Sistema de copias de seguridad

Trabajo de Sistemas Orientados a Servicios

Máster Universitario en Ingeniería Informática

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Septiembre de 2014

Alicia Boya García

Índice general

CAPÍTULO 1

Introducción

Este trabajo describe Tormes, un sistema de copias de seguridad para equipos de servidor diseñado alrededor del concepto de *streaming*.

En este proyecto se han determinado los siguientes requisitos, con diferentes niveles de importancia.

- Invocar programas de copia, de la forma más flexible posible.
- Enviar copias a distintos medios de almacenamiento, posiblemente externos a una organización (*en la nube*).
- Minimizar el número de copias intermedias.
- Utilizar cifrado y códigos de autenticidad para mantener la confidencialidad e integridad de las copias de seguridad.
- Permitir gestionar de forma centralizada las copias de seguridad de diferentes equipos.

CAPÍTULO 2

Entidades implicadas

Se han identificado las siguientes entidades en el sistema.

- **Servidor maestro**: Es un proceso que almacena un índice con las copias efectuadas en el sistema y coordina a los *agentes de copia* para identificar cada copia de seguridad y determinar dónde deben ser almacenadas.

- **Agentes de copia**: Son programas que se ejecutan periódicamente en los equipos para hacer las copias de seguridad. Negocian una serie de parámetros con el servidor maestro, ejecutan un *programa archivador* y envían su salida a un *proveedor de almacenamiento*.

  Un *agente de copia* puede estar en el mismo equipo que el *servidor maestro* o en otro distinto.

- **Controlador**: Es un nodo que puede realizar acciones administrativas sobre el servidor maestro remotamente. Típicamente el nodo del servidor maestro también actúa de controlador, aunque se puede utilizar uno separado.

- **Programas archivadores**: Leen datos de un equipo local, escribiendo a su salida estándar datos que quieren conservarse para poder ser recuperados en un futuro.

  En su funcionamiento normal, los *programas archivadores* son invocados por un *agente de copia*, que debe estar en la misma máquina.

- **Programas desarchivadores**: Reciben la información generada en el pasado por un *programa archivador* para devolver un equipo a un estado anterior, cuyos datos fueron preservados.

  Los programas archivadores y desarchivadores se combinan en el mismo ejecutable. En cada ejecución recibirán una variable de entorno `$TORMES_MODE` que podrá tener el valor `archive` o `unarchive`.

- **Proveedor de almacenamiento**: Son sistemas que permiten almacenar datos con vista a recuperarlos en el futuro, de forma íntegra y sin alteraciones.

Pueden ser internos a la empresa (copias en otros equipos o en cintas) o externos (copias de seguridad *en la nube*).

2.1 Despliegue

El diseño del sistema trata al servidor maestro, los agentes de copia y el nodo controlador como nodos separados. Sin embargo, en despliegues pequeños estas entidades pueden ser albergadas en el mismo nodo computacional bajo un único sistema operativo.

Para facilitar la configuración de este tipo de despliegue, los archivos de configuración de ejemplo hacen uso de rutas comunes. Por ejemplo: existe una clave que es requerida por los nodos controladores para autenticarse contra el servidor maestro. Tanto el fichero de configuración del servidor maestro como el fichero de configuración del nodo controlador apuntan a la misma ruta, de manera que basta con crear ese fichero para que la configuración sea correcta.

La siguiente tabla muestra los distintos archivos de configuración que utiliza el sistema y qué nodos deben contenerlos.

Archivo de configuración	Servidor maestro	Agente de copias	Controlador
agent-auth.token		✓	
controller-auth.token	✓		✓
master-keys.json	✓		
master-server.pem	✓	✓	✓
master-server.key	✓		
resources.conf.coffee		✓	
storage.conf.coffee	✓		
tormes-agent.conf.coffee		✓	
tormes-controller.conf.coffee			✓
tormes-master.conf.coffee	✓		

Figura 2.1: Archivos de configuración.

CAPÍTULO 3

Configuración de Tormes

Tormes utiliza sendos ficheros de configuración, tanto en el equipo que funciona como servidor maestro como en aquellos que funcionan como agentes de copias.

Estos dos modos de funcionamiento tienen asociados archivos de configuración diferentes. En caso de que un equipo haga uso de ambos modos de funcionamiento (*modo unificado*), se hará uso de todos ellos.

