en heroku.

4.2 Notificaciones

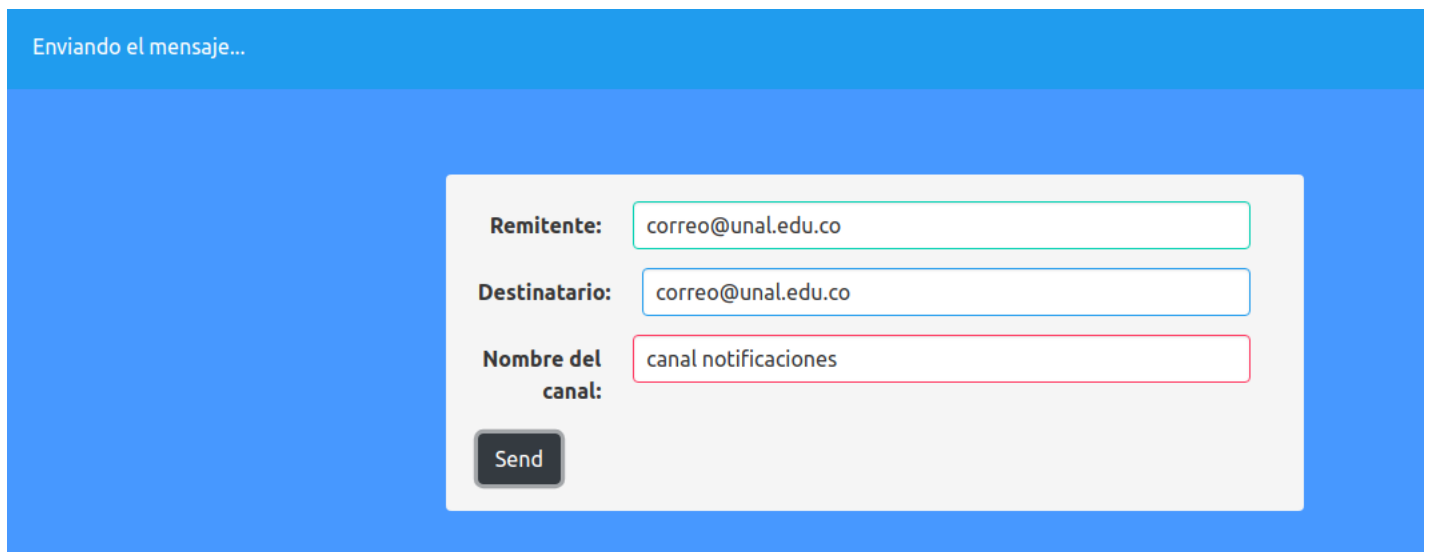
4.2.1 Creación de canal

Se agregaron dos notificaciones, cuando se está enviando las mensajes y cuando se recibe un error del servidor, esté error posiblemente ocurre cuando Heroku llega al limite de mensajes enviados.



The screenshot shows the Cypherapp interface. At the top left is a 'home' button. The main header displays 'Cypherapp'. Below the header, a yellow banner contains the message 'Error de servidor, por favor intente luego.' In the center, there is a form for creating a message. The form includes three input fields: 'Remitente:' with the value 'correo@unal.edu.co', 'Destinatario:' with the value 'correo@unal.edu.co', and 'Nombre del canal:' with the value 'hola a todos'. A 'Send' button is located at the bottom of the form.

Error de servidor



The screenshot shows the Cypherapp interface with a blue header bar that says 'Enviando el mensaje...'. Below the header, there is a form for creating a message. The form includes three input fields: 'Remitente:' with the value 'correo@unal.edu.co', 'Destinatario:' with the value 'correo@unal.edu.co', and 'Nombre del canal:' with the value 'canal notificaciones'. A 'Send' button is located at the bottom of the form.

Error de servidor

4.2.2 Cifrado y descifrado

Se han agregado notificaciones cuando el mensaje no corresponde con el tipo de cifrado, cuando no existe un ID y cuando el nombre del mensaje no se encuentra.

[home](#) Cypherapp

No coincide el tipo de cifrado

Add a secret message!

Name message:

Select type cipher:

DES

Message:

Key:

Send

Get info message!

Name message:

id message:

Key:

Select type cipher:

Criptex

Send

Error de cifrado

No se encontro el mensaje

Add a secret message!

