ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA DE SOFTWARE

ESTABLECIMIENTO DE CLAVES Y AUTENTICACIÓN MEDIANTE LA UTILIZACIÓN DE CÓDIGOS SONOROS EN ENTORNOS MÓVILES

KEY AGREEMENT AND DEVICES AUTHENTICATION OVER SOUND CODES

Realizado por
RICARDO RUIZ TUEROS

Tutorizado por
ISAAC AGUDO RUIZ

Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2016

Fecha defensa:

El Secretario del Tribunal

Resumen

Este proyecto tiene como objetivo investigar la posibilidad de realizar un intercambio de claves y autenticar a dos dispositivos utilizando como medio el canal sonoro. Investigaremos a nivel teórico la posibilidad de incluir información en un sonido orientado al intercambio de claves, teniendo en cuenta factores como la necesidad de sincronizar los sonidos, la escucha de los dispositivos, la longitud de la clave que podemos alcanzar, los ataques a los que serían vulnerable el protocolo, la necesidad de hacer persistente cierta información a nivel local...

Tras plantear a nivel teórico una posible solución investigaremos diversas tecnologías actuales que puedan soportarla, analizándolas y comparándolas entre ellas trataremos, en última instancia, de elaborar un prototipo que sea capaz de autenticar a varios dispositivos (al menos a dos) y realizar un intercambio de claves entre ellos, creando así un canal seguro a través del cual puedan comunicarse sin necesidad de que una tercera parte confiable pueda manejar información criptográfica sensible.

Palabras clave: Investigación, Sonido, Intercambio de claves, Autenticación, Seguridad, Tercera parte confiable, Prototipo.

This project has as achievement to investigate the possibility of making a key agreement and authenticate two devices using the sound media. We will research at a theoretical level the possibility of including information in a sound wave, being aware of different facts as the need of synchronize the sounds and listens states of the devices, the key length we can reach, the attacks which could vulnerate our protocol, the necessity of persist some data locally…

After stating a theoretical solution we will investigate different current technologies that could help us achieve it, we will analyze and compare the solutions trying to elaborate a prototype which would be able to authenticate several devices (at least two of them) and make a key agreement between them, making a safe channel over they can communicate without a trusted third party that can manipulate the cryptographic critical data.

Keywords: Research, Sound, Key Agreement, Authentication, Security, Trusted Third Party, Prototype..

Índice

Establecimiento de claves y autenticación mediante la utilización de códigos sonoros en entornos móviles

1. Ámbito del proyecto

Este proyecto pretende abordar la problemática del establecimiento de claves y autenticación entre dos dispositivos de forma segura para poder establecer un canal de comunicación entre ellos.

En la actualidad se utilizan una serie de protocolos seguros para realizar este intercambio de claves, como puede ser el conocido *Diffie-Hellman*, sin embargo, nuestra propuesta es la utilización de un canal *"fuera de banda"* , que nos permita realizar la autenticación de los dispositivos sin necesidad de certificados y de una *"tercera parte confiable"* como pudiera ser una Autoridad Certificadora (CA), sino entre los propios dispositivos.

En este sentido hemos decidido abordar el problema de forma similar a como se *"autentican"* dos personas en el mundo real. En principio, no hay necesidad de un certificado que diga quien eres, simplemente realizamos un reconocimiento mutuo del otro individuo con la información visible. Esto es precisamente lo que tratamos de alcanzar, una autenticación *"espontánea"* de dispositivos próximos mediante el canal sonoro.

La siguiente cuestión lógica que podríamos plantearnos es: *"¿Cómo podrían autenticarse dos dispositivos?"*, la comunicación entre dispositivos digitales en la actualidad se realiza prácticamente en su totalidad a través de Internet, pero eso implicaría la transmisión de datos a un servidor, a una red en la que participan millones de usuarios y que podría ser atacada de multitud de formas, necesitamos un canal de comunicación exclusivo entre los dos dispositivos.

¿Quizá Bluetooth?, podría ser una solución, sin embargo también existen numerosos ataques a esta tecnología, el alcance Bluetooth puede ser de varios metros con una antena pequeña, pero, por ejemplo, con una antena direccional de gran tamaño, podemos alcanzar distancias de kilómetros que dan mayor rango al atacante. Además en muchos casos somos dependientes de la plataforma (por ejemplo, no se puede emparejar un dispositivo con Android con uno que tenga iOS).

NFC *(Near Field Communicaciton)* es una de las tecnologías actuales con la que se están realizando numerosas aplicaciones, si bien no está exenta de problemas al no poderse apreciar cuando se realizan conexiones de forma no solicitada. Simplemente acercando otro dispositivo al nuestro, ya seríamos susceptibles de ser atacados[1].

Además la tecnología NFC no está tan extendida entre los dispositivos, si bien es cierto que cada vez son más los smartphones que la incorporan aún no se ha extendido en otros dispositivos como ordenadores y tampoco se sabe aún hasta qué punto será compatible entre distintas plataformas, como por ejemplo Apple[2].

Podemos utilizar un canal de comunicación aún más básico, las personas intercambian secretos mediante susurros, esto es, en definitiva sonido… ¿Y si un dispositivo pudiera *"susurrar"* un secreto a otro?.

Eso es, el canal sonoro cumple las condiciones que necesitamos: Es un canal de dispersión entre esos dos dispositivos que puede ser controlado, sencillamente, mediante el volumen. Es independiente de la plataforma, ya sea un ordenador, un smartphone, una tablet… Con cualquier sistema operativo y hardware el sonido sigue siendo el mismo (o inapreciablemente distinto), sumado a que hoy en día casi cualquier dispositivo digital incorpora un micrófono y un altavoz lo hacen el canal ideal para *"susurrar secretos"* entre dispositivos.

---

1 : Contactless payment scam
http://www.telegraph.co.uk/technology/2016/02/17/if-you-have-a-contactless-card-watch-out-for-this-scam/

2 : Apple Pay with NFC
http://www.aple.com/apple-pay/

2. Estudio de soluciones y tecnología actual

El estudio teórico de una solución a este problema nos llevó encontrar un artículo llamado "*Tactile One-Time Pad: Leakage-Resilient Authentication for Smartphones" [1].*

Este artículo que propone como solución de autenticación al usuario un código One-Time Pad basado en la vibración del dispositivo, no utiliza técnicas de códigos sonoros, pero analiza y evalúa las posibilidades de comunicación con el dispositivo móvil (incluso sonido) en el apartado *"3.1 Potential Communication Channels"*.

En este punto comenzamos a valorar las posibilidades de sincronizar dos dispositivos, para lo cual el canal háptico (vibración) no es del todo viable y, valorando las opciones del punto anteriormente mencionado, nos decidimos por el canal sonoro.

Investigando en concreto la posibilidad de sincronización de dispositivos utilizando el canal sonoro llegamos a otro artículo: "*Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound" [2]*, en el que se analiza y evalúa la utilización de sonido ambiente por dos dispositivos móviles como segundo factor en la autenticación, si el sonido ambiente grabado por los dos dispositivos es el mismo, podemos asumir que se encuentran en la misma localización.

Entre otros puntos se definen en este documento el esquema de intercambio de mensajes para realizar la autenticación, función de comparación de ondas, análisis de seguridad basados en escenarios de ataque, prototipos y evaluaciones empíricas tanto de resistencia al ruido como de usabilidad para los usuarios.

En este punto comenzamos a valorar la posibilidad de sincronizar dos dispositivos, para lo cual utilizando el canal sonoro disponemos de tres alternativas:

- **Utilizando el sonido ambiente**: Tal como sugiere *"Sound Proof"*, si el sonido ambiente de los dispositivos es similar podemos asumir que están en la misma localización.

- **Mediante emisión de un dispositivo predecible:** Un dispositivo externo puede emitir un patrón predecible para los dispositivos cuya escucha garantice que ambos dispositivos se encuentran en la misma localización.

- **Mediante un patrón de sonido emitido por el propio dispositivo:** Es la solución que hemos escogido debido a que minimiza las posibilidades de ataque por suplantación, en ambos casos anteriores el sonido ambiente o el patrón predecible pueden ser conocidos por el atacante, mientras que si el sonido es emitido por uno de los dispositivos puede ser parcialmente aleatorio y de intensidad controlada por el emisor.

Una vez planteada una posible solución a nivel teórico nos dispusimos a investigar la tecnología desarrollada en la actualidad en relación a este ámbito.

Encontramos *"Signal360 (Anteriormente SonicNotify)" [3]* una tecnología propietaria en la que participa Oracle, permite que el dispositivo reciba notificaciones con información por ultrasonidos, se orienta a la obtención de información adicional de un evento concreto, por ejemplo, los grupos que están actuando en un festival de música o para fines de marketing y publicidad, sorteos en eventos, compartición de imágenes mediante redes sociales, etc… Desgraciadamente al ser una solución propietaria no cuenta con un SDK público.

"Audio Modem: Data over sound" [4] es la siguiente tecnología que encontramos en cuanto a códigos sonoros. desarrollada por la empresa francesa Appdilium, su página se analizan las distintas frecuencias que podrían ser utilizadas y razona la utilización de un rango de frecuencias de 18.4kHz-20.8kHz (ultrasonidos).

Además, utilizan su propia implementación para la modulación de onda, siguiendo el algoritmo DBPSK que permite hacer más robusto el canal gracias a la redundancia de datos, así como facilitar la de-modulación con un oyente de tipo no-coherente.

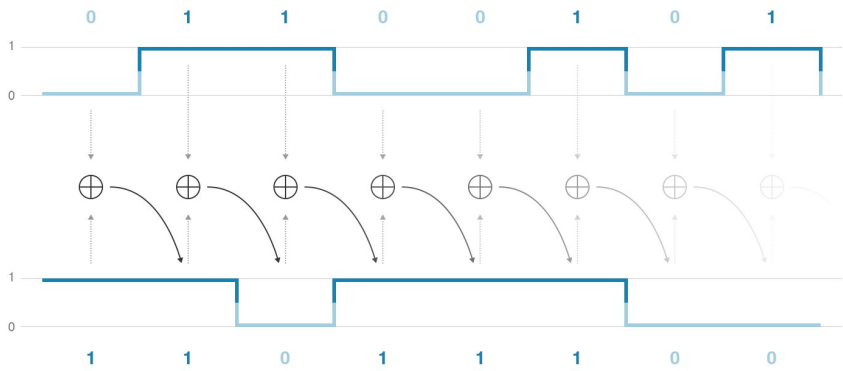


Figura 1: Modulación DBPSK[1]

Para la sincronización de los dos dispositivos plantean la utilización de los "Códigos de Barker", de forma que la transmisión se realiza con el siguiente esquema

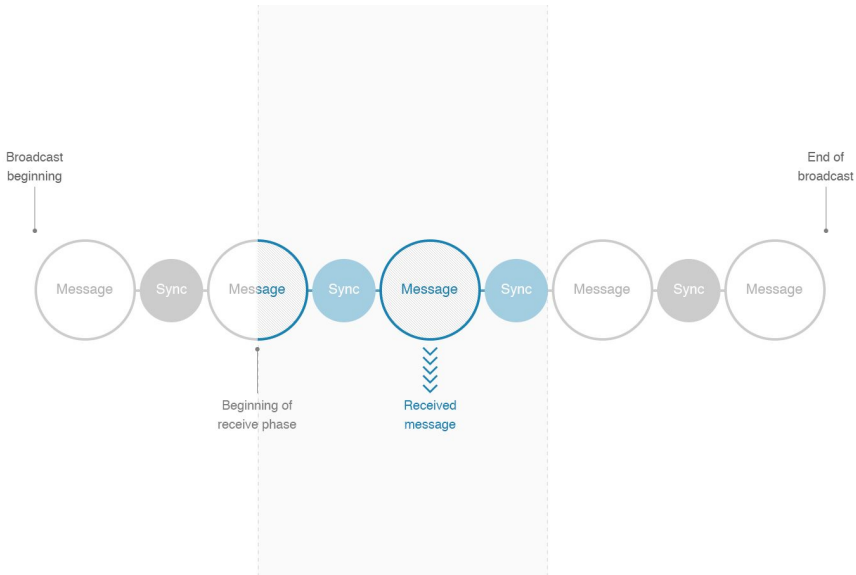


Figura 2: Esquema de transmisión con Códigos Barker[2]

Figuras 1 y 2 extraídas de:
https://applidium.com/en/news/data_transfer_through_sound/

1 : Differentially encoded Binary Phase-Shift Keying
https://en.wikipedia.org/wiki/Phase-shift_keying#Example:_Differentially_encoded_BPSK

2 : Barker Code
https://en.wikipedia.org/wiki/Barker_code

Sin embargo, Audio Modem no ofrece tampoco un SDK, aunque si un proyecto de prueba para iOS que es bastante desestructurado y críptico por lo que no hemos podido obtener conclusiones sobre su código. Pese a esto su análisis es muy interesante y nos hizo considerar que el código sonoro debía contar con dos características fundamentales: modulación y redundancia.