3.1 Configuración del servidor maestro

- `tormes-master.conf.coffee`: Define la configuración del servidor maestro. Incluye variables para establecer en qué puerto y dirección escuchará el servidor maestro, dónde se guardará la base de datos, qué certificado y clave privada se utilizará y de qué archivo se leerá el código de autenticación del nodo controlador.

- `master-keys.json`: Almacena la clave maestra utilizada indirectamente para cifrar las copias de seguridad, así como todas aquellas claves maestras pasadas que se hayan utilizado en el sistema en el pasado pero ya no sean usadas en nuevas copias.

  El tema Seguridad describe con detalle los protocolos de seguridad de Tormes.

  Este fichero no está pensado para ser editado de forma manual, sino que se proporciona una herramienta automática denominada `tormes-keygen` que crea este fichero si no existe y añade una clave automáticamente de forma segura.

- `storage.conf.coffee`: Contiene dos propiedades:

  - `storageOptions`: Define posibles configuraciones (opciones) de almacenamiento. Cada ocpión tiene un nombre y dos propiedades: * `module`: El nombre del módulo que implementa la comunicación con el proveedor de almacenamiento. * `config`: Los parámetros que se proporcinarán al proveedor de almacenamiento.

El nombre de la opción de almacenamiento es guardado en el catálogo, por lo tanto no se deben modificar opciones existentes con las que ya se hayan hecho copias.

Ejemplo:

```
storageOptions =
  disk:
    module: 'tormes-storage-disk'
    config:
      rootDirectory: '/media/san/backups'
  aws:
    module: 'tormes-storage-aws'
    config:
      bucket: 'misbackups'
      awsConfig:
        region: 'eu-west-1'
        accessKeyId: '...'
        secretAccessKey: '...'
```

- `assignments`: Para cada nodo, define los recursos de los que pueden hacerse copias y qué opción de almacenamiento debe utilizarse para cada uno. Alternativamente, puede definirse la propiedad `selectStorage` como una función que haga esta asociación de cualquier otra forma.

  Ejemplo:

```
assignments =
  mi_nodo:
    recurso1: 'disk'
    recurso2: 'aws'
```

3.2 Configuración de los agentes de copias

- `tormes-agent.conf.coffee`: Almacena el nombre del nodo, la dirección del servidor maestro en el que registrará las copias, su certificado y permite especificar el archivo donde se leerá el código de autenticación de agente (por defecto `agent-auth.token`).

- `resources.conf.coffee`: Almacena la lista de recursos de los que se podrá hacer copia de seguridad, incluyendo en cada caso un nombre, el programa archivador y una serie de parámetros para éste.

  Los parámetros serán enviados al proceso archivador a través de variables de entorno con el nombre `TV_` (*Tormes Variable*) seguido por el nombre de parámetro especificado en la configuración.

  Ejemplo:

```
resources = [
  {
```

```
            name: 'homes'
            archiver: 'tormes-archiver-files'
            sourcePath: '/home'
            compression: '-z'
            incremental: yes
            snarBaseDir: '/var/snar'
          }
        ]
```

3.3 Configuración del nodo controlador

- `tormes-controller`: Indica la dirección del servidor maestro, su certificado y el código de autenticación.

3.4 Lenguaje de configuración

La mayoría de ficheros de configuración de Tormes son ficheros ejecutables en lenguaje CoffeeScript.

Utilizar un lenguaje de programación para configuración permite una gran flexibilidad, al poder recoger datos de configuración de cualquier fuente o generarlos de maneras novedosas.

CAPÍTULO 4

Herramientas desarrolladas

Tormes está diseñado para ser utilizado desde la consola de los sistemas operativos basados en UNIX. Por este motivo incluye una serie de ficheros ejecutables para realizar diferentes operaciones con el sistema, los cuales pueden añadirse a la variable de entorno `$PATH` si se desea.

Cada uno de estos programas admite los modificadores `-h` o `-help` para consultar de forma detallada su uso y la lista de opciones que aceptan.

4.1 Programas para el servidor maestro

- `tormes-master`: Ejecuta una instancia del servidor maestro, que atenderá peticiones de los agentes de copias y actualizará el catálogo de copias.

