Capítulo 3

Implementación del trabajo

A continuación se explica las decisiones para la implementación propuesta. Es importante notar que esta implementación basa sus decisiones primordialmente en el Anexo 20 de la Resolución de la Miscelánea Fiscal y en la Ley de Notariado del Estado de Puebla previamente descritas, además se tomaron decisiones en la implementación para mejorar el uso y entendimiento por el público de todos los procesos detrás del modelo de Notaria Digital propuesto persiguiendo así una implementación que sea transparente y segura debido al claro entendimiento de las personas sobre el mismo.

3.1. Firma y verificación

El proceso de firma y verificación es el procedimiento más esencial de la implementación. Este proceso se encuentra contenido dentro de nuestro paquete de *notaria.js*, dentro de este paquete se exponen como publicas dos funciones: verificar y firmar.

Como se observa en la figura 3.1 se realiza primero la creación de un hash utilizando SHA-256 del archivo original, a continuación este hash se transforma utilizando base 64. Posteriormente, se extraen las llaves públicas de los pares de llaves recibidos. Se realiza el proceso de firmado del hash del archivo original utilizando la primer llave privada recibida con firmado RSA con llave privada habilitando la utilización de padding en caso de que no se encuentren los bloques completos. A continuación, se realiza el firmado RSA con la siguiente llave privada recibida, en esta ocasión y en las posteriores ocasiones se deshabilita la utilización de padding ya que los bloques se en-

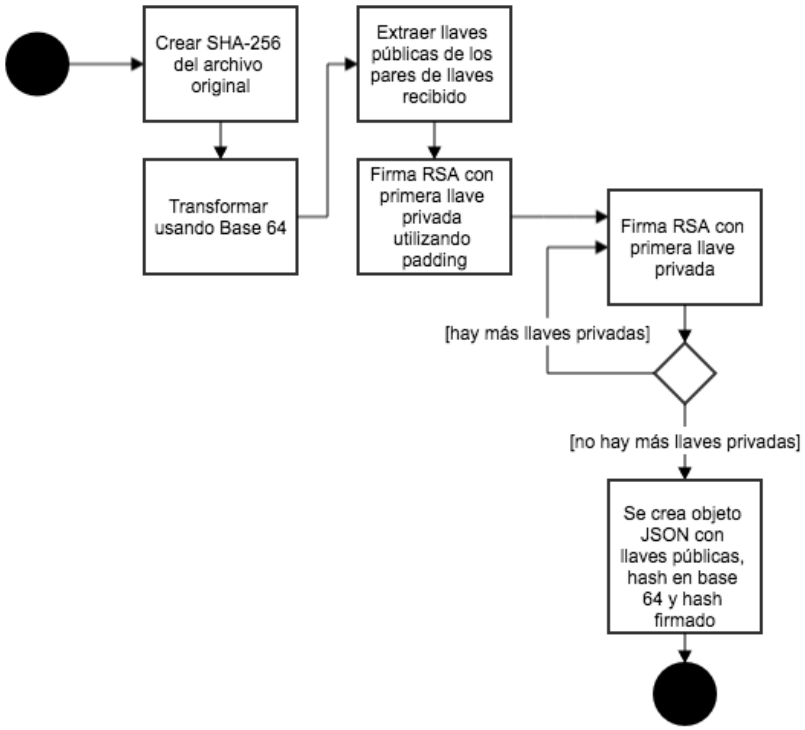

Figura 3.1: Firma

cuentran completos, este proceso se realiza con todas las llaves privadas. Al finalizar, se crea un objeto JSON con el hash del archivo original procesado con base 64, las llaves públicas extraídas y el hash firmado.

En la figura 3.2 se observa la verificación de los documentos utilizando el objeto JSON creado durante el proceso de firmado. Primero, utilizando el archivo original se crea un hash utilizando SHA-256 y se compara al hash recibido en el objeto JSON, en caso de no ser idénticos el proceso finaliza, de lo contrario continuamos con el siguiente paso. Utilizando la última llave púbica recibida se realiza el proceso de verificación del hash firmado con verificación RSA con llave pública deshabilitando la utilización de padding ya que los bloques se encuentran completos, este proceso se realiza con todas las llaves públicas excepto la última. Se realiza entonces la verificación RSA con llave privada habilitando la utilización de padding en caso de que no se encuentre los bloques completos. Se compara al hash recibido en el objeto JSON, en caso de no ser idénticos el proceso finaliza, de lo contrario se concluye que los datos recibidos en el objeto JSON si representan el archivo presentado.

