DECIDE EUROPA MIXNET

Grupo 1

Opera ID: 128

Antolín Bermúdez, Miguel Ángel: **Implicación**: 5
De la Torre Díaz, David: **Implicación**: 5+
Rodríguez Aránega, Cristian: **Implicación**:5

Sevilla, enero de 2019

Enlaces de interés:

- Repositorio de trabajo.
- Enlace a Heroku.
- Enlace a Travis-ci.
- Enlace a la wiki del equipo.

Resumen

El equipo de decide Europa Mixnet se propuso como primer objetivo desarrollar unas pruebas de cero conocimiento con las cuales se puede verificar que el voto que llega a la aplicación, es de la persona que dice ser, no ha sido modificado por un tercero y nosotros no hemos abierto ese correo.

Consideramos que en un sistema de voto electrónico garantizar esta propiedad es uno de los puntos más importantes.

Para realizar esta funcionalidad invertimos mucho tiempo leyendo artículos relacionados con las pruebas de cero conocimiento, encontramos un código abierto de *matlab* el cual utilizamos de base y se adaptó a *Python* con la opción de que un usuario pudiera comprobar distintas claves.

Introducción y contexto

Una vez repartidos los grupos de Decide, nuestro equipo decidió realizar los cambios en la rama de seguridad de la aplicación, la Mixnet. Durante las primeras semanas discutimos cuál de las mejoras integrar en nuestro proyecto. La mejora que más nos llamó la atención fueron las pruebas de cero conocimiento.

Las pruebas de cero conocimiento son uno de los apartados más importantes en el terreno de la seguridad estas se encargar de garantizar que el voto que llega a la aplicación es verídico (es de quien dice ser, no ha sido alterado y no se ha abierto el voto). A raíz de esto buscamos información teórica de estas pruebas y encontramos un artículo. En este artículo se explica de manera bastante profunda toda la base teórica para realizar dichas pruebas.

A raíz de esta base teórica encontramos un algoritmo en el lenguaje de programación *Matlab,* se realizaron muchos cambios para ejecutarlo en el entorno de desarrollo de Decide. No obstante se consiguió implementar la funcionalidad. Además se ha añadido la posibilidad de que el usuario pueda introducir las variables necesarias para realizar las pruebas, y así comprobar el correcto funcionamiento de estas.

La funcionalidad descrita anteriormente es la más importante de nuestro desarrollo, no obstante también empezamos a investigar con la finalidad de modificar el sistema de encriptación por uno más sencillo, optamos por la encriptación simétrica (ECB). Se encontró una librería de *Python* que nos permitiría implementar fácilmente esta funcionalidad.

Actualmente esta funcionalidad se encuentra en desarrollo ya que actualmente está implementada en el sistema para que un usuario pueda verificar la

correcta encriptación y desencriptación con este sistema. No se usa como sistema de encriptación para Decide.

Descripción del sistema

Definimos diferentes subsistemas que son más o menos independientes entre sí y que se interconectan implementando una API concreta.

Cada subsistema se encarga de una tarea concreta en el sistema de voto. Puede haber tareas que impliquen modificaciones en más de un subsistema, por lo que habrá que coordinar el desarrollo para que puedan comunicarse entre sí.

Autenticación

Autenticación de votantes. En una votación debemos autenticar a los votantes y asegurarnos de que no se vota más de una vez o de que el votante puede votar en esa votación. Para ello debemos autenticar a la persona y esta autenticación ha de ser segura, de tal forma que se reduzca la posibilidad de fraude, suplantación de identidad, etc.

Posibles formas de autenticación: * Usuario / Contraseña * Enlace único por correo electrónico * Enlace único por SMS o sistema de mensajería (whatsapp, telegram, etc) * Certificado FNMT * Redes sociales (twitter, facebook, google).

Censo

Una parte importante de una votación es el censo, que es el grupo de personas que pueden votar en una votación. Gestionar el censo es una tarea importante a la hora de definir una votación y de controlar quién ha votado y quién no.

El censo está relacionado con la autenticación, pero no es lo mismo, el censo es la autorización para votar en una votación en concreto.

Votaciones

Una votación puede definir una o varias preguntas, con diferente número de opciones y diferentes configuraciones. Puede ser voto simple, ordenación de una lista de opciones, elección múltiple, etc.

Cabina de votación

Interfaz para votar. Este subsistema se encarga de mostrar la interfaz de voto de una votación en concreto, permitiendo al votante votar de la forma más sencilla posible.