  Durante el desarrollo se puede utilizar la siguiente orden para que este servidor sea reiniciado (y actualizado) al hacer cambios en los ficheros que lo conforman.

```
nodemon $(which tormes-master).coffee
```

- `tormes-keygen`: Genera una nueva clave maestra y la añade al fichero de claves.

4.2 Programas para los agentes de copias

- `tormes-backup`: Realiza una copia de seguridad en el recurso del nodo local especificado como primer argumento. Adicionalmente se pueden proporcionar más argumentos, en cuyo caso serán entregados al programa archivador.

- `tormes-archiver-files`: Se trata de un programa archivador para hacer copias de seguridad de ficheros. Utiliza la utilidad GNU tar, soportando copias incrementales.

  A diferencia de los demás ejecutables, no está escrito en CoffeeScript sino en Bash.

- `tormes-header`: Dentro de un programa archivador, escribe a la salida estándar una cabecera que permite a `tormes-backup` determinar si una copia es incremental, y en este caso cuál es su copia padre. Esta cabecera consiste de un documento JSON delimitado por un carácter terminador `NUL`.

- `tormes-restore`: Restaura una copia de seguridad. Como argumento debe recibir un código *ticket* generado con `tormes-grant-ticket`.

4.3 Programas para el controlador

- `tormes-catalog`: Consulta el catálogo de copias. Permite filtrar por nodo, recurso y fecha.

- `tormes-auth`: Gestiona las claves de autenticación de los agentes de copias. Permite añadir (`-a`), eliminar (`-r`) y listar claves (`-l`).

- `tormes-grant-ticket`: Solicita al servidor maestro un ticket de restauración de una copia de seguridad cuyo id es indicado como argumento.

CAPÍTULO 5

Protocolos de comunicación

De forma añadida a los protocolos propios de cada proveedor de almacenamiento, los agentes de copias y los nodos controladores necesitan comunicarse con el servidor maestro para intercambiar información de gestión: por ejemplo, que una copia va a ser realizada o que se quiere consultar el catálogo de copias.

Inicialmente para este propósito se utilizó una API basada en peticiones HTTP específicas, por ejemplo */start-backup* para iniciar un proceso de copia. La implementación de cada punto de acceso HTTP se hacía de manera manual utilizando las funcionalidades de red de *node.js* y el framework minimalista *express.js*.

Conforme el proyecto avanzó, se vio la necesidad de minimizar el esfuerzo de implementación de nuevos métodos remotos y la repetición de código, así como simplificar la gestión de errores. Para este fin se adoptó un protocolo RPC.

Se barajó la implementación manual de un protocolo RPC sencillo contra la utilización de bibliotecas ya construidas al efecto. Finalmente se adoptó una solución mixta: se usó la biblioteca *Jayson* que provee una implementación del protocolo JSON-RPC en servidor y cliente para *node.js*. A esta biblioteca se le hicieron varios añadidos propios:

- Una función envoltorio para definir los puntos de entrada con promesas [1] en vez de *callbacks*, siguiendo el estilo de programación del resto del proyecto.

- Funciones envoltorio adicionales para controlar la autenticación. Las funciones envueltas no son ejecutadas si la autenticación es incorrecta y recibirán un parámetro adicional con la identidad del nodo iniciador previamente verificado.

- Captura y serialización automática de excepciones. Algunos tipos particulares de excepciones han sido registradas para uso en RPC. Cuando un proceso servidor arroja una de ellas, una función envoltorio automáticamente la captura y la envía al proceso cliente.

    - Las excepciones RPC no producen una traza de error en el proceso servidor. Sin embargo, la producirán en el cliente si no son capturadas.

[1] Las promesas son una forma de describir código asíncrono.
https://github.com/petkaantonov/bluebird#what-are-promises-and-why-should-i-use-them

- Funciones envoltorio para las peticiones de los clientes, trabajando también con promesas y realizando la deserialización de excepciones registradas. En caso de producirse un error de red, automáticamente muestran un error apropiado y terminan la aplicación. Se han desarrollado diferentes funciones para los diferentes tipos de autenticación (sin autenticación, agente de copias, controlador).

CAPÍTULO 6

Proceso de copia

En la realización de una copia se han identificado los siguientes pasos.