En este punto encontramos una aplicación relativamente nueva llamada *"Chirp" [5]* y que tiene como finalidad el intercambio de archivos multimedia entre dos o más dispositivos cercanos mediante el *"piar"* de un pájaro de forma sencilla.

En realidad, y dada la rapidez con la que se puede enviar un video de varios megabytes, lo que se envía a través del sonido es un enlace al archivo, que ha sido alojado en su servidor. Al recibirlo el otro usuario lo descarga en su aplicación y lo visualiza casi de forma instantánea.

En su sitio web hemos podido comprobar que tienen un SDK en desarrollo y que permiten su uso para desarrolladores registrados a petición expresa y durante un periodo de tiempo limitado, por lo que procedimos a crearnos una cuenta y a solicitar las claves necesarias para el uso del SDK con objeto de realizar este proyecto bajo la Universidad de Málaga. Una vez el equipo de Chirp aceptó nuestra solicitud pudimos comprobar que cuentan con un SDK tanto en Android como en iOS así como en plataformas web.

Aunque tiene una orientación al envío de contenido multimedia, como utilizan en su aplicación, hemos decidido utilizar parte de la funcionalidad que ofrecen para nuestro proyecto, que se explicará con más detalle en el apartado de soluciones propuestas.

En la actualidad alcanzar esta tecnología es el objetivo de grandes empresas, prueba de ello es que Google publicara su artículo de *" función cercana" [11]* durante el desarrollo del proyecto, en el que pretenden utilizar el canal sonoro y las redes cercanas como medio para realizar distintas actividades, como compartir fotos o jugar en línea, para ello están tratando de utilizar las redes wifi detectadas, la tecnología bluetooth, y también el sonido, como hemos propuesto en este proyecto, por lo que se trata de una tecnología novedosa y con interés para grandes empresas que pronto podrá estar implantada en nuestros dispositivos.

3. Soluciones propuestas

En el desarrollo de una solución teórica al intercambio de claves y autenticación de los dispositivos móviles necesitamos asumir que el atacante no tendrá acceso acceso a los dos canales de comunicación que utilizamos de forma simultánea, sí podrá escuchar canal sonoro en el momento de emisión u observar la información del servidor.

Se hace necesario, por tanto, la elaboración de un protocolo de comunicación que garantice la creación de un canal seguro entre ambos dispositivos.

Así, hemos llegado a plantear tres alternativas del protocolo mejoradas respecto de la anterior en iteraciones sucesivas que nos han llevado a una solución óptima en cuanto a los parámetros que estábamos considerando.

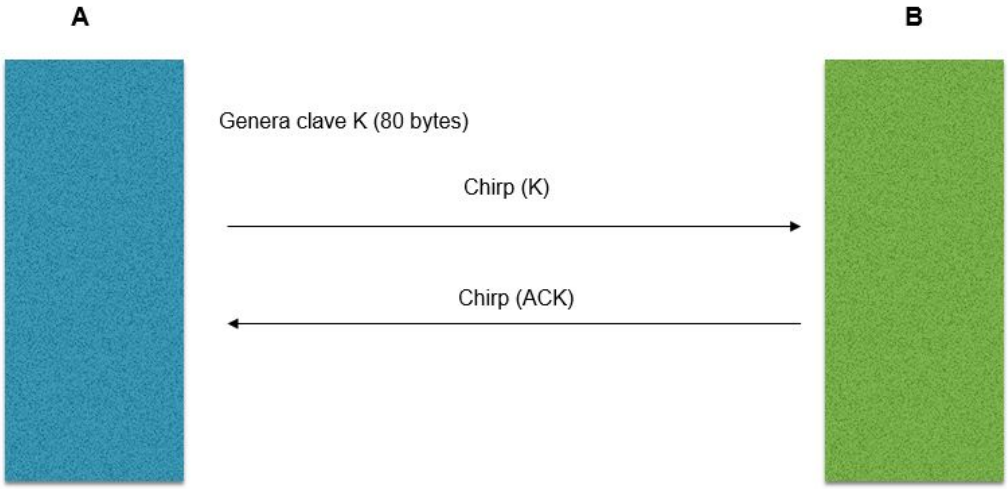


Figura 3: Primera iteracion del protocolo

La primera aproximación consistía en el envío directo de la clave mediante códigos sonoros, sin embargo, esta opción es susceptible de un ataque evidente y muy simple: La simple escucha de otro dispositivo, aunque este es un problema difícil de solventar enviar la clave en claro no parece la mejor opción.

Además, y aunque lo explicaremos en el apartado siguiente, podemos adelantar que Chirp solo nos permite enviar 80 bytes por mensaje entre dispositivos sin pasar por su servidor, lo cual reduce el tamaño de nuestra clave o nos obliga a enviarla en varios mensajes, lo que puede resultar tedioso para el usuario y puede llevar a problemas de sincronización.

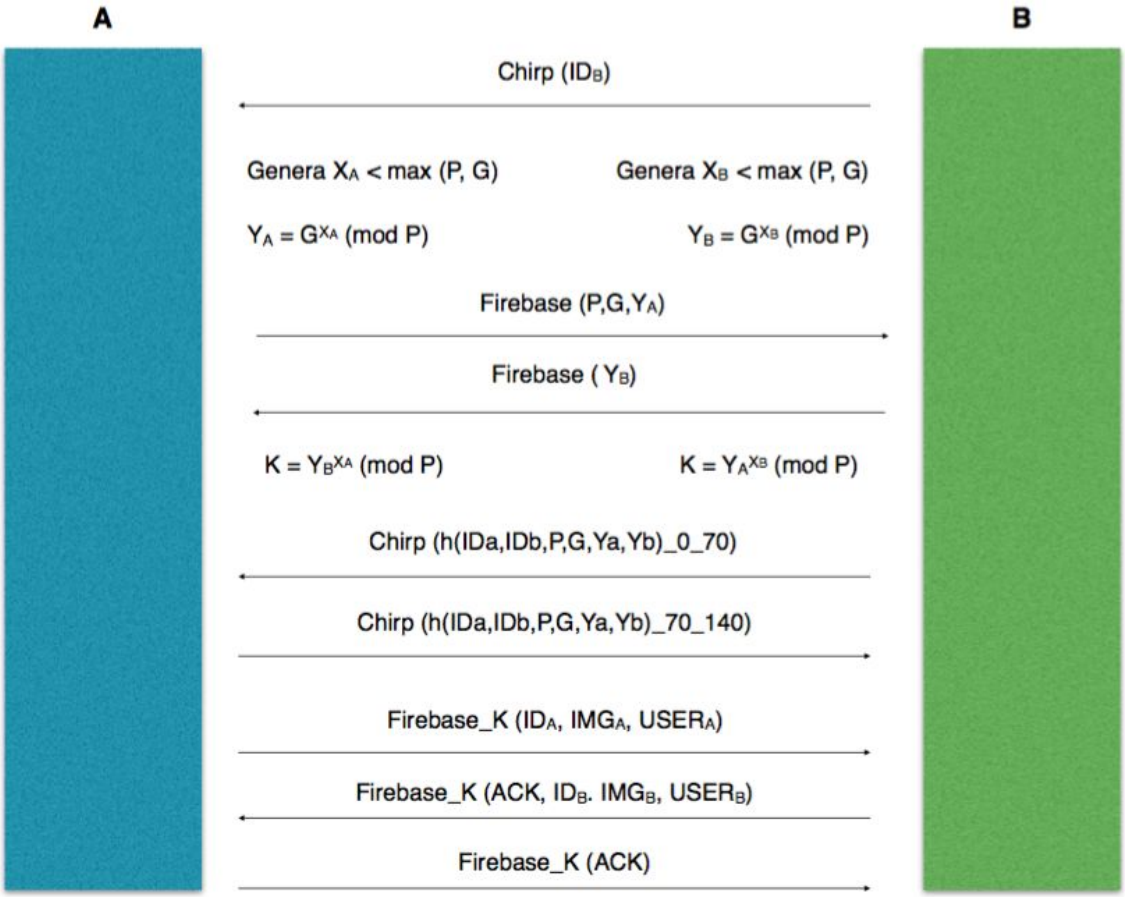


Figura 4: Segunda iteración del protocolo

La segunda aproximación que planeamos fue utilizar el protocolo Diffie-Hellman iniciado con una comunicación Chirp. De esta forma podría realizarse la autenticación del protocolo mediante códigos sonoros con el *hash* de la clave compartida, salvando así la necesidad de una *"tercera parte confiable"*, sería similar a la autenticación mediante los códigos visibles de Telegram[1] en ambos dispositivos.