Se tomó la decisión de utilizar este proceso de firmado y verificación debido que son los que mejor cumplen las necesidades que se buscan cubrir con esta implementación. Esta implementación permite ver quienes firmaron el documento utilizando sus llaves públicas y solamente utilizando todas las llaves públicas de las personas involucradas en el acto verificar la autenticidad del documento, es así que esta implementación permite confirmar que todas las personas involucradas aceptaran el acto en cuestión y no pueden rechazar firmar y ser parte del grupo de firmas presentes en el mismo. Así mismo, permite verificar que el documento digital presentado es válido ya que se puede verificar la validez del mismo junto con el archivo JSON presentado, es decir, si el archivo original y el archivo JSON presentados no corresponden en el momento de la verificación y si las firmas públicas no corresponden a la de las personas involucradas en el acto no se puede hacer la verificación del archivo original presentado y por lo tanto se presenta un implementación confiable y segura.

El proceso previamente descrito solamente corresponde al proceso de verificación y firmado ubicado dentro del paquete *notaria.js*, dicho proceso no equivale aun a todo el proceso llevado acabo como procesos de verificación dentro de la implementación presentada.

Por otra parte, se hace mención de que para la comunicación entre el cliente y servidor no utiliza ningún tipo de encriptación a nivel aplicación. La