Almacenamiento de votos (cifrados)

Los votos se almacenan cifrados en una base de datos, donde tenemos la relación directa de votante y voto. No se puede saber la intención de voto porque el voto estará cifrado en la base de datos, pero sí tendremos la información de quién ha votado y quién no.

Recuento / MixNet

Subsistema que se encarga de la parte criptográfica de una votación. La mixnet es una red de ordenadores que generarán una clave compartida para cifrar los votos. De esta manera sólo se podrá descifrar un voto cuando todas las autoridades se pongan de acuerdo, a la hora de hacer el recuento.

En el proceso de recuento se desliga el votante del voto, cada autoridad aplicará un paso de barajado de votos, antes de descifrar su parte, de tal forma que cuando se obtenga el listado de votos en claro, no habrá forma de saber qué ha votado cada usuario.

Post-procesado

Una vez realizado el recuento, tenemos una lista de números, este subsistema se encarga de traducir esa lista de números a un resultado coherente con el tipo de votación.

Por ejemplo, si es una votación de tipo simple, simplemente habrá que contar el número de veces que aparece cada opción y se dará como ganadora la opción con más opciones.

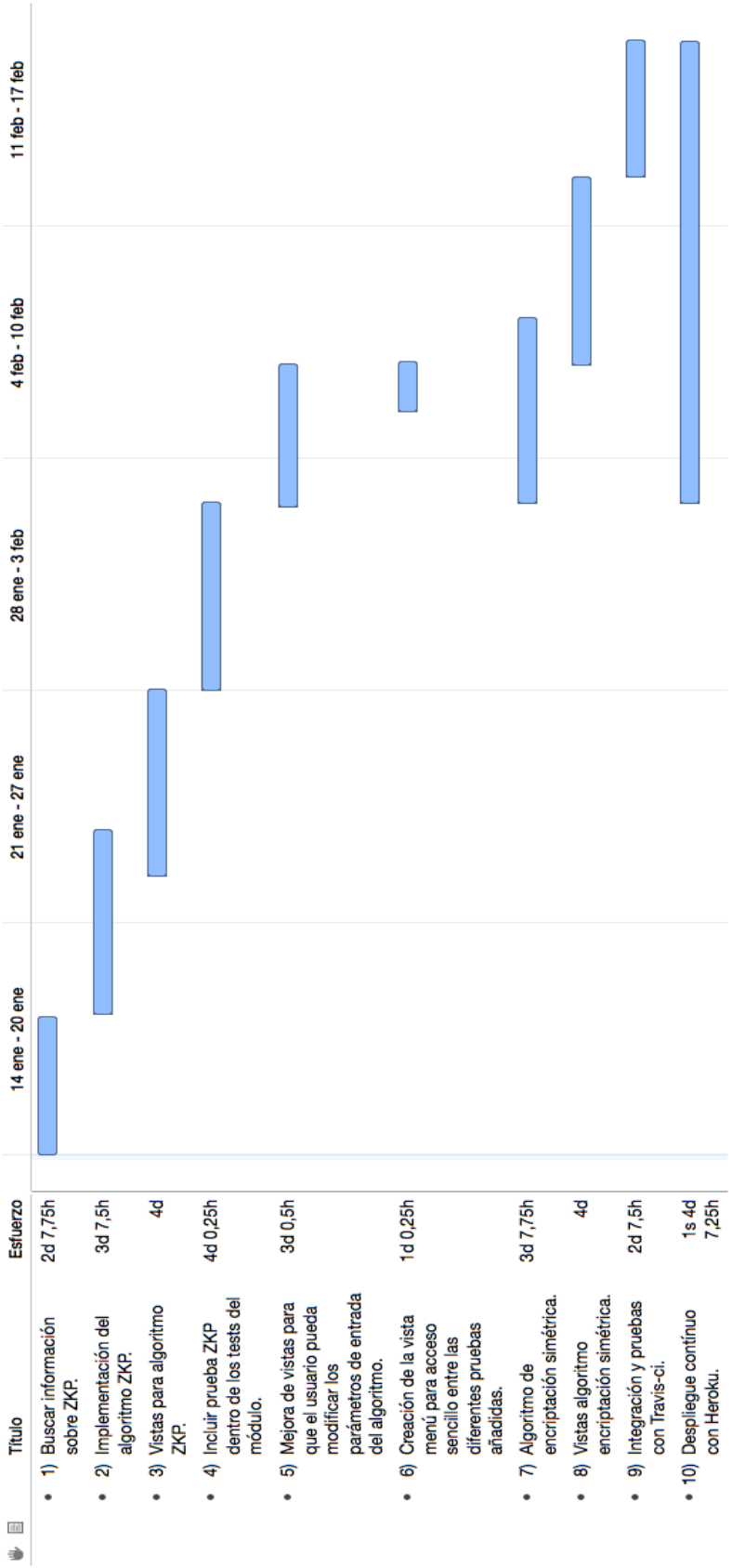
Este subsistema también es el encargado de aplicar diferentes reglas a los resultados, por ejemplo, aplicar reglas de paridad, ley d'Hondt, etc.