1 : Telegram MTProto FAQ
https://core.telegram.org/techfaq

Sin embargo, aunque este esquema es más seguro, el principal problema de esta es que se necesitan enviar al menos tres mensajes con códigos sonoros, uno para iniciar la comunicación con el ID y dos más para la autenticación con el HMAC

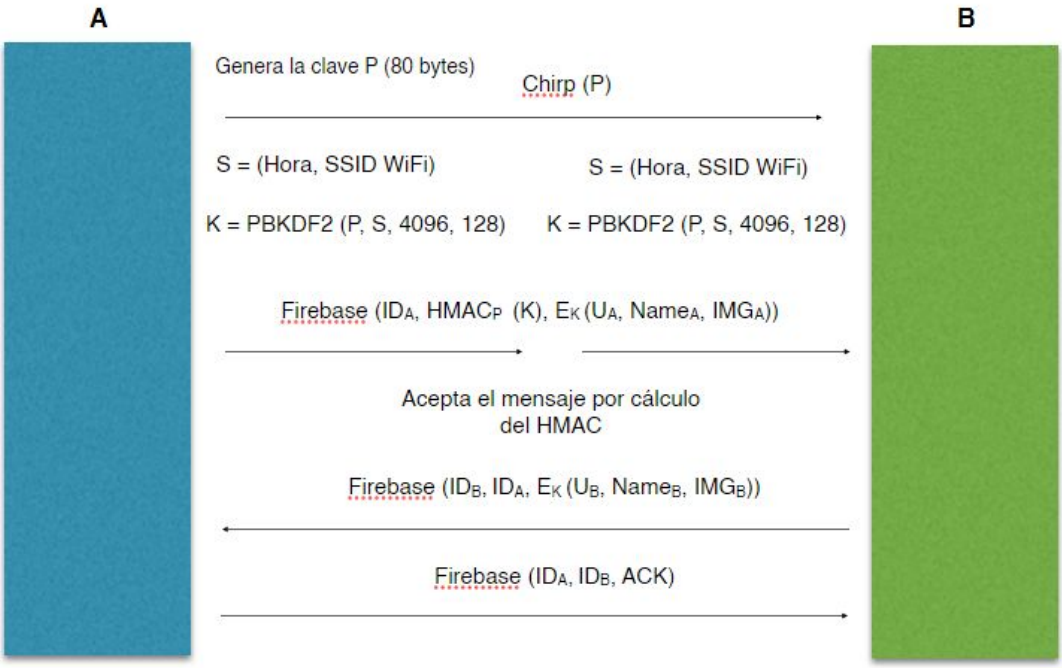


Figura 5: Tercera iteración del protocolo

La tercera aproximación fue desarrollar nuestro propio protocolo basándonos en la primera idea, pero tratando de solucionar el tamaño de la clave y limitando algunos ataques que podrían realizarse, para ello hacemos uso de la función PBKDF2 (*Password-Based Key Derivation Function 2 [6])* que nos permitirá a partir de la clave corta intercambiada con Chirp generar una clave de mayor extensión dada una *"salt".*

Para la generación de esta *"salt"* que debe ser la misma en los dos dispositivos utilizamos dos parámetros:

- Hora actual: En horas y minutos, de esta forma sabemos que la sincronización se está llevando a cabo en un instante de tiempo determinado y podemos evitar que se grabe la clave y se utilice con posterioridad, lo que se conocen como *"Replay Attacks[1]".*

1 : Replay Attacks in cryptography

http://www2.imm.dtu.dk/~fnie/Papers/GBDN07.pdf

- Redes WiFi: Además del código sonoro podemos comprobar que ambos dispositivos se encuentran próximos si reciben las mismas redes WiFi. Para ello utilizamos el SSID algunas de las redes con mayor señal. De esta forma podemos evitar los conocidos como *"Wormhole Attacks[1]"* en redes Ad-hoc, además de añadir entropía a la clave generada con PBKDF2.

Una vez calculada la clave por ambos dispositivos se procede a enviar la información privada del usuario (nombre e imagen) cifradas así como un código HMAC que servirá para autenticar al receptor y confirmar que la clave se ha calculado correctamente.

Si el código HMAC es correcto se procede a añadir la información transmitida como un nuevo contacto y establecer una ID para el canal de comunicación con el mismo. En caso contrario la información se descarta y no se responde con la información propia al mensaje recibido.

La información de los contactos añadidos se almacena de forma local en la aplicación y no es accesible por el servidor. Las IDs se generan aleatoriamente en cada comunicación, por lo que se almacenan las ID utilizadas por el emisor y el receptor en la sincronización y se emplean (en un orden determinado) para establecer la ID del canal de comunicación entre ellos.

El objetivo de esta sincronización es que únicamente se envíen al servidor dos tipos de información: IDs aleatorias de comunicación y mensajes cifrados.

De esta forma, aunque alguien pudiera atacar y observar la información del servidor no podría conocer el contenido de los mensajes enviados, ni siquiera podría identificar usuarios concretos, ya que en cada canal de comunicación utilizan ID distintos, garantizamos así no solo la confidencialidad de los mensajes sino también la privacidad de los usuarios.

---

1 : Wormhole Attacks In Wireless Sensor Networks
http://www.cs.uml.edu/~glchen/papers/wormhole-cipbook07.pdf

Si comparamos la entropía de la clave obtenida, el número de mensajes enviados a través del canal sonoro y la posibilidad de ataques de repetición obtenemos la siguiente tabla:

	Entropía	Número de mensajes mediante canal sonoro	Susceptible Ataques Replay
Clave en claro	80 bytes	1	Si
Diffie-Hellman	Más de 80 bytes	3	No
Protocolo con PBKDF2	80 bytes + WiFi + Fecha	1	No

Podemos observar que en cuanto a los parámetros considerados que el protocolo propuesto es que mejor se ajusta ofreciendo un compromiso entre la entropía que es capaz de utilizar en la clave y la cantidad de mensajes que son necesarios enviar a través del canal sonoro.

El primer protocolo nos ofrece una solución simple sin necesidad de enviar varios mensajes a través del canal sonoro pero es poco seguro ya que es susceptible de ataques de repetición y su única entropía son los 80 bytes generados con Chirp.

El segundo de ellos es una extensión del protocolo Diffie-Hellman, nos ofrece todas sus ventajas además de evitar los ataques del tipo *“Man In The Middle”* gracias a la autenticación de los usuarios mediante el canal sonoro. Sería la solución ideal en términos de seguridad pero la necesidad de sincronizar tres mensajes sonoros lo hace poco usable y lento.

El tercer protocolo propuesto es una mejora sobre el primero en el que, gracias a la generación de una clave con PBKDF2 con una *salt* adecuada, podemos evitar lo ataques de repetición y de *“agujero de gusano”* además de añadir entropía a la propia clave mediante el uso de las redes WiFi cercanas y la fecha actual. Mantiene la usabilidad de la primera versión al necesitar un único mensaje a través del canal sonoro.

4. Tecnologías utilizadas

En este punto se explicarán las tecnologías que se han decidido utilizar en la elaboración del prototipo.

La plataforma sobre la que se desarrolla el prototipo es iOS, esto se debe principalmente a que la primera tecnología que tratamos de utilizar fue *Audio Modem* y el proyecto de prueba que traía es exclusivo de iOS. Además nos pareció interesante desarrollar para esta plataforma ya que no se ve nada de ella en la Universidad y mi tutor me informó de que contaban con un perfil de desarrollo académico acordado entre Apple y la Universidad de Málaga del que podíamos hacer uso.

Son cuatro las tecnologías y APIs utilizadas, aunque no la únicas que he investigado, ya que hemos evaluado otras alternativas como *Audio Modem*, la API de *Telegram* para el envío de mensajes o utilizar un servidor web en lugar de Firebase.

El principal problema de *Audio Modem* como hemos comentado es que no incluye un SDK propiamente dicho, sino sólo un proyecto de prueba en iOS con un código fuente bastante críptico del que no pudimos obtener resultados.

Por otra parte la API de *Telegram* parece una buena solución, pero al tratar de utilizar su proyecto en iOS hemos tenido muchos problemas para firmar el código y desplegar la aplicación ya que no disponemos un certificado aprobado por *Telegram* además, como veremos, necesitamos controlar distintos tipos de mensajes que pueden recibir los usuarios en nuestro protocolo y el API de Telegram no nos da control directo sobre esto.

A continuación se exponen las tecnologías utilizadas explicando brevemente la parte de su API que hemos empleado en el prototipo:

Chirp

Chirp es una tecnología que se basa en la comunicación de información entre dispositivos utilizando el canal sonoro.

Para ello emplean una asociación unívoca entre un byte y una determinada frecuencia, de forma que el carácter 'a' podría tener asociada la frecuencia de 500Hz, el carácter 'b' la de 510Hz…

Distinguen dos tipos de mensajes en su SDK:

Los *"shortcodes"* son mensajes de un máximo de 80 bytes, enviados en forma de 10 caracteres que no pasan por el servidor de Chirp y que se envían a través del canal sonoro de un dispositivo a otro siguiendo la codificación en frecuencias anteriormente explicadas.

Los mensajes con diccionario, por otra parte, son estructuras de datos más complejas que se envían con una clave, esta clave es, precisamente, un *"shortcode"* de los anteriormente mencionados. Estos mensajes con diccionario son, en última instancia, una clave de 10 caracteres que identifica un mensaje en formato JSON con la información estructurada.

Los mensajes con diccionario a diferencia de los mensajes *"shortcode"* si necesitan subirse al servidor de Chirp. En su funcionamiento, el usuario final recibe mediante el canal sonoro un *"shortcode"* que utiliza como *"token"* frente al servidor de Chirp para obtener la estructura de datos en formato JSON.

A continuación comentamos brevemente las funciones que Chirp nos facilita:

```
## Initialising the SDK

Before using the Chirp SDK, you must set your app key and secret.
If you don't have an app key, please [register as a Chirp developer](https://developers.chirp.io).

```
  [[ChirpSDK sdk] setAppKey:YOUR_APP_KEY
                  andSecret:YOUR_APP_SECRET
             withCompletion:^(BOOL authenticated, NSError *error)
  {
      if (authenticated) { NSLog(@"Authenticated OK."); }
  }];
```
```

Figura 6: Función setAppKey

La función setAppKey es la primera que debemos llamar para comenzar a utilizar el SDK (representado por el objeto sdk), en ella introduciremos nuestra clave

y secreto (credenciales solicitados a Chirp personalmente) para autenticarnos como un desarrollador registrado en su servidor.

```
## Sending shortcodes when offline

An **identifier** is a ten-character string of characters from Chirp's 32-character alphabet, `[0-9a-v]`.

```
    Chirp *chirp = [[Chirp alloc] initWithIdentifier:@"a7cnn9c0li"];
    [[ChirpSDK sdk] chirp:chirp withCompletion:^(Chirp *chirp, NSError *error)
     {
         NSLog(@"Completion: %@", chirp.identifier);
         if (error)
         {
             NSLog(@"ERROR: %@", error);
         }
     }];
```
```

Figura 7: Funciones initWithIdentifier y chirp

La función *chirp* nos permite enviar un *shortcode* o un mensaje de tipo diccionario mediante el canal sonoro, para ello es necesario dar el mensaje (un string para el *shortcode* o un diccionario para el mensaje con diccionarios) al objeto sdk mediante la función *initWithIdentifier*, posteriormente basta con llamar a la función *chirp* en la que se puede especificar un bloque de código *"withCompletion"* que sea llamado tras la emisión del sonido.

```
## Receiving shortcodes when offline

```
    [[ChirpSDK sdk] setOfflineOnly:YES];
  [[ChirpSDK sdk] startListening:^(Chirp *chirp, NSError *error)
  {
    // Receive a chirp
    if (!error)
    {
      NSLog(@"Heard chirp with id : %@", chirp.identifier);
    }
  }];
```
```

Figura 8: Funciones setOfflineOnly y startListening

La primera de estas funciones, *setOfflineOnly,* nos permite especificar al objeto *sdk* que no realizaremos ninguna comunicación con el servidor en el envío de mensajes, no es estrictamente necesario, ya que si no utilizamos ningún diccionario no se enviará al servidor, pero es conveniente desactivar esta funcionalidad.

La función *startListening* activa el micrófono del dispositivo en modo escucha a la espera de recibir un *shortcode*. Cuando esto ocurre se ejecuta el bloque de código a continuación de la función, como parámetros incluye el objeto Chirp recibido, que contendrá el shortcode enviado y, en caso de existir, el objeto JSON ya convertido a una estructura de diccionario.