1. **Invocación de ‘‘tormes-backup‘‘ en el agente de copias.**

   Los sistemas operativos incluyen distintas herramientas para la ejecución de tareas periódicas, tales como *cron*.

   El administrador de sistemas puede configurar cualquiera de estas utilidades para que periódicamente invoque a `tormes-backup` especificando un recurso y argumentos. Por ejemplo, en el caso de copias incrementales, este argumento puede indicar el nivel de copia [2]

2. **Negociación con el servidor maestro.**

   El agente de copias envía un mensaje de iniciación de copia (`startBackup`) al servidor maestro indicando qué identificador de nodo tiene y qué recurso va a copiar.

   El servidor maestro consulta la tabla de asociaciones de `storage.conf.coffee` para determinar en qué proveedor de almacenamiento deberá ser almacenada dicha copia, y qué parámetros le proporcionará a este proveedor.

   En caso de que el servidor maestro no encuentre una asociación válida, se enviará una respuesta nula y el proceso de copia se cancelará.

   En caso de que sí se encuentre una asociación, se generará además un identificador de copia. Los identificadores de copia son numéricos, únicos e incrementales. Este identificador será enviado al agente de copia junto con la información de almacenamiento.

[2] En copias incrementales, el nivel de copia es un número entero que indica de cuántas otras copias depende una dada. Un nivel 0 es una copia base (no incremental). Una copia de nivel 1 depende de la copia inmediatamente anterior en el tiempo de nivel 0, y así sucesivamente con niveles superiores.

El servidor maestro guarda en su base de datos una pequeña colección para leer y escribir el último identificador asignado y trata con cuidado la asignación de estos para evitar condiciones de carrera que pudieran ocasionar que dos copias recibieran el mismo identificador.

3. **Invocación del archivador**

   El proceso `tormes-backup` invocará en segundo plano al proceso archivador que tenga asignado a ese recurso en `resources.conf.coffee`.

   En una primera etapa `tormes-backup` esperará leer una cabecera JSON delimitada por un caracter `NUL` con metadatos sobre la copia. Estos metadatos indicarán si la copia que se va a realizar tiene dependencias con otras copias anteriores, y en caso afirmativo, cual es el identificador de la copia padre.

4. **Redirección al proveedor de almacenamiento**

   Una vez que los metadatos han sido leídos correctamente, en una segunda etapa, se redirigirá toda la salida del proceso archivador a una clase `Writer` del módulo proveedor de almacenamiento especificado por el servidor maestro.

   El módulo proveedor de almacenamiento es un módulo de Node.js que cumple una interfaz concreta, definiendo dos clases de tipo `Stream`: una clase `Reader`, para descargar archivos del proveedor de almacenamiento y una clase `Writer` para enviarlos.

5. **Terminación del archivador y confirmación de almacenamiento**

   Eventualmente el proceso archivador terminará. Como cualquier proceso UNIX, el archivador devolverá un código de retorno. Si este código es cero, se entenderá que los datos que ha generado son correctos y se confirmará la copia con el proveedor de almacenamiento.

   Al confirmar la copia el proveedor devolverá un objeto de *ubicación* que permitirá recuperar la copia posteriormente desde ese mismo proveedor de almacenamiento.

   En caso contrario se entenderá que ha habido un error y se abortará la transferencia, llamando al método `.abort()` de la clase del proveedor de almacenamiento.

6. **Registro en el catálogo de copias**

   Finalmente, se enviará al servidor maestro un mensaje de finalización (`finishBackup`) para añadir la nueva copia al catálogo de copias.

   Este mensaje especificará:

   - El nodo que ha hecho la copia.
   - El recurso respaldado.

- El identificador de copia asignado.
- El identificador de la copia padre, si procede.
- Fecha y hora en la que comenzó la copia, en UTC [3].
- El nombre del módulo proveedor de almacenamiento.
- El objeto de ubicación devuelto por el proveedor de almacenamiento.

Si el servidor maestro no encuentra inconsistencias en los datos proporcionados

[3]Universal Coordinated Time.

CAPÍTULO 7

Proceso de restauración

Restaurar una copia de seguridad con Tormes es un proceso en pasos.

En un primer paso, el administrador conecta al servidor maestro, consulta las copias disponibles y genera un *ticket* de restauración.