Name message:

Select type cipher:

DES

Message:

Key:

Send

Get info message!

Name message:

id message:

Key:

Select type cipher:

DES

Send

Mensaje no encontrado

4.2.3 Firmas

Anteriormente se podía ver el mensaje si las firmas no coincidían, ahora se le notifica que no está autorizado y no se muestra el mensaje.

No esta autorizado para leer el mensaje.

NOMBRE DEL CANAL:
THIS IS A TEST

Crear mensaje

Mostrar mensaje

key

12345

type chipher

DES

n

12345

public key

12345

id

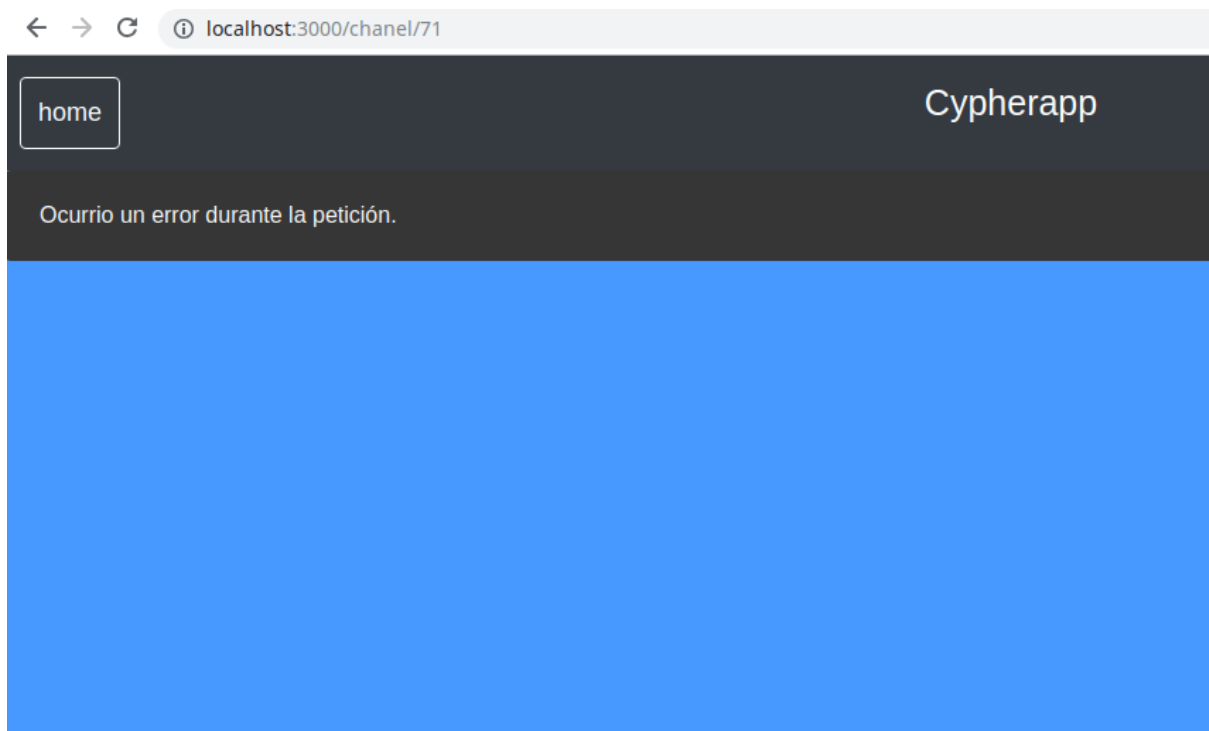
1

Send

No autorizado

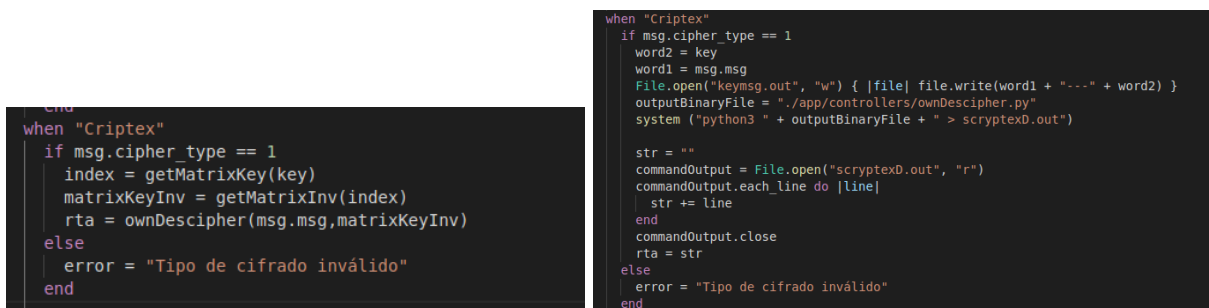
4.3 Errores

Se corrigió el error cuando se intentaba entrar a un canal por medio de un link, el error es producido por el componente que espera un prop del nombre del canal, por lo que hace obligatorio ingresar directamente con la GUI.



URL error

Para evitar las vulnerabilidades encontradas al momento de ejecutar el cifrado y descifrado con el algoritmo Criptex, se decidió migrar del lenguaje Python a Ruby. Esto con el propósito de librarse del llamado que se realizaba al script donde estaban las instrucciones del algoritmo Criptex y no abrir una puerta de acceso al sistema.