Además, durante el desarrollo del proyecto, Chirp ha publicado una nueva versión de su SDK (a la que hemos actualizado), incorporando un SDK nuevo en iOS: *"iOS UI Components"*.

Este nuevo SDK no incorpora ninguna funcionalidad adicional a la ya explicada, pero nos permite incluir en nuestro proyecto algunos indicadores del estado de la transmisión del código sonoro.

En nuestro caso, y como veremos en las capturas del prototipo, hemos decidido incluir una barra inferior en la que se puede apreciar la onda sonora captada por el micrófono en tiempo real, parpadeando con un color verde cuando se recibe correctamente un *shortkey.*

Estos elementos de interfaz se deben integrar con el SDK ya comentado anteriormente, a continuación explicaremos brevemente cómo hemos incluido Chirp en nuestro proyecto:

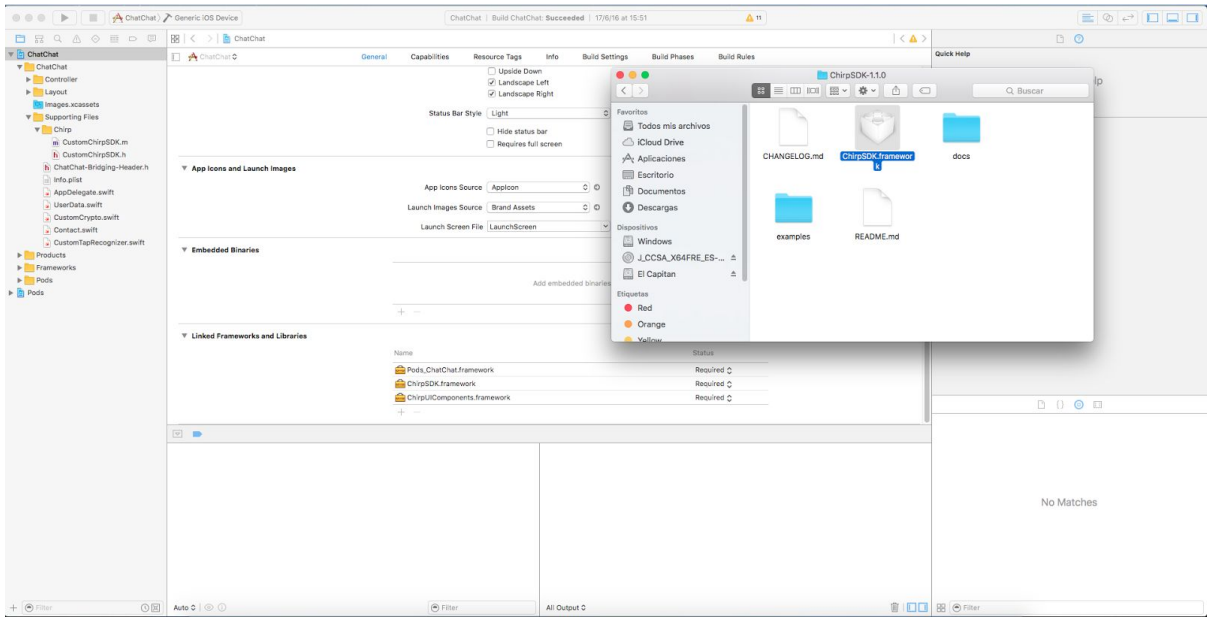


Figura 9: Cómo añadir el framework de Chirp al proyecto.

Para poder integrar el SDK de Chirp en el proyecto arrastramos el archivo *ChirpSDK.framework* al apartado "*Linked Frameworks and Libraries"*

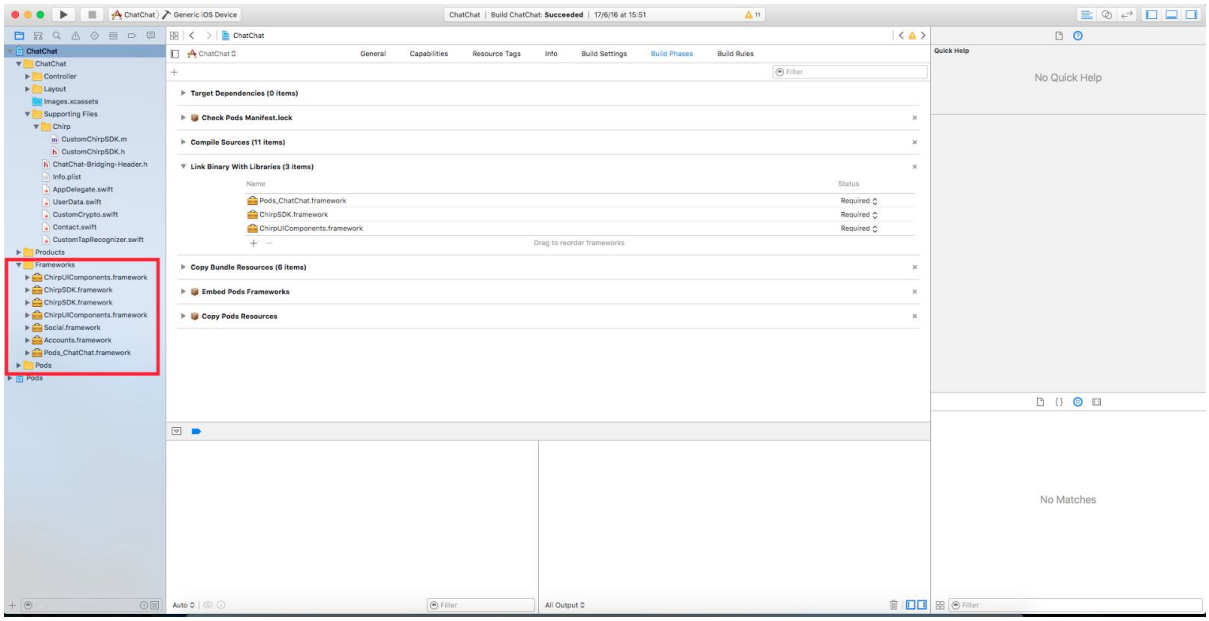


Figura 10: Comprobación del framework de Chirp en el proyecto.

Confirmamos que el *framework* se ha añadido correctamente, debiendo aparecer una copia en la jerarquía del proyecto y referenciado en el apartado "*Link Binary With Libraries"* cómo "*Required".*

Firebase

Firebase [7] es una tecnología relativamente nueva, que, durante el desarrollo del prototipo, ha sido comprada por Google.

Aunque ahora ha integrado muchos servicios con Google podemos definir Firebase en su inicio como una *“Base de datos No-SQL online basada en JSON”.*

En la actualidad incorpora gran cantidad de servicios adicionales en colaboración con Google como Analytics, Autenticación, Almacenamiento de ficheros, Hosting, Monetización de aplicaciones mediante Google Admob..

.

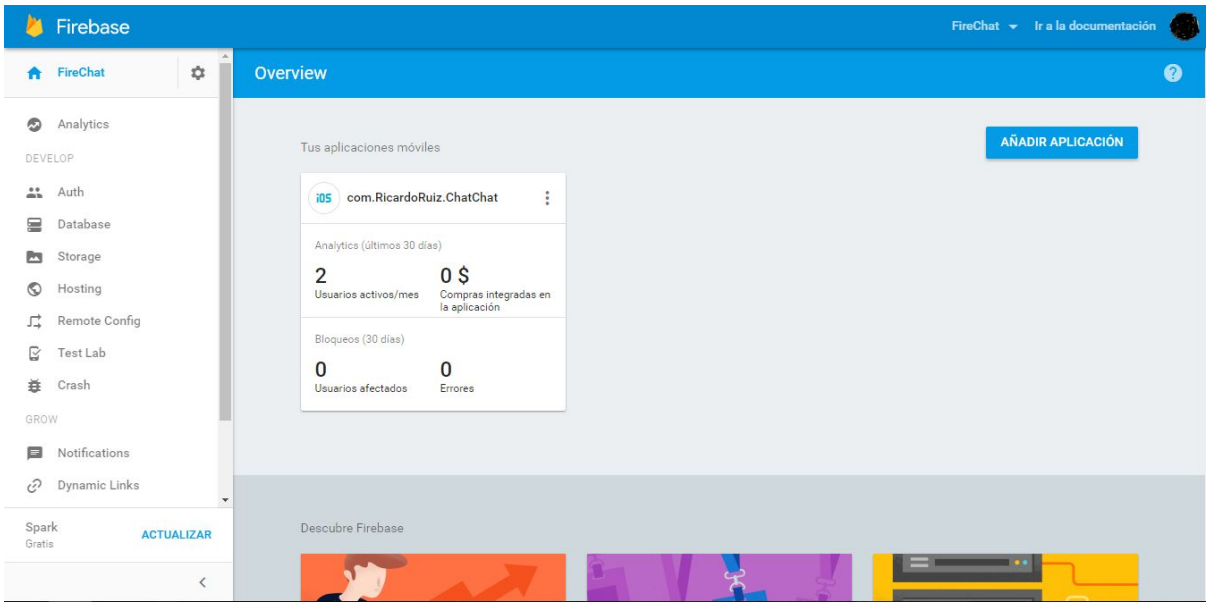


Figura 11: Página principal de Firebase

Para incorporar Firebase en nuestro proyecto hemos hecho uso de los *“CocoaPods” [8]*.

CocoaPods es un gestor de dependencias para Swift y Objective-C, está hecho en *Ruby* e incluye más de 18.000 librerías comúnmente utilizadas en el desarrollo de aplicaciones iOS.

Para utilizar CocoaPods en nuestro proyecto hemos descargado la aplicación *CocoaPods* de su página web, al incluirse Ruby de forma nativa en OS X no necesitamos ninguna instalación ni configuración previa.

Al abrir la aplicación podemos seleccionar un proyecto de la lista de proyectos existentes en XCode.