En un segundo paso, el administrador conecta al servidor donde quiere restaurar la copia y utiliza el ticket para descargar esa copia de seguridad.

7.1 Generación del ticket

1. **Invocación de ‘‘tormes-restore-ticket‘‘**

   La orden `tormes-restore-ticket`, ejecutada en el servidor maestro, acepta como parámetro un identificador de copia y devuelve un código aleatorio denominado *ticket* que puede ser utilizado para restaurar la copia desde otro nodo.

   El servidor maestro mantiene una colección de tickets en su base de datos. Cada ticket está asociado a un identificador de copia e incluye una fecha de expiración, de manera que los tickets de restauraciones antiguas puedan ser eliminados automáticamente.

7.2 Uso del ticket

2. **Invocación de ‘‘tormes-restore‘‘**

   La orden `tormes-restore`, ejecutada en el equipo de destino, acepta como parámetro un *ticket* generado en el paso anterior.

3. **Negociación con el servidor maestro**

El equipo de destino envía el *ticket* al servidor maestro en un mensaje de iniciación de restauración (`startRestore`).

El servidor maestro comprueba que el *ticket* exista y no haya expirado. En caso positivo, consulta el catálogo de copias buscando el identificador de copia asociado. Si la copia es incremental, consultará también los datos de las copias padres hasta llegar a una copia base.

El servidor maestro devolverá una lista ordenada de copias, indicando para cada una en qué proveedor de almacenamiento se encuentra y el objeto de ubicación. Si las copias están cifradas, se consultarán las semillas y se calcularán y enviarán las claves de descifrado de cada una de ellas.

4. **Descarga e invocación del desarchivador**

   El equipo de destino iniciará la descarga de las copias utilizando los módulos proveedores de almacenamiento especificados y paralelamente ejecutará el programa desarchivador, alimentándolo con los archivos de respaldo descifrados.

   Este programa desarchivador es el encargado de reconstruir el estado del sistema a partir de los ficheros de respaldo.

   En caso de trabajar con copias incrementales, se invocará una vez el programa desarchivador para cada copia.

CAPÍTULO 8

Comunicación

A continuación se describen los métodos RPC implementados por el servidor maestro, incluyendo sus parámetros y el tipo de cliente que puede utilizarlos.

- `startBackup`

  Requiere autenticación de agente de copias. Recibe los siguientes parámetros:

  - `node`: El nombre del nodo solicitante. Este parámetro se extrae automáticamente del objeto de autenticación del agente de copias, para evitar su manipulación.
  - `resource`: El nombre del recurso a copiar.

  Negocia un proceso de copia. Consulta el proveedor de almacenamiento configurado para el par nodo-recurso proporcionado y obtiene un nuevo identificador de copia.

  Devuelve un objeto con los siguientes valores:

  - `storageModule`: Nombre del módulo proveedor de almacenamiento.
  - `storageParams`: Parámetros para el proveedor de almacenamiento. El agente de copias entregará este objeto sin modificación al proveedor de almacenamiento.
  - `backupId`: Identificador de copia asignado.
  - `finishTicket`: Un código que el agente utilizará posteriormente para comunicar la finalización de la copia.
  - `secondaryKey`: Claves de cifrado y autenticación de mensaje que deben ser utilizadas para cifrar la copia.

- `finishBackup`

  Requiere autenticación de agente de copias. Recibe los siguientes parámetros:

  - `node`: El nombre del nodo solicitante.

- `finishTicket`: El ticket de finalización generado por `startBackup`.
- `parent`: Identificador de la copia padre, o `null` si no procede.
- `location`: Objeto de ubicación del proveedor de almacenamiento.

Inserta los datos de una nueva copia de seguridad en el catálogo de copias de seguridad.

No devuelve ningún valor.

- `listBackups`

  Requiere autenticación de controllador. Recibe un único parámetro con un objeto filtro en sintaxis de NeDB [4].