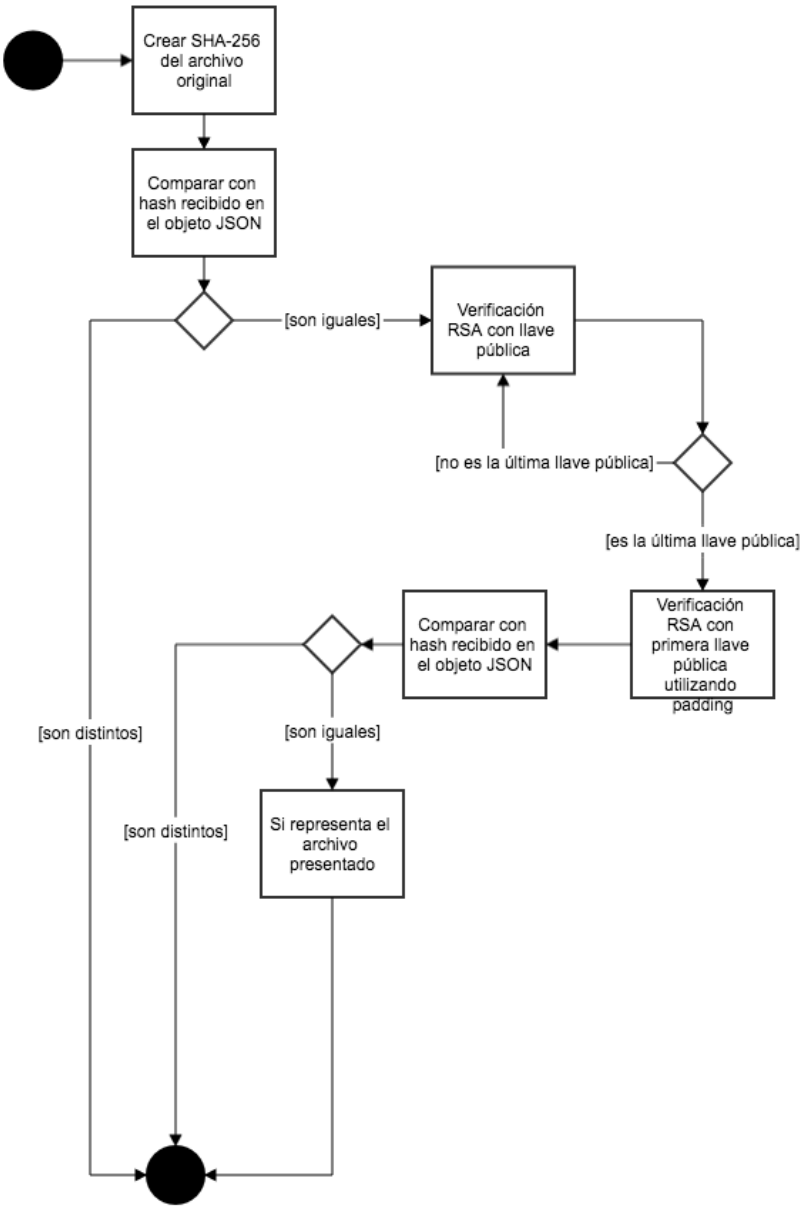

Figura 3.2: Verificación

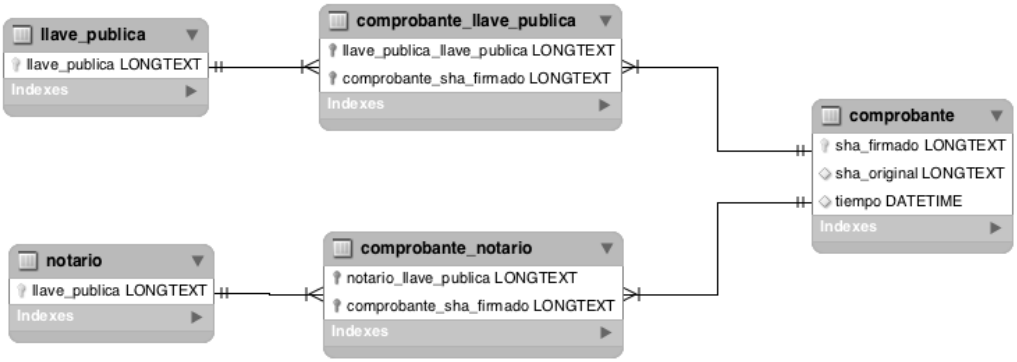

Figura 3.3: Base de datos

comunicación entre el cliente y servidor podría utilizar una implementación del algoritmo Diffie-Hellman para la transmisión de los datos. El algoritmo Diffie-Hellman funciona mediante la utilización de p que es un entero primo de gran tamaño y g que es un raíz primitiva modulo p, además se seleccionan dos enteros al azar de gran tamaño a y b que permanecerán secretos. Para calcular A se realiza $A = g^a \mod p$ y para B se utiliza $B = g^b \mod p$, finalmente para obtener s para A se utiliza $s = B^a \mod p$ y para B se utiliza $s = A^b \mod p$, siendo s el secreto compartido. Por ejemplo, al utilizar este algoritmo a y A podrán ser calculadas por el servidor, b y B por el cliente para asegurar la comunicación utilizando el secreto compartido s[49].

3.2. Base de datos

Otra parte central es la implementación de la base de datos, esta implementación solamente comprende los aspectos mínimos requeridos para la correcta implementación de una Notaria digital, se deberá ampliar los datos mínimos requeridos para comprender como mínimo los datos mínimos requeridos dentro del Anexo 20 de la Resolución Miscelánea Fiscal.

La base de datos cuenta con 5 tablas con 9 columnas totales, donde solamente 5 son únicas y las otras 4 son referencias a columnas dentro de otra tabla y la podemos observar en la figura 3.3.

Se cuenta primero con una tabla llamada *notario* donde estarán dadas de altas en la columna llave publica las llaves públicas de los notarios aceptados en la implementación, más adelante se hablará de esto pero en caso de que

no exista una llave pública de un notario dentro de uno de los pares enviados de llaves públicas no se realiza el proceso de alta de un nuevo documento.

A continuación tenemos la tabla *comprobante* donde se almacena el sha firmado como llave primaria, por lo que deberá ser único, además se almacena el sha original y la estampa de tiempo del momento de la creación de este registro que se toma, como se ve más adelante, en el momento en que se hace la petición al servidor.

La tabla *llavepublica* almacena, al igual que en la tabla notario, las llaves publicas de los distintos actores físicos o morales en los actos, como se verá más adelante esta tabla almacena todas las llaves publicas indistintamente y solamente en caso de ya estar dado de alta no lo guarda nuevamente.

Finalmente, la tabla *comprobantenotario* liga a un notario con un comprobante al igual que lo hace la tabla *comprobantellavepublica* que liga una llave pública con un comprobante.

Por otra parte, se aclara que aunque se cuentan con tablas internas para almacenar las llaves públicas tanto de los notarios como de las personas físicas y morales, es importante que se cuente con un servicio proveído por el SAT para poder verificar estas llaves públicas y no solamente como una base de datos interna del sistema, por otra parte, en esta implementación no se cuenta con una propuesta para la alta de las llaves públicas de los notarios.