Visualización de resultados

Este subsistema es el encargado de coger los datos obtenidos tras el post procesado y pintarlos de una manera gráfica y entendible, generando gráficas y tablas.

También se encargará de generar los diferentes informes necesarios de estos resultados, ya sea en formato web, pdf, texto plano, etc.

Planificación del proyecto



Cada uno de los integrantes del equipo, ha elegido una incidencia, y ha ido asignándose las tareas una vez finalizada la anterior.

Entorno de desarrollo

Como editor de código se ha utilizado *Visual Studio Code,* en su última versión: 1.30. Este editor es muy ligero y nos permite descargar las funcionalidades necesarias en cada momento de manera sencilla y eficiente.

Como entorno de desarrollo se ha utilizado uno virtual para *Python,* el cual se encuentra aislado del resto de módulos del sistema. A continuación facilitamos los pasos para la instalación y correcto funcionamiento:

- Instalación:
    - Crear entorno de trabajo: `python3 -m venv decide-env`
    - Accedemos al entorno: `source decide-env/bin/activate`
    - Instalar dependencias: `pip3 install -r requirements.txt`

- Ya podemos ejecutar los comandos necesarios dentro de nuestro entorno virtual:
    - `Python3 manage.py makemigrations mixnet`
    - `Python3 manage.py migrate`
    - `Python3 manage.py runserver`

Una vez se han ejecutado los comandos mencionados arriba nuestra aplicación se lanzará de forma automática en localhost:8000. Si por algún motivo el puerto se encontrará utilizado podríamos ejecutar la comanda del runserver con el puerto a utilizar al final.

Gestión de incidencias

Gestión de incidencias internas:

Todas las incidencias deberán de respetar la siguiente estructura:

Título de las issues:

Ha de contener un título descriptivo con el fin de poder concretar de qué se trata dicha issue.

Contenido:

El contenido de la issues ha de tener al menos los siguientes elementos:

- Breve descripción sobre el problema a tratar.
- Pasos a seguir para replicar el error.
- Traza del error, si la hubiera.
- Adicionalmente, se puede incluir información como la versión de las herramientas que estamos utilizando o cualquier elemento que el autor considere de utilidad para la resolución de la misma

Etiquetado:

Se deberá incluir una de las siguientes etiquetas proporcionadas por git a la hora de crear una incidencia con la finalidad de localizar rápidamente de qué error se trata.

- Bug: La incidencia describe un fallo de programación en el software.
- Duplicate: Ya existe otra incidencia que describe los mismo queesta.
- Enhancement: Una solicitud de mejora en el software.
- Help wanted: Esta incidencia describe una petición de ayuda.
- Invalid: Esta incidencia es incorrecta, refleja una petición que no tiene sentido.
- Question: Esta incidencia describe una pregunta a responder para el equipo de desarrollo.
- Wontfix: Esta incidencia no tiene solución posible. Se emplea cuando se describe como incidencia un comportamiento que parece ser defectuoso, pero que en realidad es una funcionalidad. También se puede emplear para una incidencia que describe un problema, para la que no existe solución posible. Hay que tener una buena razón para etiquetar una incidencia de esta manera.

Niveles de prioridad

Se ha establecido unos niveles de prioridades de las incidencias según la urgencia a resolver estas para el correcto funcionamiento del proyecto:

- Low Problema leve, resolverlo cuando las issues de más rango estén resueltas. Por ejemplo fallos de visualización, faltas de ortografía etc.
- Medium Problema que afecta a la funcionalidad de la aplicación pero con el cual la aplicación sigue funcionando.
- High Problema que afecta a gran parte de las funcionalidades creadas. Arreglar con la mayor rapidez posible.
- Critical Problema que rompe las funcionalidades del proyecto. Arreglar con urgencia. Todo el equipo se deberá de poner a trabajar para arreglar este tipo de issues

Desarrollo

Cuando un miembro del equipo crea una issue deberá de reflejarla en la pestaña de 'Projects' de github (https://github.com/mruwzum/decide-europa-mixnet/projects/1) de esta manera todos los miembros del equipo sabrán issues se deben arreglar y en el estado que están:

- To do: En esta columna se escribirán las issues a realizar
- In progress: En esta columna se escribirán las incidencias en las que se está trabajando en este momento
- Revisión: En esta columna se escribirán las issues que están siendo revisadas.
- Done: En esta columna se escribirán las issues finalizadas.