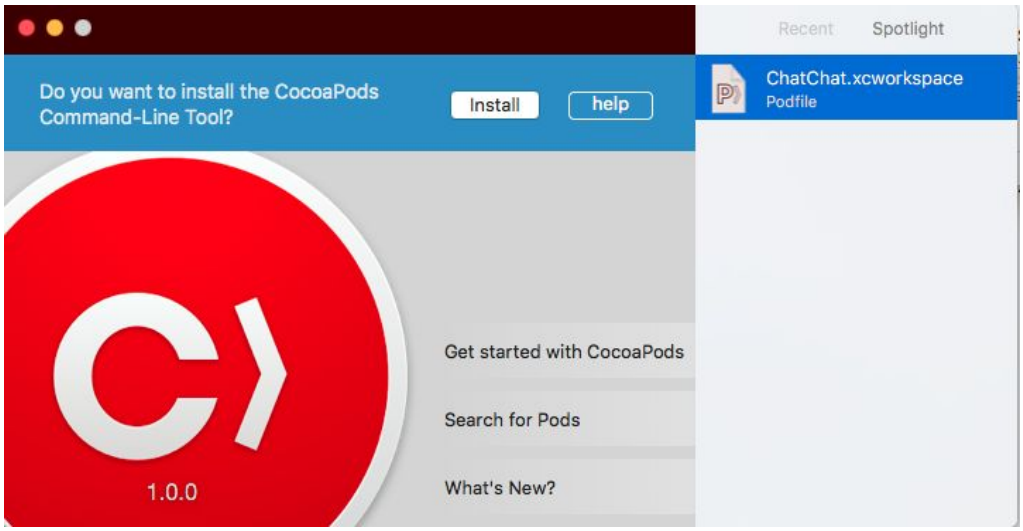


Figura 12: Ventana inicial de CocoaPods

Al hacerlo se creará un archivo *"Podfile"* en el que se pueden incluir las librerías con la siguiente sintaxis

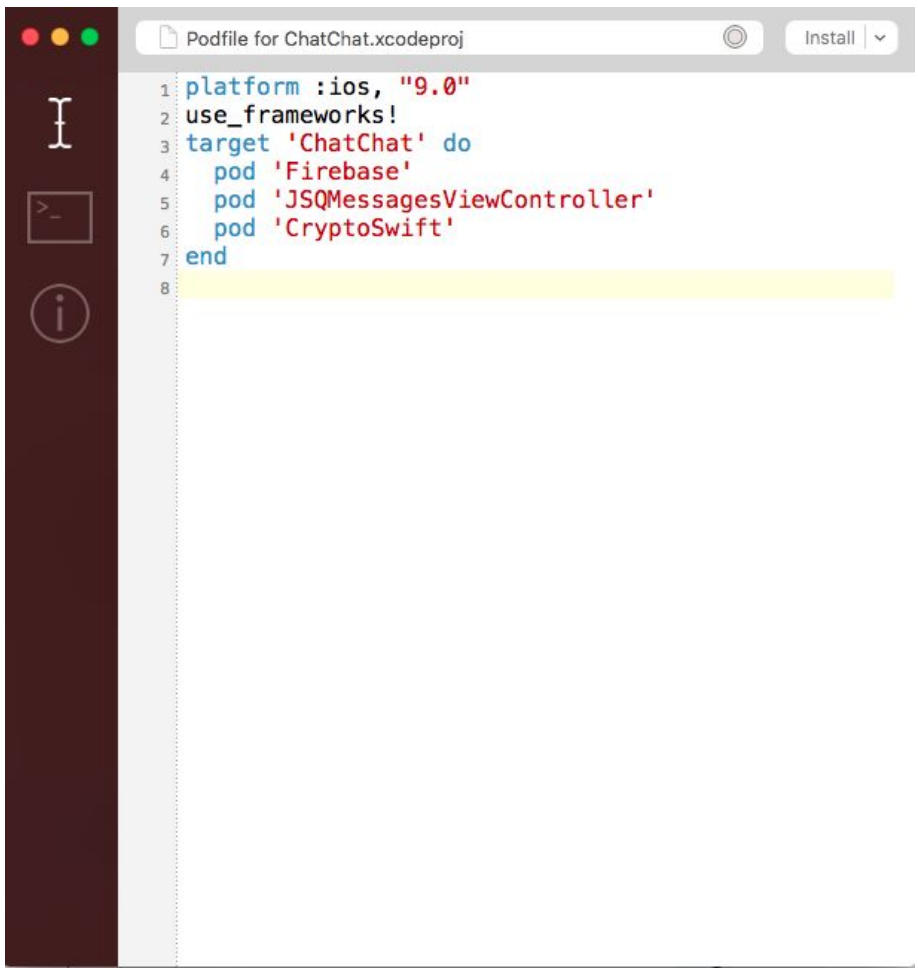


Figura 13: Sintaxis en CocoaPods

En nuestro proyecto no especificamos versión, lo que significa que *CocoaPods* simplemente descargará e incluirá la última versión disponible de las librerías en su repositorio, pero también podríamos haber especificado una versión concreta, una rama de un repositorio git externo a *CocoaPods,* dependencias entre proyectos propios y muchas otras opciones.

En nuestra aplicación únicamente hemos tenido que hacer uso de dos de los servicios de Firebase: Autenticación y Bases de Datos.

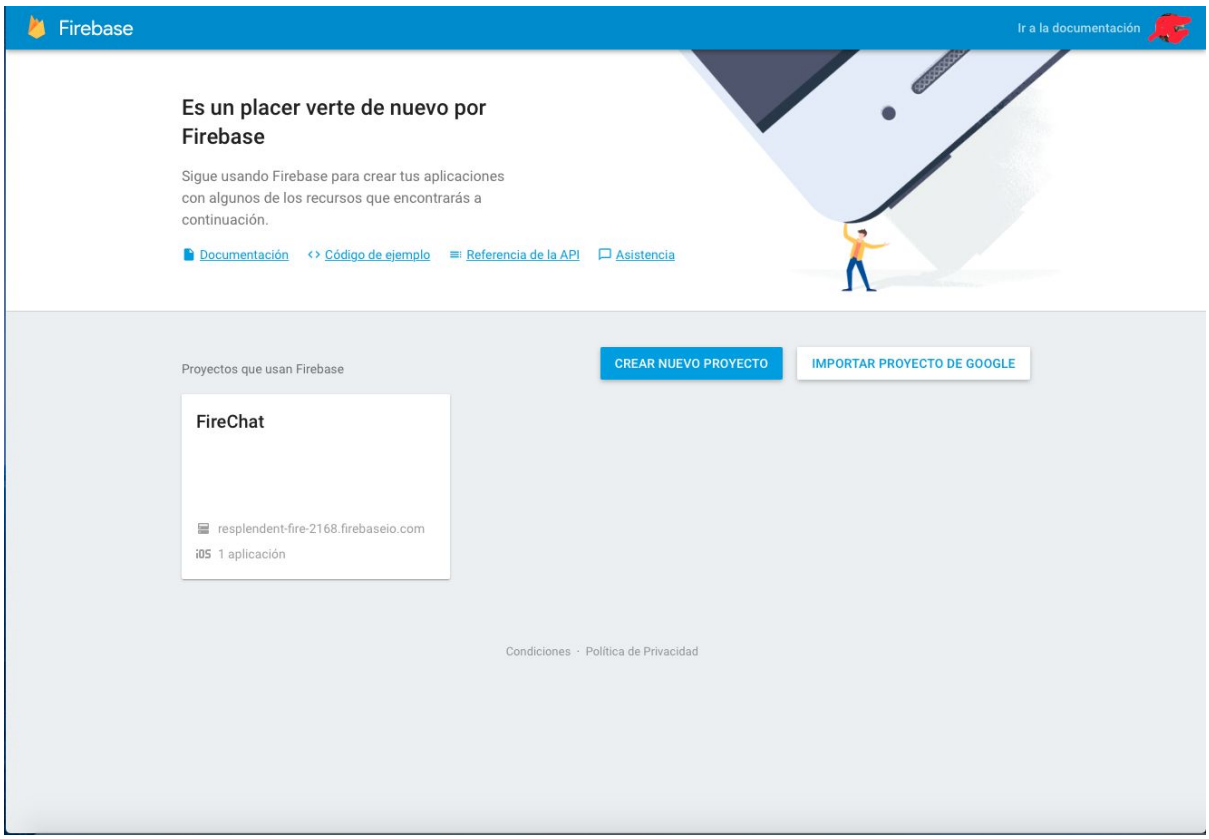


Figura 14: Ventana del proyecto en Firebase

En primer lugar, tras entrar con nuestra cuenta de Google, creamos un proyecto en la web de Firebase que he llamado *"FireChat"*, Firebase nos asigna una URL para poder conectar con sus servicios.

En este caso la URL asignada es: *"resplendent-fire-2168.firebaseio.com"*

Figura 15: Base de Datos con Firebase

Este es un ejemplo de una base de datos con Firebase, podemos observar que la información se estructura en subramas en forma de árbol, con un nodo inicial: nuestro identificador de Firebase.

A continuación se explica cómo hemos utilizado la API de Firebase en nuestro proyecto:

Al entrar en una ventana de chat:

```
// MARK: Properties
var ref = Firebase(url: "https://resplendent-fire-2168.firebaseio.com")

override func viewDidLoad()
{
    super.viewDidLoad()

    ref.authAnonymouslyWithCompletionBlock { error, authData in
        if error != nil {
            print ("Error loging in")
        } else {
            UserData.ID = self.ref.authData.uid
            print ("Authenticated with random ID: " + UserData.ID)
        }
    }
}
```

Figura 16: Código de inicialización de Firebase

Creamos un objeto Firebase que servirá como medio para realizar todas las operaciones con el servidor. La función *viewDidLoad* se llama cuando carga la interfaz de la aplicación, este es el punto en el que nos autenticamos ante el servidor de forma anónima. En caso de error será notificado por consola.

```
override func viewDidLoad() {
  super.viewDidLoad()
  setupBubbles()
  messageRef = rootRef.childByAppendingPath("Messages").childByAppendingPath("General")

  // No avatars
  collectionView!.collectionViewLayout.incomingAvatarViewSize = CGSizeZero
  collectionView!.collectionViewLayout.outgoingAvatarViewSize = CGSizeZero
}
```

Figura 17: Creación de ramas en Firebase

A continuación, para el canal general hemos creado la rama *Messages* (que contendrá los mensajes de los chats) y a su vez la rama *General* (para diferenciarlos de los canales entre usuarios).

```
override func didPressSendButton(button: UIButton!, withMessageText text: String!, senderId: String!, senderDisplayName: String!, date: NSDate!) {

  let itemRef = messageRef.childByAutoId() // 1

  let messageItem = [
      "Sender ID": UserData.ID,
      "Message": text
  ]

  itemRef.setValue(messageItem) // 3

  // 4
  JSQSystemSoundPlayer.jsq_playMessageSentSound()

  // 5
  finishSendingMessage()
  isTyping = false
}
```

Figura 18: Envío de mensajes con Firebase

Para enviar un nuevo mensaje a Firebase necesitamos crear un hijo de un nodo, para eso utilizamos la función *childByAutoId* que crea un hijo del nodo sobre el que se efectúa la operación asignándole un ID aleatorio.

Una vez creado el hijo basta con declarar los campos del mensaje y su contenido en forma de diccionario (String - NSObject) y utilizar la función *setValue* para efectuar el envío.

```
internal func observeMessages() {
  let messagesQuery = messageRef.queryLimitedToLast(25)

  messagesQuery.observeEventType(.ChildAdded) { (snapshot: FDataSnapshot!) in

    let id = snapshot.value["Sender ID"] as! String
    let text = snapshot.value["Message"] as! String

    self.addMessage(id, text: text)

    self.finishReceivingMessage()
  }
}
```

Figura 19: Recepción de mensajes con Firebase

Para leer valores de Firebase utilizamos la función *observeEventType* que hemos incluido dentro de la función *observeMessages*. La función *observeEventType* toma un parámetro un tipo de evento de Firebase, en esto caso, al utilizar *ChildAdded,* el bloque de código será llamado por cada objeto existente o añadido a la rama sobre la que se ejecuta el método. Nótese que hemos limitado la búsqueda a los 25 últimos mensajes.

Para recuperar los valores de la rama disponemos de un objeto *FDataSnapshot* durante el bloque de código, esto es, una “captura” del valor en Firebase (copia por valor). Podemos obtener los valores del objeto diccionario buscando por la clave (String) que le habíamos dado anteriormente.

CryptoSwift

CryptoSwift [9], tal y como su nombre sugiere, es una librería criptográfica en el lenguaje de programación Swift (utilizado conjuntamente con objective-c a lo largo del desarrollo del proyecto iOS).