3.3. Rutas RESTful

Las rutas son la base en Express para definir los procesos que se llevarán acabo. Se cuenta con solamente tres rutas, ambas rutas se dirigen al recurso principal, es decir, el servicio, donde utilizando los métodos post y get definimos la función a realizar. Al utilizar el método GET obtendremos, si ningún recurso es definido (no se pasa la firma del documento como identificador del recurso a obtener) se regresa la versión del servicio. En el caso del método get (obtener, en español) si se pasa la firma del documento que se utiliza, como ya se definió previamente, como identificador del recurso, que en este caso es nuestro documento. Se realiza entonces la búsqueda de la firma del documento y en caso de existir se regresa un código HTTP 200 (OK) y la estampa de tiempo que se especifica más adelante, en caso de no existir regresa un código HTTP 404 (No encontrado).

El proceso que el método post (enviar, en español) realiza al recibir una nueva petición es la creación de la estampa de tiempo en formato ISO 8601, la

obtención del JSON enviado y el archivo original. A continuación se busca en la base de datos la existencia de otra firma idéntica. Se realiza un proceso de búsqueda para encontrar entre las llaves públicas recibidas la perteneciente al notario, esta búsqueda se realiza probando cada llave pública recibida con las almacenadas en la base de datos transformada usando base 64. Finalmente, se realiza el proceso de verificación especificado previamente. En caso de que no se encuentre otra firma idéntica, se encuentre la llave pública del notario en las llaves públicas almacenadas en la base de datos y el proceso de verificación resulte exitoso se dan de alta los datos en la base de datos, primero se da de alta la firma del documento, el hash original del documento y la estampa de tiempo, en el caso de la firma y el hash original se almacenan en base 64. A continuación, se inserta un nuevo registro para ligar el documento y el notario utilizando la llave pública del notario y la firma del documento. Además, se insertan las llaves públicas de todos los actores posterior a la remoción del notario de la lista de llaves públicas recibidas. Finalmente, se realiza la liga de las llaves públicas y la firma del documento dentro de la base de datos.En caso de que todo el proceso se lleve acabo correctamente se regresa un status HTTP 201 (Created), en el caso contrario se regresa un status HTTP 400 (Bad request).

Como podemos ver solamente se realiza el proceso de verificación en la ruta post. El firmado se debe llevar acabo localmente, para esto se ofrece una librería minificada utilizando Browserify, el cual se abordará a continuación.

3.4. Firmado

El proceso de firmado deberá realizarse localmente para evitar la transferencia de las llaves privadas. Se provee una librería JavaScript para este proceso, la librería se produce utilizando el módulo *notaria.js* y el modulo *read.js*, módulo que permite leer un archivo desde un input del DOM, lo lee utilizando FileSystem API, lo transforma de forma asíncrona y lo regresa como un buffer.

Browserify lee el archivo *index.js* y crea una versión empaquetada de las librerías necesarias para este paquete, es decir incluye la librería Crypto, en este caso se debe incluir la librería crypto-browserify.

3.5. Verificado

El proceso de verificado al igual que el proceso de firmado deberá ser realizado, cuando se busca verificar la existencia de un archivo en el sistema, de manera local. Utilizando igualmente la librería descrita en el firmado se introduce el certificado y el archivo original, se realiza el verificado de manera local y finalmente se revisa la existencia de la firma en el sistema para verificar la existencia de este documento.

3.6. Cryto-browserify

Debido a la utilización de esta librería de código abierto, se participó en la implementación de la misma debido a la falta de secciones necesarias para nuestra implementación. Se participó en la documentación de la sección de encriptación RSA con llave privada y desescripción con llave privada.

3.7. pg-execute-query

Finalmente, una de las librerías utilizadas en la implementación para la conexión con la base de datos se encuentra limitada en la implementación ya que no administra de forma sencilla la creación de consultas asíncronas. Se creó un módulo para cubrir este detalle faltante y se publicó en NPM como una librería de código abierto.

Esta librería ha recibido hasta el día de hoy más de 700 descargas y se liberó posteriormente a la finalización de la implementación presentada la versión 2 del software que mejora el manejo de errores separando el manejo de errores y el manejo de peticiones correctas cumpliendo así el formato estándar de otras librerías similares.