Ejemplo de una Issue de nuestro proyecto:

Si se accede a: https://github.com/mruwzum/decide-europa-mixnet/issues/9 podrá ver una incidencia cerrada.

En esta imagen se puede ver como se han respetado todos los aspectos que se han nombrado anteriormente. Título descriptivo, una descripción más profunda del problema, pasos para la reproducción y detalles del sistema.

Esta *issue* tiene dos etiquetas: Bug y High, tal y como también se ha descrito anteriormente con la finalidad saber rápidamente qué tipo de problema hay que solucionar

En respuesta de esta *issue* se ha respondido con el *commit* en la cual se ha resuelto definitivamente.

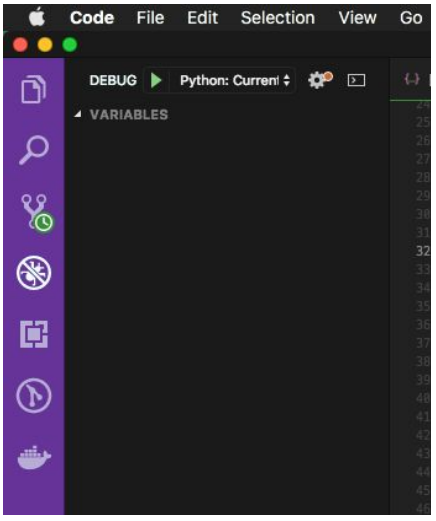
Gestión de incidencias externas

Se tratarán de forma idéntica a las internas, la incidencia externa se tratará como interna y se copiarán sus propiedades.

Gestión de depuración

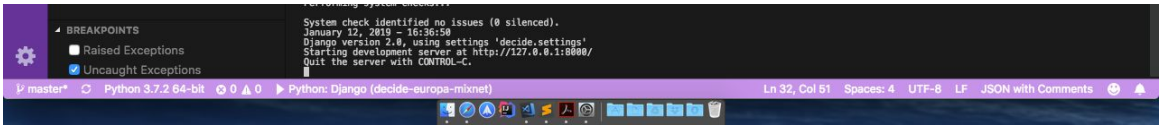
Gracias a las herramientas de Visual Studio Code para el desarrollo y la depuración de código, podemos hacer debug tras la siguiente configuración:

En la pestaña debug de VS Code, añadimos un nuevo debugger para python:



Después, en el json de configuración, editamos la línea donde se encuentra la ruta del manage.py, para indicarle que el nuestro se encuentra en decide/manage.py.

Ya podemos ejecutar en modo depuración y el intérprete parará ahora en los puntos que le indiquemos para poder realizar la depuración.



Gestión del código fuente

Guía de commits:

Los commits contendrán una etiqueta en el título que describe el tipo de cambio que se ha realizado, estos estados han sido obtenidos de la guía.

- Feat: Una nueva característica.
- Fix: Se soluciono un bug.
- Docs: Se realizaron cambios en la documentación.
- Style: Se aplicó formato, comas y puntos faltantes, etc; Sin cambios en el código.
- Refactor: Refactorización del código en producción.
- Test: Se añadieron pruebas, refactorización de pruebas; Sin cambios en el código.
- Chore: Actualización de tareas de build, configuración del admin. de paquetes; Sin cambios en el código.

Los commits deben de tener un título descriptivo y además adjuntar una breve descripción con el cambio que se ha añadido. Si el commit realizado arregla una funcionalidad anteriormente creada, el título del commit deberá ser el mismo pero con un identificador numérico. Por ejemplo si se ha subido el archivo zkp.py al repositorio con el commit "pruebas de cero conocimiento", la siguiente versión de deberá ser "pruebas de cero conocimiento 1". De esta manera se podrá localizar con facilidad la versión del código.

Al realizar un commit, el miembro del equipo que lo realice debe asegurarse de que el código que va a subir funciona y compila sin errores. Si un miembro del equipo sube un código con errores debe ser el que arregle esa parte del código.

