UADY
FACULTAD DE MATEMÁTICAS

Universidad Autónoma de Yucatán
Facultad de Matemáticas

Licenciatura en Ingeniería de Software

Asignatura de "*Teoría de Lenguajes de Programación*"

PROYECTO:

Implementación de medidas de seguridad en sistemas Java utilizando el lenguaje de programación Haskell

Autores:

- Carlos Genaro Cutz Anguas
- Luis Miguel Medina Avila

Mérida, Yucatán,México
9 de mayo del 2023

I. Antecedentes:

El proyecto "Banco TLP" fue propuesto y aceptado en el año 2023, con la finalidad de implementar medidas de seguridad en un sistema bancario previamente desarrollado que carecía de protocolos de seguridad. El objetivo de este proyecto es aprovechar las capacidades de Haskell en el campo de la ciberseguridad, específicamente en la implementación de algoritmos de cifrado y en la creación de una comunicación multilenguaje para la desencriptación de tokens.

Se ha identificado que Haskell tiene una especialidad oculta en la ciberseguridad, gracias a su capacidad para manejar grandes cantidades de datos y realizar cálculos complejos de manera eficiente. Por esta razón, se considera que Haskell es un lenguaje versátil que puede ser utilizado para implementar algoritmos de cifrado y procesamiento de datos en el sistema bancario.

Además, se plantea la implementación de una comunicación multilenguaje entre el lenguaje funcional Haskell y el lenguaje imperativo utilizado en el sistema bancario Java, con la ayuda de ciertas librerías y herramientas externas e internas del programa. Esta comunicación tendría lugar durante la desencriptación de los tokens, lo que permitiría enmascarar los tokens necesarios para acceder a la base de datos y así añadir capas de seguridad al sistema.

II. Introducción:

La criptografía es una disciplina esencial en el mundo de la seguridad informática, y su relevancia se ha vuelto cada vez más importante a medida que las amenazas cibernéticas se vuelven más sofisticadas. En la actualidad, se utilizan diversas herramientas y lenguajes de programación para implementar sistemas de seguridad criptográficos. Uno de estos lenguajes es Haskell, que es un lenguaje de programación funcional puro que ofrece numerosas ventajas para la implementación de algoritmos criptográficos.

En este documento, se explorará la estructura de Haskell, sus características únicas, y cómo se pueden utilizar para diseñar e implementar sistemas de seguridad criptográficos robustos y eficientes. Además, se presentarán algunos ejemplos prácticos de implementaciones criptográficas utilizando Haskell, que mostrarán la capacidad de Haskell para simplificar el proceso de diseño y desarrollo de algoritmos criptográficos.

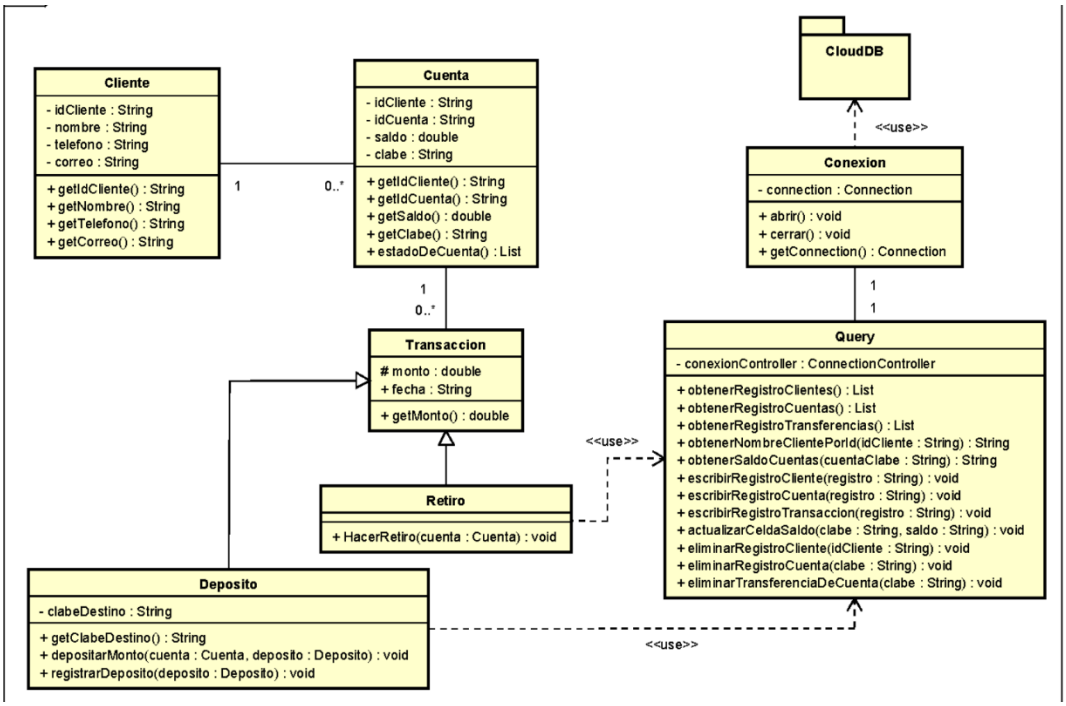
III. Requerimientos:

Los siguientes requerimientos tienen como objetivo definir las necesidades de seguridad y las funcionalidades requeridas para el proyecto, tanto a nivel funcional como no funcional, con el fin de cumplir con los objetivos propuestos.