  Deveulve una lista de metadatos de copia, los cuales incluyen:

  - `node`: Nodo en el que se originó la copia.
  - `resource`: Recurso copiado.
  - `id`: Id de copia.
  - `storageOption`: Almacenamiento utilizado.
  - `encryption`: Datos (públicos) de cifrado. Incluye: * `masterKeyId`: El identificador de la clave maestra utilizada. * `cipherSeed`: Semilla utilizada para generar la clave de cifrado. * `macSeed`: Semilla utilizada para generar la clave de autenticación de mensaje. * `date`: Fecha en la que se realizó la copia, en formato ISO 8601. * `location`: Objeto de ubicación. * `parent`: Id de copia padre o `null` si no procede.

- `grantTicket`

  Requiere autenticación de controllador. Recibe un único parámetro con un id de copia.

  Devuelve un objeto con una propiedad `ticketId` que contiene un código que puede ser utilizado por cualquier agente de copias para restaurar la copia indicada.

- `startRestore`

  No requiere autenticación. Recibe un único parámetro con un código *ticket* generado por la orden `grantTicket`.

  Devuelve una lista con los datos de las copias que debe restaurar. En caso de copias incrementales, la lista está ordenada de manera que las copia raíz esté en el primer lugar. Cada elemento de la lista contiene:

[4] https://github.com/louischatriot/nedb#finding-documents

- `id`: Id de copia
- `resource`: Recurso copiado.
- `storage`: Objeto con dos propiedades: * `module`: Módulo de node.js con el proveedor de almacenamiento (ej. `tormes-storage-aws`). * `config`: Parámetros del proveedor de almacenamiento.
- `date`: Fecha de creación de la copia.
- `location`: Objeto ubicación que puede ser utilizado para restaurar la copia.
- `encryption`: Objeto con las claves de cifrado y autenticación de mensaje utilizadas en la copia.

- `authorizeNode`

  Requiere autenticación de controlador. Recibe un único parámetro con el nombre de un nodo.

  Genera y devuelve una nueva clave aleatoria que podrá ser utilizada por el agente de copias de ese nodo para autenticarse.

- `listAuthorizedNodes`

  Requiere autenticación de controlador. No requiere ningún parámetro.

  Devuelve una lista con todos los nodos autorizados. Cada uno se representa como un objeto con las propiedades:

  - `name`: Nombre del nodo.
  - `code`: Clave de autenticación.

- `revokeAuth`

  Requiere autenticación de controlador. Recibe un único parámetro con el nombre de un nodo.

  Elimina a un nodo de la lista de nodos autorizados. No devuelve ningún valor.

CAPÍTULO 9

Seguridad

Tormes incorpora una serie de mecanismos de seguridad, explicados a continuación.

9.1 Cifrado de copias

Tormes permite cifrar las copias de seguridad para garantizar un nivel de confidencialidad independiente del proveedor de almacenamiento escogido.

En caso de utilizarse, el servidor maestro envía al agente de copias un par semilla-clave durante la negociación. El agente de copias deberá escribir en la cabecera del fichero que envía a los proveedores de almacenamiento la semilla en claro (sin cifrar), seguida del contenido del fichero cifrado con la clave recibida.

El cifrado se realiza con AES de 256 bit, con el modo de operación *Contador* (CTR[5]). Este modo de operación es paralelizable tanto en cifrado como en descifrado, ofreciendo así un mejor rendimiento comparado con modos más tradicionales como *Encadenamiento de Bloques Cifrados* (CBC[6]) y simplifica la implementación al no requerir un método de relleno [7].

Se utiliza como *nonce* o *vector de inicialización* (IV) de CTR un vector de 256 bit relleno de ceros. Esto no supone una vulnerabilidad porque el sistema está diseñado de forma que dos copias no se cifren nunca con la misma clave.

[5] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Counter_.28CTR.29

[6] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher-block_chaining_.28CBC.29

[7] Un método de relleno es un algoritmo utilizado para codificar información binaria de longitud arbitraria en un número variable de bloques de tamaño fijo. El método de relleno debe permitir recuperar la longitud del mensaje codificado de manera no ambigua. Los métodos de relleno han sido la base de los ataques más recientes contra SSL, tales como *BEAST* (2011), *CRIME* (2012), *BREACH* (2013) y *POODLE* (2014).

9.2 Autenticación

Las copias enviadas a los proveedores de almacenamiento no sólo contienen información cifrada, sino que además añaden códigos de autenticación de mensaje para asegurar que los datos no puedan ser manipulados, y que los datos utilizados en la recuperación sean exactamente los mismos, bit a bit, que los enviados en el momento del respaldo.