En el caso de haber un merge se deberá de poner en contacto con el miembro que ha tenido el conflicto para llegar a un acuerdo. Si no se llegara a una conclusión se dialogará con el jefe del proyecto.

Ejemplo de commit:

En esta imagen se ve un commit con la etiqueta de [FIX] en el cual aparece un título descriptivo del problema que se resuelve y un comentario ampliado del problema en cuestión.

¿Cómo se gestiona el repositorio?

Este repositorio es un fork de decide-Europa, del cual hemos creado Decide-Europa-Mixnet en este se encuentran todos los miembros del equipo de Mixnet. La coordinación del grupo se llevará a cabo en el grupo de Whatsapp del equipo. En este todos los miembros del equipo pueden trabajar sobre master si así lo desean o crear una rama propia con la finalidad de probar cambios.

Dentro de Decide-Europa-Mixnet se ha creado un entorno virtual para hacer nuestras pruebas con la finalidad de estar aislados.

Gestión de la construcción e integración continua

Para el proceso de integración continua hemos utilizado la herramienta proporcionada en clase *Travis-ci.* Con ella cada vez que se sube un nuevo commit a master la aplicación de *Travis-ci* vuelve a ejecutar el código que hemos programado anteriormente para verificar que el código funciona correctamente.

Para más información sobre el CI/CD, mirar la wiki del equipo.

Gestión de liberaciones, despliegue y entregas

Para el proceso de liberación de código, despliegue y entregas de forma contínua se ha usado el servicio en la nube Heroku.

Heroku en un servicio similar a Travis, vinculandolo con Github podemos crear un pipeline de automatización que en cada commit detectado de nuestro proyecto, cree un nuevo desplegable.

Para las entregas, se enviará al cliente el enlace de la versión pertinente desplegada en Heroku.

Los entregables tendrán nombre decide-europa-mixnet_vXX con la versión correspondiente del software a la iteración.

Para más información sobre el CI/CD, mirar la wiki del equipo.

Mapa de herramientas

En el esquema presentado en la siguiente página se muestran los programas que se han utilizado para el desarrollo de nuestro proyecto.

El código fuente se ha programado en *Visual Studio Code,* en este el código es programado en *Django.* Cuando se realizan los cambios, estos se suben a la plataforma de *Git.*

A continuación se ejecutan los procesos de *Travis-ci* y *Heroku.* Los cuales se encargan de comprobar si los cambios que se han añadido son funcionales.

Ejercicio de propuesta de cambio

Para realizar un cambio funcional en Decide-Europa-Mixnet, se seguirán los siguientes pasos:

1. Como primer paso se crea una necesidad de cambio a realizar.
2. Esta necesidad se expresa en el diario del equipo en el apartado de *projects* del equipo.
3. El equipo decide cómo repartir esta faena o un miembro del equipo se asigna esa tarea desde el apartado de *projects.*
4. El miembro del equipo que realiza el cambio, cuando tiene un cambio funcional sube un commit a su rama o al master.
5. Los procesos de *Travis-ci* y *Heroku* se disparan automáticamente con ese *commit*, comprobarán si el código que se ha subido es funcional. Si este fallara se enviaría un correo al encargado de *Travis-ci* y *Heroku.* En este se informará del error.
6. Si hubieran errores, se crearía una *Issue* y se notificará al usuario que ha subido dicho cambio para que arregle dicho error.

Conclusiones y trabajo futuro

Como conclusiones de este trabajo el equipo de Decide-Europa-Mixnet ha llegado a las siguientes conclusiones:

- Se ha aprendido un nuevo *framework* de desarrollo web basado en *Python,* exactamente *Django*. Nos ha parecido un *framework* muy interesante y rápido de aprender, además de potente.
- Al escoger el módulo de *Mixnet* nuestro equipo ha tenido mucho trabajo de investigación para poder realizar las pruebas de cero conocimiento. A pesar de no haber cursado ninguna asignatura de seguridad en esta escuela hemos podido realizar el trabajo de una manera satisfactoria.
- El equipo desconocía la herramienta de *Heroku, la cual* nos ha parecido muy interesante para poder desplegar de manera contínua esta aplicación.
- Trabajar en un equipo grande (más de tres personas) puede llegar a ser complicado, más cuando ciertas personas no tienen el mismo objetivo con el trabajo a realizar.

Como posibles mejoras para el siguiente año se podría acabar de implementar la encriptación de manera más sencilla con ECB o cualquier criptosistema simétrico que los alumnos quieran utilizar.