Se trata de un proyecto de código abierto en Github que incluye las operaciones básicas de criptografía que se utilizan actualmente, entre ellas: Hash (MD5, SHA-1.. 512), CRC, Cifrados (AES-128..256, ChaCha20, Rabbit), HMAC (MD5, SHA-1..256, Poly1305), Modos de bloques (ECB, CBC, CTR…), *Password-Based Key Derivation Function* (PBKDF 1 y 2) y Padding (PKCS#7).

Para incluir CryptoSwift en nuestro proyecto hemos hecho uso de *Cocoapods*, basta con especificar esta nueva librería como un *"pod"* y actualizar las librerías existentes desde la aplicación.

Para las aislar las operaciones criptográficas del control de la aplicación hemos creado una clase llamada "*CustomCrypto"* que contiene los métodos necesarios para realizar todas las operaciones criptográficas con *CryptoSwift* ofreciendo el resultado esperado al control de la aplicación.

A continuación se explica la clase *"CustomCrypto"*

```
static func EncryptUserData (realID : String, username : String, image : NSData) -> String
{
    let key = SubKey()

    let encryptedID = try! realID.encrypt(AES(key: key, iv: iv)).toBase64()
    let encryptedUsername = try! username.encrypt(AES(key: key, iv: iv)).toBase64()
    let encryptedImage = try! (image.encrypt (AES (key: key, iv: iv))).arrayOfBytes().toBase64()

    return encryptedID! + " " + encryptedUsername! + " " + encryptedImage!
}

static func DecryptUserData (encryptedUserData : String) -> (String, String, NSData)
{
    let key = SubKey()

    let userData = encryptedUserData.characters.split{$0 == " "}.map(String.init)

    let ID = try! userData[0].decryptBase64ToString (AES (key: key, iv: iv))
    let username = try! userData[1].decryptBase64ToString (AES (key: key, iv: iv))
    let image = try! NSData (bytes: userData[2].decryptBase64 (AES (key: key, iv: iv)))

    return (ID, username, image)
}
```

Figura 20: Funciones de cifrado y descifrado de información del usuario

Las funciones *EncryptUserData* y *DecryptUserData* cifran y descifran (en base 64 para su posterior almacenamiento) la información del usuario que toma como parámetro. La información de usuario a almacenar es un ID, un nombre y su imagen de perfil.

Nótese que para obtener el bloque cifrado de datos o para descifrarlo los distintos atributos del contacto se separan mediante el carácter espacio.

```
static func EncryptMessage (message : String, channelKey : String) -> String
{
    let encryptedMessage = try! message.encrypt (AES (key: channelKey, iv: iv, blockMode: .CBC, padding: PKCS7 ())).toBase64()

    return encryptedMessage!
}

static func DecryptMessage (encryptedMessage : String, channelKey : String) -> String
{
    let message = try! encryptedMessage.decryptBase64ToString (AES (key: channelKey, iv: iv, blockMode: .CBC, padding: PKCS7 ()))

    return message
}
```

Figura 20: Funciones de cifrado y descifrado de mensajes

Para cifrar y descifrar un mensaje de texto basta con realizar una llamada a la función *encrypt* o *decrypt* de *CryptoSwift* con los parámetros adecuados, el resultado devuelto es el mensaje cifrado o descifrado adecuadamente.

```
static func HMACTimed () -> String
{
    keyPKCS5 = KeyWithPKCS5()

    return try! Authenticator.HMAC(key: Array(CustomChirpSDK.ShortKey().utf8), variant: .sha256).authenticate (keyPKCS5).toHexString()
}
```

Figura 21: Función HMAC

La función *HMACTimed* es una función específica para la generación del código HMAC del protocolo, es sencillamente una función HMAC sobre la clave generada con PBKDF2 que toma como *"salt"* la hora y las redes WiFi.

```
private static func KeyWithPKCS5 () -> [UInt8]
{
    let timeSalt = TimeSalt ().utf8.map {$0}
    let shortkey = CustomChirpSDK.ShortKey().utf8.map {$0}

    let key = try! PKCS5.PBKDF2 (password: shortkey, salt: timeSalt, iterations: 4096, keyLength: shortkey.count, hashVariant: .sha256).calculate()

    return key
}
```

Figura 22: Función de generación de la clave con PBKDF2

Adicionalmente contamos con dos funciones auxiliares utilizadas por la propia clase. La primera de ellas es *KeywithPKCS5* que nos devuelve una clave de mayor tamaño calculada con la función *PBKDF2* utilizando como función hash SHA-256.

```
private static func SubKey () -> String
{
    if (keyPKCS5 == [UInt8.allZeros]) {
        keyPKCS5 = KeyWithPKCS5 ()
    }

    let keyHex = keyPKCS5.toHexString ()

    return keyHex.substringWithRange (keyHex.startIndex..<keyHex.startIndex.advancedBy (32))
}
```

Figura 23: Función de generación de subclave

La segunda, la función SubKey nos devuelve una clave del tamaño adecuado para realizar las operaciones de cifrado con AES.

```
private static func GetRandomString (length: Int) -> String
{
    let letters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".characters
    let lettersLength = UInt32(letters.count)

    let randomCharacters = (0..<length).map { i -> String in
        let offset = Int(arc4random_uniform(lettersLength))
        let c = letters[letters.startIndex.advancedBy(offset)]
        return String(c)
    }

    return randomCharacters.joinWithSeparator("")
}
```

Figura 24: Función de generación de cadena alfanumérica aleatoria

Por último la función GetRandomString nos devuelve una cadena de caracteres alfanumérica aleatoria de la longitud especificada como parámetro.

JSQMessages

JSQMessages [10] es una librería de gráficos y control que nos permite crear de forma sencilla los elementos básicos y con un diseño estandarizado de una ventana de chat en iOS.

Entre los elementos se incluyen diferentes burbujas con mensajes: texto, contenedoras de videos o imagenes, localización geográfica. Todo a nivel de interfaz de usuario.

Dado que se trata de una librería compleja y la elaboración de un chat es sólo una excusa para poner de manifiesto la solución teórica propuesta hemos hecho un uso reducido de esta librería limitándonos a la creación de ventanas de chat, burbujas de mensajes e indicadores de *"escribiendo…".*

A continuación explicaremos el uso que hemos dado de la librería JSQMessages para la elaboración de nuestro prototipo:

```
class ChatViewController: JSQMessagesViewController {

  // MARK: Properties
  let rootRef = Firebase(url: "https://resplendent-fire-2168.firebaseio.com/")
  var messageRef: Firebase!
  var messages = [JSQMessage]()
```

Figura 25: Subclase de JSQMessagesViewController

En primer lugar la clase de ventana de chat que vaya a hacer uso de la librería debe heredar de la clase *JSQMessagesViewController*

```
override func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
  let cell = super.collectionView(collectionView, cellForItemAtIndexPath: indexPath) as! JSQMessagesCollectionViewCell

  let message = messages[indexPath.item]

  if message.senderId == UserData.ID { // 1
    cell.textView!.textColor = UIColor.blueColor() // 2
  } else {
    cell.textView!.textColor = UIColor.whiteColor() // 3
  }

  return cell
}
```

Figura 25: Sobreescribiendo la función collectionView

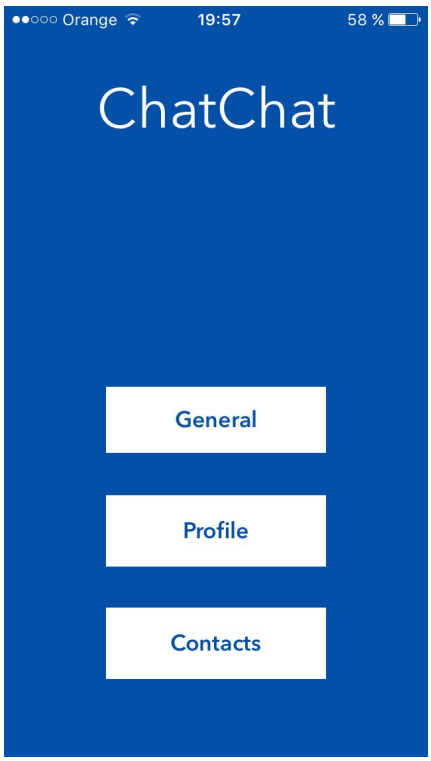
A continuación necesitamos sobreescribir el método *collectionView* de la clase padre para que cuando se detecte un nuevo mensaje le sea asignada una burbuja de chat, en color azul si el ID del remitente del mensaje coincide con el nuestro y en color blanco para el caso contrario (es un mensaje de otro usuario).

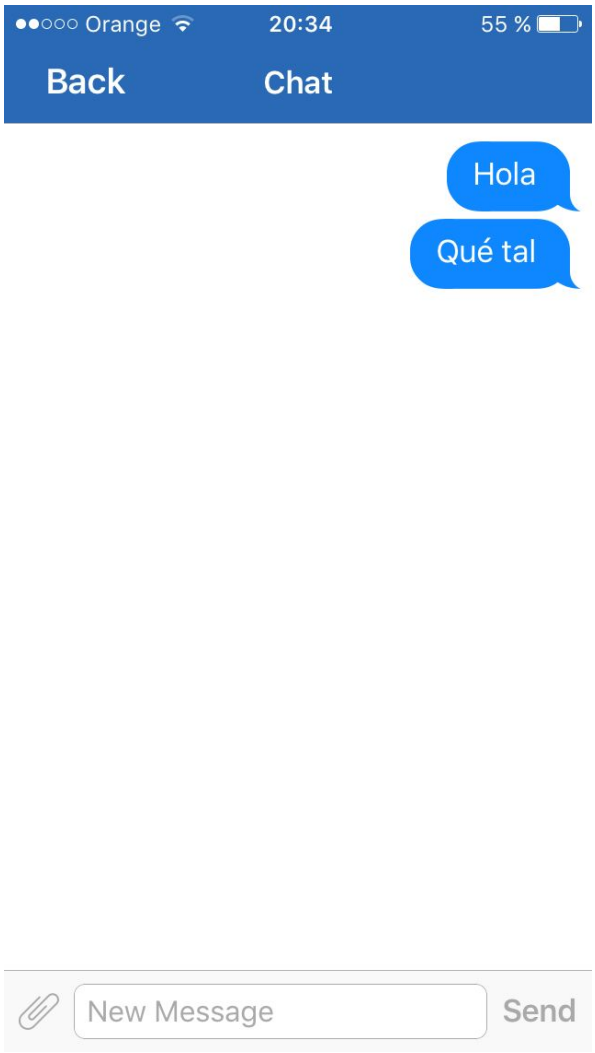
5. Prototipo: ChatChat

Como aplicación prototipo para la utilización del canal sonoro con Chirp e implementación del protocolo propuesto hemos elaborado una aplicación de chat para dispositivos iOS, que hemos denominado ChatChat.

A continuación explicaremos el funcionamiento completo de la aplicación y la sincronización de dos dispositivos mediante capturas:

En esta primera captura se observa la ventana principal de la aplicación, en la que podemos ver el nombre de la misma y tres botones.