Contraste de implementaciones

4.4 Validaciones

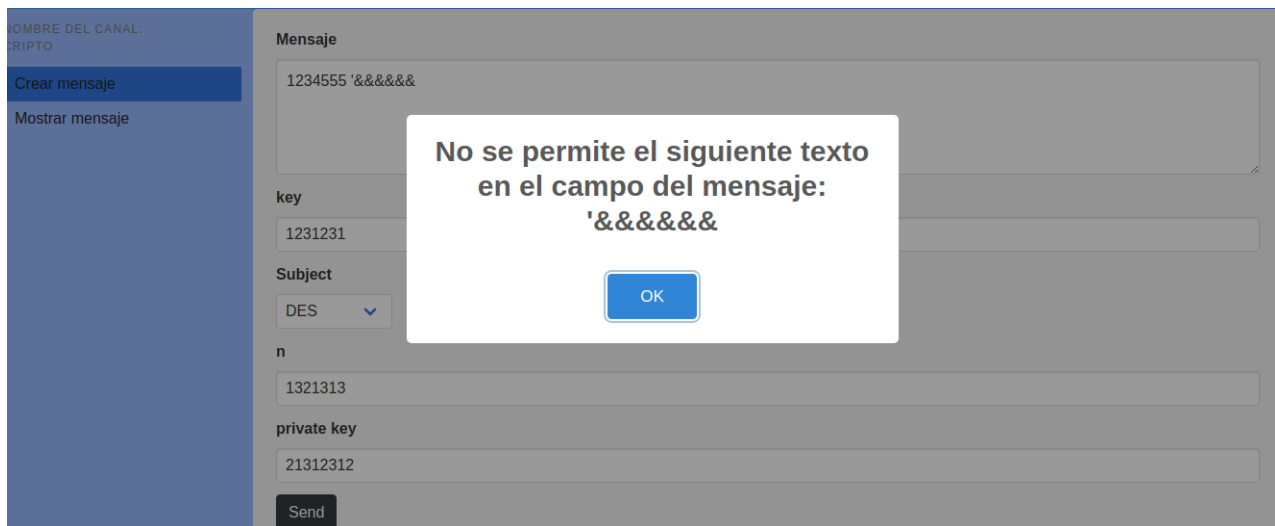
Se agregaron validaciones que evitan caracteres como el apóstrofe (') , este caracter hace que el archivo .c tenga problemas a la hora de ejecutarlo además permitia ejecutar codigo externos como bucles. Para solucionar esto se hace uso de validaciones del campo del mensaje, el cual sera posteriormente encriptado.

```
1 function validate(text, exp){
2   var expresion = RegExp(exp);
3   return [!expresion.test(text) , text.match(exp) ]
4 }
```

Listing 1: Función para validar

```
1 const isValid = validate(info.msg , "['\&]+")
2   console.log(isValid)
3   if(isValid[0]){
4     this.props.createMessage(info);
5   }else{
6     swal("No se permite el siguiente texto en el campo del mensaje: " + isValid[1][0] );
7   }
```

Listing 2: Uso de función en componentes



Validacion frontend

Con la misma expresion regular, se valida en los controladores.

```
1 if(!Cripto.isvalid(crypto_params[:msg]))
2   render json: {rta: nil, status: :non_authoritative_information}
3   return
4 end
```

Listing 3: Validación en controladores