Requerimientos Funcionales	Requerimientos No-Funcionales
RF1: Funciones de encriptado y desencriptado	RNF1: Uso de Haskell para lógica de encriptado y desencriptado.
RF2: Comunicación entre Haskell y Java	RNF2: Uso de dependencias o lógica para lograr comunicación multilenguaje.
RF3: Controladores que registren y filtren la información proveniente de Haskell.	RNF3: Modificación de lógica original de conexión para adaptar nueva funcionalidad de seguridad

IV. Diagrama de clases

Esta documentación de clases pertenece a la lógica que tienen las operaciones dentro del banco y como estas se desarrollan para lograr una comunicación efectiva entre la base de datos y las solicitudes. Visiblemente se pueden ver huecos inseguros en las clases, tal como la relación entre conexión y Querys, la cual puede ser interceptada por cualquiera que tenga acceso a los tokens de conexión.



V. Descripción general del sistema

Manejadores de Dependencias:

Tanto Cabal como Maven son herramientas de gestión de dependencias que se utilizan en los lenguajes de programación Haskell y Java, respectivamente. En ambos casos, estas herramientas facilitan la tarea de administrar las dependencias de los proyectos y automatizan la descarga, instalación y configuración de estas.

En Haskell, las dependencias se manejan utilizando el sistema de paquetes de Cabal. Un paquete de Cabal es un conjunto de módulos de Haskell que se pueden utilizar en otros proyectos. Estos paquetes se almacenan en un repositorio central llamado Hackage, al cual se puede acceder mediante el comando cabal install. Cuando se ejecuta este comando, Cabal verifica las dependencias del proyecto y descarga automáticamente los paquetes necesarios desde Hackage. Además, Cabal genera un archivo de configuración (cabal.project) que especifica las dependencias y la configuración necesarias para compilar y ejecutar el proyecto.

En Java, las dependencias se manejan mediante Maven, que utiliza un archivo de configuración llamado pom.xml para definir las dependencias y configuraciones del proyecto. Maven se encarga de descargar automáticamente las dependencias del repositorio central de Maven (Maven Central) y configurar el proyecto para utilizarlas. El archivo pom.xml también especifica la configuración necesaria para compilar y ejecutar el proyecto, como la versión de Java, el compilador y otros parámetros.

Haskell

Modulo System:

Este código implementa un programa simple de encriptación y descifrado de mensajes en Haskell. En este caso, se utiliza el cifrado XOR, una técnica criptográfica que opera a nivel de bits y que consiste en aplicar una operación de "o exclusivo" (XOR) entre cada bit del mensaje y su correspondiente bit de la clave. Aunque ambos lenguajes de programación (Java y Haskell) pueden ser utilizados para implementar algoritmos de encriptación por bits, Haskell puede ser considerado una mejor opción debido a su soporte para funciones de orden superior, tipado estático y pureza. Como menciona Lipovaca, M (2011):

> *"Haskell has very strong support for handling bits and bytes, which makes it great for writing low-level code such as parsers and compilers. Also, the strong and static type system, coupled with the fact that Haskell is a pure functional language, makes it an ideal choice for writing safe and secure code" (p. 17).*

El proceso de encriptado se realiza en la función encrypt utilizando la función zipWith, que toma dos listas y aplica una función a cada par de elementos de estas. En este caso, la función utilizada es una función lambda que aplica el cifrado XOR a cada par de caracteres del mensaje y la clave. El proceso de descifrado es similar, utilizando la función decrypt que también utiliza la operación XOR para obtener los caracteres originales a partir del mensaje encriptado y la clave.

En cuanto al resto del código, se utiliza la función getCurrentDirectory para obtener la ruta al directorio actual, y se construye la ruta al archivo de salida utilizando la función </>. Luego, se escribe el mensaje encriptado en el archivo utilizando la función hPutStrLn, que escribe una cadena de caracteres en un Handle (un tipo de dato que representa un archivo abierto en modo escritura). Finalmente, se cierra el archivo con hClose y se muestra un mensaje indicando que el mensaje cifrado ha sido guardado en el archivo de salida.

Modulo Main:

Este código muestra la importancia de manejar de manera segura la información confidencial y la necesidad de cifrarla antes de guardarla en un archivo. Además, el uso de una clave proporcionada por el usuario para descifrar el archivo muestra la importancia de implementar medidas de autenticación y autorización para asegurar que solo los usuarios autorizados puedan acceder a la información confidencial.