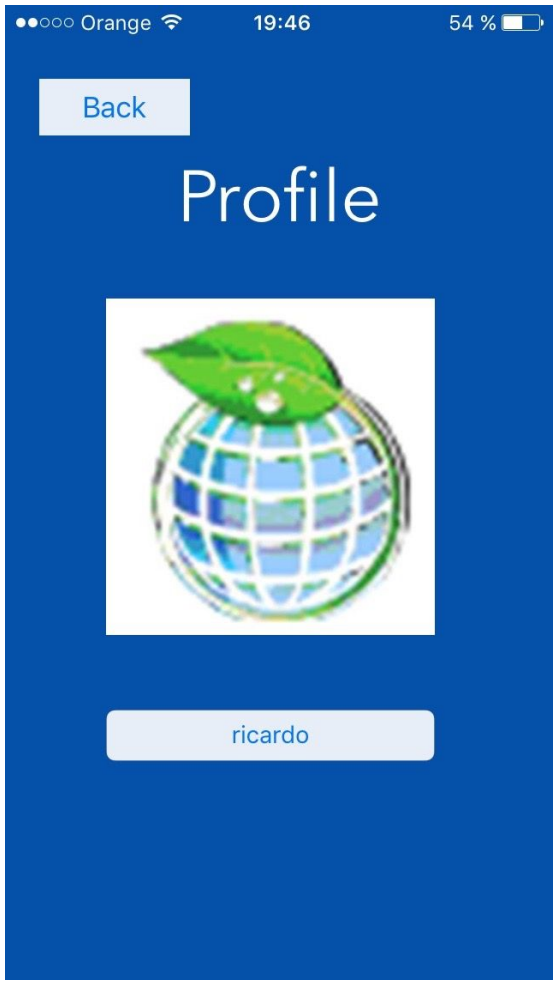




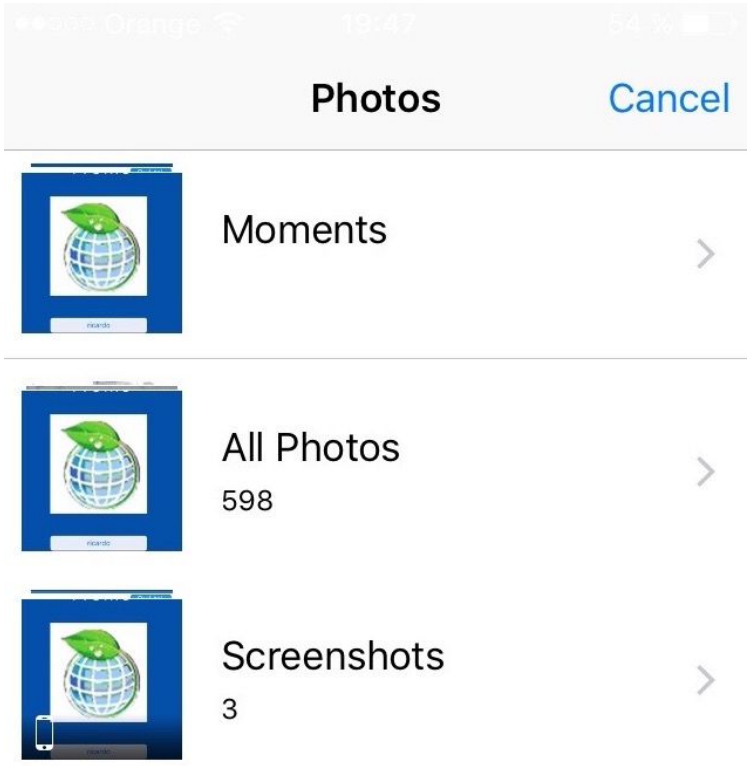
Si pulsamos sobre el botón *"General"* nos llevará a una ventana de chat general. En la esquina izquierda superior tenemos un botón de *"Back"* con el que podemos volver a la vista principal.

La ventana de chat general muestra en burbujas los mensajes enviados al "Chat General" de Firebase, autenticados como un usuario anónimo podemos enviar nuevos mensajes al chat que se almacenarán en Firebase en claro y serán visibles para el resto de usuarios de la sala.

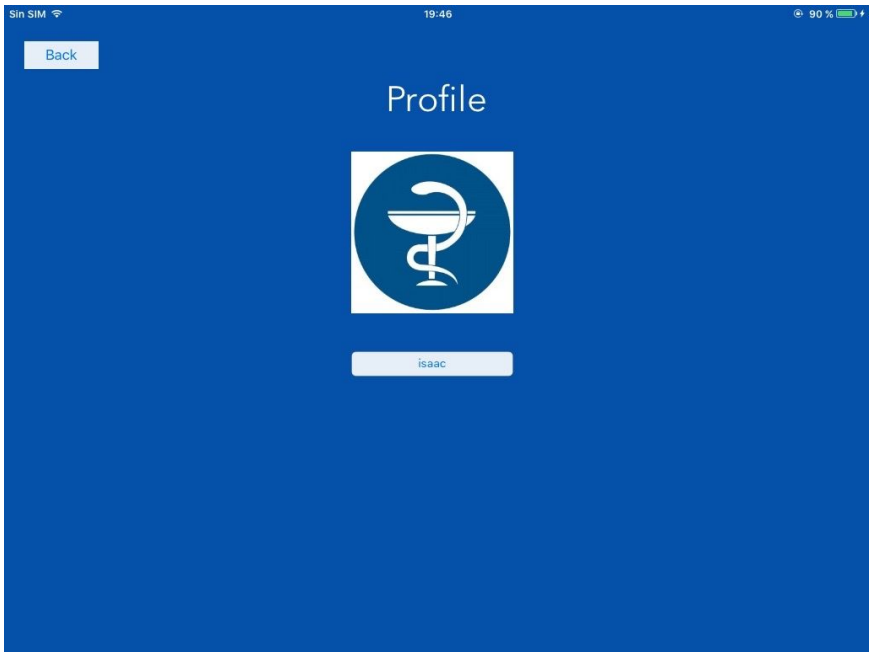
Nótese que si salimos y volvemos a entrar en el chat general Firebase nos asigna un nuevo ID aleatorio, por lo que no hay forma de identificar los mensajes pertenecientes a un mismo dispositivo.



En la ventana de perfil podemos editar la información de usuario: nuestro nombre y nuestra imagen. Esta información se almacena en un fichero local y es persistente aunque cerremos la aplicación.

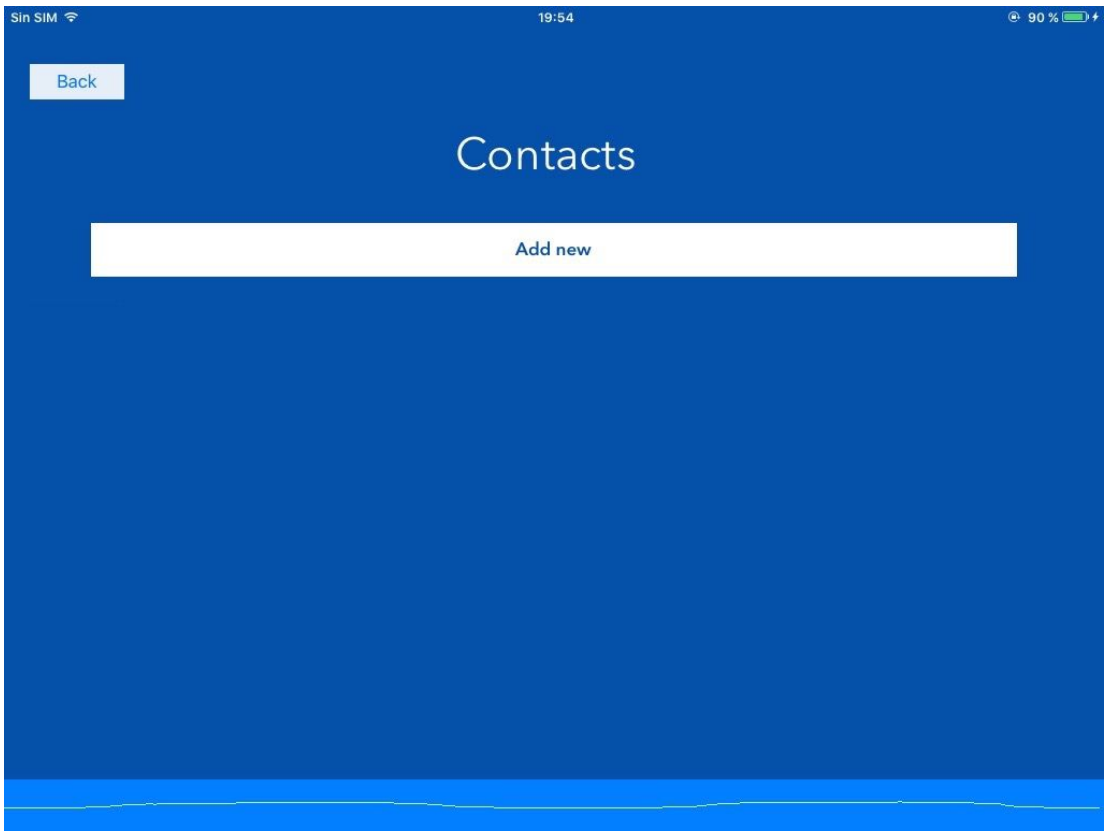


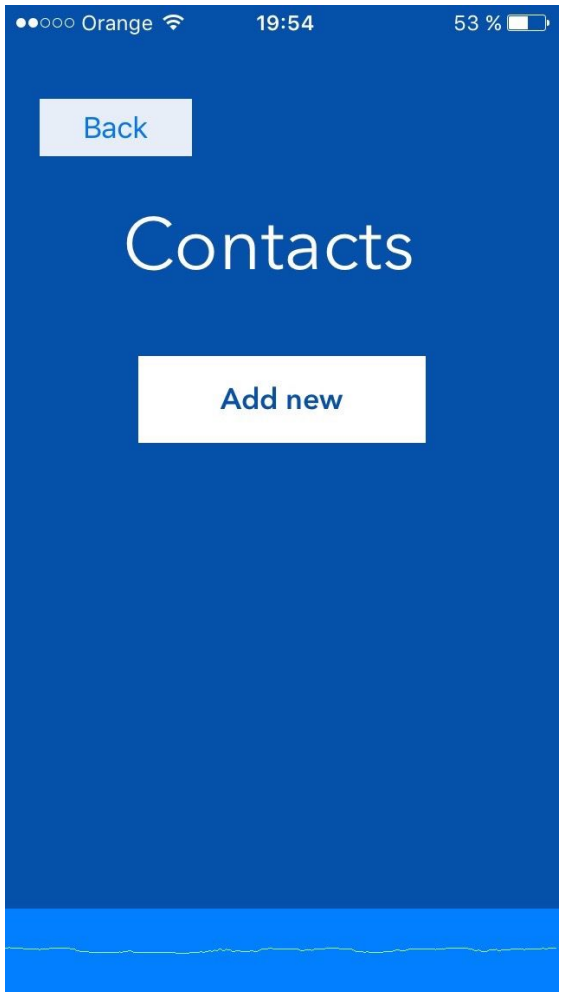
Si tocamos sobre la foto podemos seleccionar una imagen de la galería como nuestra foto de perfil.



Esta es la información del otro usuario con el que realizaremos la sincronización.