Estos códigos de autenticación forman una parte importante del esquema de cifrado de Tormes, ya que el modo *CTR* hace funcionar a cifradores de bloques como AES como cifradores de corriente, los cuales son maleables [8].

La integridad tiene una importancia añadida en Tormes ya que en ciertas situaciones los mensajes cifrados pueden contener código ejecutable (por ejemplo, SQL o copias de archivos ejecutables). Es un requisito de seguridad que no se procese (y por tanto, potencialmente se ejecute) ningún mensaje cuya autenticación no haya sido previamente validada.

Para este fin, se utilizará el algoritmo de autenticación de mensajes basado en funciones *hash*, *HMAC*, utilizando la función hash SHA-2 con un tamaño de bloque de 256 bits (referida a veces como *SHA-256*).

El algoritmo HMAC recibe un flujo de datos de longitud arbitraria y una clave privada de autenticación. El resultado es un código de autenticación del tamaño de la salida de la función hash. Este código se almacena junto al texto cifrado. Por las propiedades del algoritmo HMAC, no es posible recuperar el flujo de datos a partir del código de autenticación y – suponiendo que la función hash no tenga vulnerabilidades – no es posible alterar un mensaje obteniendo otro al que le corresponda el mismo código de autenticación, ni generar un código de autenticación a partir de un mensaje sin conocer la clave privada de autenticación.

9.2.1 Autenticación continua

En el caso de Tormes, el procesamiento de las copias de seguridad se realiza en *streaming*. Esto impone requisitos adicionales sobre el sistema de autenticación de mensajes: No basta con colocar un código de autenticación al final del archivo que contenga toda la copia de seguridad, puesto que habría que descargar el fichero completo y comprobar su autenticidad antes de poder procesar su contenido para cumplir con el requisito de no procesar datos no autenticados.

Para hacer frente a este problema, los datos de las copias de seguridad se dividen en paquetes de hasta 1 MiB, al final de los cuales se coloca un código de autenticación HMAC. De esta forma es posible alimentar al programa desarchivador con los datos de estos paquetes sin necesitar memoria intermedia para almacenar el fichero completo.

[8] La maleabilidad de un cifrado define la facilidad de un atacante para modificar un texto cifrado de manera que se descifre como otro elegido por él.

Para mayor seguridad, los códigos de autenticación de cada paquete están encadenados: El flujo de entrada de cada autenticador HMAC consiste en la salida del anterior, concatenada con los datos de ese paquete.

9.3 Derivación de claves

La semilla y clave se calculan en el servidor maestro a partir de un generador de números aleatorios de calidad criptográfica (*CSRPNG*) y una clave maestra mediante un algoritmo de *derivación de claves* basado en funciones hash. Este algoritmo se denomina *HMAC-based Key Derivation Function* o *HKDF* y está descrito en el RFC 5869[9].

La versión de HKDF implementada en Tormes no require las funciones de expansión opcionales definidas en el RFC, ya que utiliza bloques del mismo tamaño (256 bits) tanto para la clave como para la semilla (también denominada *salt*) y para la salida de la función hash empleada que será utilizada como clave secundaria.

9.4 Comunicación con el servidor maestro

El servidor maestro escucha conexiones únicamente en HTTPS, utilizando un certificado autofirmado que debe configurarse en los agentes de copias y el nodo controlador. Por defecto este certificado se busca en el fichero `master-server.pem` en el directorio de configuración de Tormes.

Junto a los ficheros de configuración de ejemplo se incluye un script, `make-cert.sh`, que genera un certificado automáticamente.

9.5 Autenticación de los agentes de copias

El servidor maestro mantiene una base de datos con los agentes de copias habilitados en el sistema. Esta base de datos se puede manipular con la herramienta `tormes-auth` desde un nodo controlador.

Para cada agente de copias se registra el nombre de nodo y un código de seguridad generado aleatoriamente.

Las peticiones de los agentes de copia deberán incluir ambos campos. Las peticiones sin un nombre y código correcto son rechazadas automáticamente.

La comparación de claves se realiza con el módulo de node.js `scmp` para hacer evitar vulnerabilidades frente a ataques de tiempo.

[9] http://tools.ietf.org/html/rfc5869