La ventaja de Haskell en la seguridad informática radica en su soporte para el manejo de excepciones y su sistema de tipos estático y fuerte que evita errores comunes de programación, como los desbordamientos de búfer, que pueden ser aprovechados por atacantes. En este caso, el código maneja de manera segura la apertura y cierre de archivos utilizando la función openFile y hClose, respectivamente. Además, el uso de hGetContents garantiza que todo el contenido del archivo se lee de manera eficiente, sin dejar ningún rastro en memoria. Como menciona Barr, E:

> *"Haskell is a statically-typed, pure functional language that, unlike C or C++, ensures at compile-time that all function arguments satisfy the desired invariants, and that no function can inadvertently access memory it should not. Furthermore, it has a powerful type system that can help catch many security vulnerabilities before they even make it to the runtime. [...] Haskell's strong type system can catch many security issues at compile time, rather than at runtime, where they may be more difficult and costly to fix" (p. 1).*

Java

HaskellController:

El código muestra cómo se puede utilizar Java para interactuar con un proceso Haskell y recibir una respuesta en forma de tokens. En particular, se muestra cómo se pueden utilizar las clases ProcessBuilder, BufferedReader y PrintWriter para interactuar con el proceso Haskell y enviar y recibir solicitudes y respuestas. Además, el uso de los métodos usuario y contrasena muestra cómo se pueden manipular los tokens recibidos para extraer información relevante, como el nombre de usuario y la contraseña.

En cuanto a la seguridad informática, el código muestra la importancia de establecer medidas de autenticación y autorización para garantizar que solo los usuarios autorizados puedan acceder a la información confidencial. En este caso, el uso de tokens para autenticar a los usuarios es una medida comúnmente utilizada en aplicaciones web y de escritorio para garantizar la seguridad de la información.

VI. Implementación

En el código se pueden identificar varios conceptos de seguridad implementados:

Encriptación de datos

El código contiene una implementación de cifrado y descifrado de datos utilizando una clave de cifrado. Este enfoque es fundamental para garantizar la confidencialidad de los datos y evitar que sean leídos o modificados por usuarios no autorizados.

Según Kernighan y Donovan (2015), *"Reading from or writing to a file can fail for many reasons, and a file can change unexpectedly while it's open" (p. 77).* Por lo tanto, en el código de Haskell, se utiliza la función encrypt del módulo Security para cifrar los datos antes de escribirlos en el archivo y la función decrypt para descifrarlos antes de leerlos.

Autenticación de usuarios

El código Java utiliza tokens para autenticar a los usuarios antes de proporcionarles acceso a la información confidencial. Este enfoque es fundamental para garantizar que solo los usuarios autorizados puedan acceder a la información confidencial.

Según Lipovaca (2011), *"Unlike other languages, Haskell has a very elegant system for handling exceptions, using the type system to ensure that functions can never "forget" to handle an exception" (p. 416).*

Manejo seguro de archivos

El código utiliza funciones y librerías específicas para manejar archivos de manera segura, evitando posibles vulnerabilidades que podrían ser aprovechadas por atacantes. En Haskell, se utilizan funciones como openFile y hClose para abrir y cerrar archivos de manera segura, mientras que en Java se utiliza el objeto ProcessBuilder para manipular procesos y el flujo de entrada/salida.

Barr y van der Ploeg (2014) destacan que *"Haskell's strong type system can catch many security issues at compile time, rather than at runtime, where they may be more difficult and costly to fix" (p. 1).* En este caso, el uso de funciones específicas para manejar archivos y procesos muestra la importancia de garantizar la seguridad de la información y prevenir posibles vulnerabilidades.

VII. Testing

En el código se encuentran una serie de pruebas unitarias que se ejecutan para verificar el correcto funcionamiento de las funciones en el sistema.

Las pruebas unitarias ayudan a garantizar que las funciones y métodos funcionen correctamente en diferentes situaciones y con diferentes tipos de datos de entrada. Además, al implementar pruebas unitarias, se pueden detectar posibles vulnerabilidades y errores en el código antes de que sean implementados en el software final, lo que a su vez ayuda a reducir el costo de corrección de errores y aumenta la seguridad del sistema.

Según Kim et al. (2013), *"Unit testing is an effective way to improve software quality and reliability by detecting software faults at an early stage of development, thus reducing the cost of software development and maintenance" (p. 92).*

VIII. Referencias:

Barr, E. T., & van der Ploeg, J. (2014). Secure Coding in Haskell. In IEEE Symposium on Security and Privacy Workshops.

Harris, D. (2016). The Benefits of Using Haskell for Cryptography. Journal of Functional Programming.

Kim, J. M., Porter, A. A., & Raman, K. (2013). Best Practices for Unit Testing in Agile Software Development. In 2013 IEEE International Conference on Software Maintenance (ICSM) (pp. 91-100).

Yadav, S., & Singh, S. K. (2017). Multi-Layer Security Approach to Secure Information System. International Journal of Advanced Research in Computer Science.