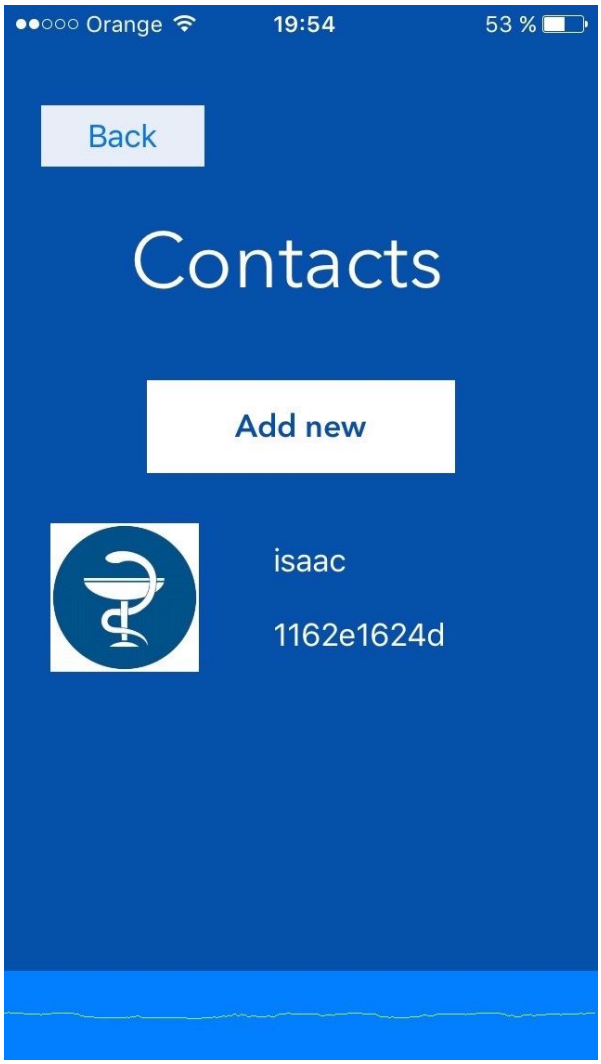
Si abrimos la ventana de contactos en ambos dispositivos:

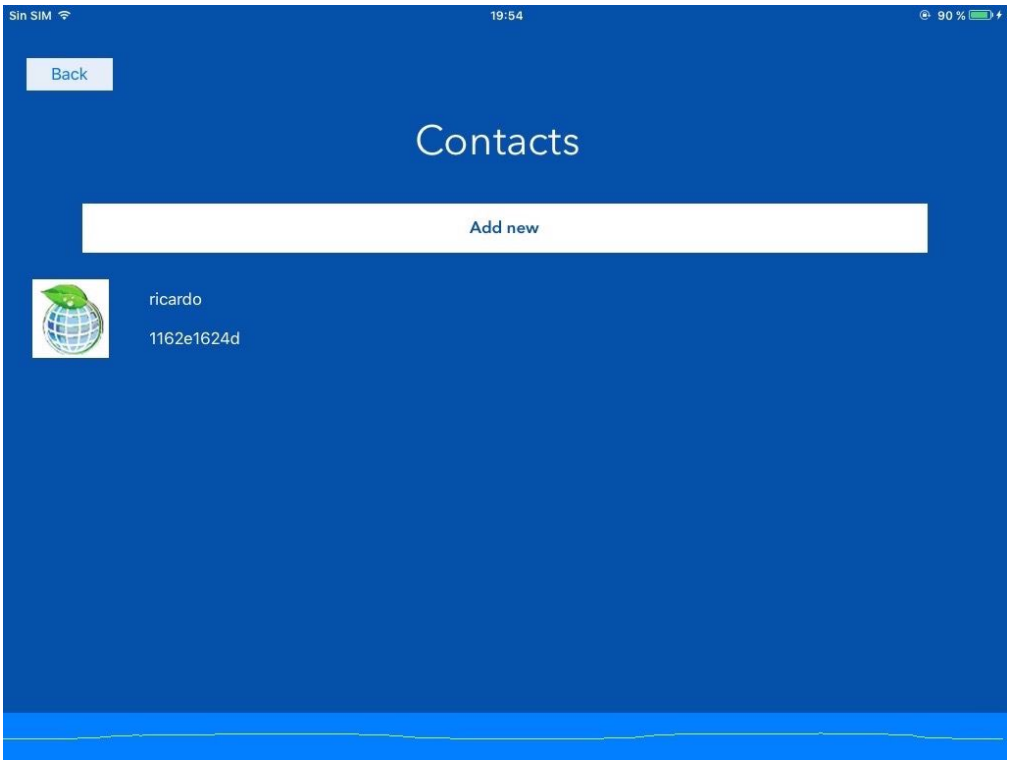




Podemos observar que no hay ningún contacto añadido, en la parte inferior tenemos la onda sonora en tiempo real que incluimos con la librería *“Chirp UI Components”.*

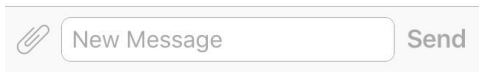
Si tocamos ahora el botón *“Add new”* comenzará el protocolo, enviándose la clave corta a través del canal sonoro, generando la clave real mediante PBKDF2 y enviando la información de contacto cifrada y el HMAC mediante Firebase entre los dispositivos.

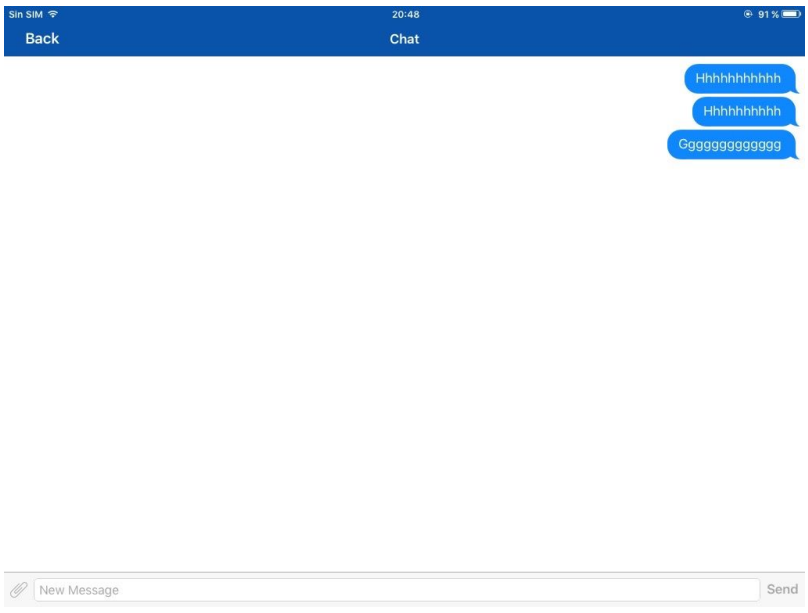

●●○○○ Orange
19:54
53 %
Back
Contacts
Add new
isaac
1162e1624d



El resultado de la comunicación es que en ambos dispositivos se ha agregado mutuamente como contacto. El número inferior al nombre de usuario es la ID del canal (constituida a partir de la ID de ambos contactos en la sincronización) que será la rama en *Messages* en la que ambos realizarán la comunicación.

Si tocamos la imagen del contacto agregado podemos acceder a tener una conversación con él en una ventana de chat similar a la del canal general.





Vemos que los mensajes se envían y reciben como si fueran en claro, pero si observamos Firebase podremos apreciar el siguiente esquema.

Distinguimos en nuestro esquema de Firebase dos grandes ramas:

- Contact Sync: Es la rama en la que se envían los mensajes de sincronización. Tiene dos subramas: Receiver y Sender, en función de quién haya iniciado la sincronización con Chirp y quien haya recibido el mensaje sonoro. Se envía la información de usuario cifrada, el HMAC y el ID aleatorio

- Messages: Es la rama dedicada al envío de mensajes, distinguimos una rama General en la que podemos observar que se envían los mensajes en claro (texto *"Hola"* y *"Que tal"*) y una rama con un ID de canal coincidente con el ID que aparecía debajo del nombre del contacto agregado constituido por el ID aleatorio que ambos tenían. En esta subrama (canal de contactos sincronizados) podemos observar que los mensajes se envían efectivamente cifrados.

El objetivo teórico que propusimos se ve por tanto cumplido. Por el servidor de Firebase únicamente pasan dos tipos de información: IDs aleatorios cada vez que establecemos un canal de comunicación (que garantizan la privacidad de los usuarios al no poder identificarse con los mensajes enviados) y mensajes cifrados (en claro únicamente en el canal general).

6. Conclusiones y mejoras sobre el proyecto

Podemos concluir sobre este proyecto que los resultados teóricos que pretendíamos alcanzar son viables con la tecnología actual.

Aunque la tecnología para el envío de información utilizando el canal sonoro aún está en vías de desarrollo y, como vemos, en Chirp tiene una extensión máxima de 10 caracteres es suficiente para poder generar una clave de mayor tamaño, pudiendo ser utilizado como protocolo de intercambio de claves y autenticación de los dispositivos.

Otras tecnologías utilizadas están también ahora en su auge: Firebase ha sido recientemente comprada por Google durante el desarrollo del proyecto ampliando los servicios que ofrece y extendiendo su uso a tecnologías móviles, CryptoSwift se ha actualizado para incorporar las mejoras que Swift 3 ofrece…

En definitiva, las tecnologías utilizadas en este proyecto están en crecimiento continuo y el prototipo que proponemos puede ser, salvando las distancias, una de las infraestructuras en el desarrollo de aplicaciones del futuro.

En cuanto a las posibles mejoras sobre el prototipo elaborado es cierto que nuestro protocolo no incluye "*Perfect Forward Secrecy (PFS)".* Si un atacante consiguiera hacerse con la clave privada de un canal podría descifrar todos los mensajes pasados de ese canal.

Como solución podemos proponer la renovación de la clave privada con el servidor cada cierto tiempo, siendo la primera sincronización de los dispositivos a través del canal sonoro y únicamente utilizando el servidor para renovarla. Sin embargo, esta solución nos obligaría a tener un mayor control sobre la parte servidor, por lo que deberíamos utilizar un servidor web propio, y no Firebase.

7. Referencias bibliográficas

[1] Tactile One-Time Pad: Leakage-Resilient Authentication for Smartphones
https://www.syssec.rub.de/media/emma/veroeffentlichungen/2015/02/02/vibLogin.pdf

[2] Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound
http://arxiv.org/pdf/1503.03790.pdf

[3] Signal360 (Anteriormente SonicNotify)
http://www.gizmag.com/sonicnotify-audio-signals/21385

[4] Audio Modem. Data over sound.
https://applidium.com/en/news/data_transfer_through_sound/

[5] Chirp
http://chirp.io

[6] PBKDF2 (Password-Based Key Derivation Function 2) - RFC
https://tools.ietf.org/html/rfc2898#section-5.2

[7] Firebase
https://firebase.google.com/

[8] CocoaPods
https://cocoapods.org/

[9] CryptoSwift
https://github.com/krzyzanowskim/CryptoSwift

[10] JSQMessages
https://github.com/jessesquires/JSQMessagesViewController/tree/master

[11] Función cercano utilizada por Google en Android

https://support.google.com/accounts/answer/6260286?p=google_settings_nearby&rd=1

[12] The Swift Programming Language (Swift 3 beta). iBook version.

https://itunes.apple.com/es/book/swift-programming-language/id1002622538?mt=11

[13] Using Swift with Cocoa and Objective-C (Swift 2.2). iBook version.

https://itunes.apple.com/es/book/using-swift-cocoa-objective/id888894773?mt=11