TRABAJO DE FIN DE MASTER

UNIVERSIDAD REY JUAN CARLOS

ESCUELA TECNICA SUPERIOR DE INGENIERÍA INFORMATICA

MASTER EN CIBERSEGURIDAD Y PRIVACIDAD

DISEÑO E IMPLEMENTACIÓN DE UN ESQUEMA DE INTERCAMBIO DE CLAVE PARA N-USUARIOS

Autor:
DANIEL DE LAS HERAS MONTERO

Supervisor:
MARÍA ISABEL GONZÁLEZ VASCO

Índice

1 Abstract/Resumen

El siguiente texto tiene por objetivo detallar el proyecto realizado para el Trabajo Fin de Máster de Ciberseguridad y Privacidad de la Universidad Rey Juan Carlos.

El objetivo del proyecto es la implementación del protocolo de generación de una clave e id de sesión compartido por un grupo. Dicho esquema se basa en la definición realizada en la publicación '*Authenticated Key Establishment: From 2-Party To Group'* [1].

La codificación del esquema, así como la documentación utilizada, está almacenado y publicado en un repositorio público de GitHub (https://github.com/danydlhm/2PartytoGroupCompiler).

El código resultante de este proyecto debe tener la capacidad de simular la ejecución del protocolo anteriormente mencionado en una ejecución de principio a fin. Adicional a la implementación, se analizará como el algoritmo escala en tiempo con respecto al número de participantes.

Este trabajo se divide en cinco partes:

- *Preliminares*
  - *Algoritmo generador de compromisos*, donde se enuncia la definición de esquema de generación de compromisos.
  - *Authenticated Key Establishment: From 2-Party to Group*, donde se explica el protocolo y los cálculos que se realizan en las rondas definidas por el protocolo.
- *Metodología de desarrollo.* Sección en el cual se expone la forma en la que se ha desarrollado tanto la implementación como la adquisición de los conocimientos previos necesarios para la realización de este. .
- *Implementación*. Bloque central del trabajo que desarrolla las elecciones realizadas a la hora de diseñar e implementar el protocolo.
- *Resultados*. Sección donde se exponen los resultados obtenidos al estudiar la escalabilidad con respecto al tiempo del protocolo.
- *Conclusiones*. Apartado donde se presentan las conclusiones del trabajo realizado.

2 Preliminares

2.1 Algoritmo de Intercambio de Claves (AKE)

Un protocolo AKE 2-parte, es un esquema criptográfico de intercambio de claves autenticado (Authentication Key Exchange) que permite que dos usuarios generen un secreto común, de tal manera que la comunicación entre ambos sea segura y quede autenticada.

Este tipo de protocolos es una herramienta fundamental debido a que permite establecer una comunicación segura entre 2 usuarios sobre un canal abierto.

2.2 Algoritmo generador de compromisos

Informalmente, un esquema de generación de compromisos (Commitment Scheme) se puede visualizar como una "caja cerrada", donde el contenido de la caja está oculto (sin la clave) y que solo se puede abrir de una forma.

Un algoritmo de generación de compromisos se puede definir más formalmente como un protocolo eficiente de dos fases: Commit y Reveal. En la fase de Commit, el protocolo genera un compromiso donde a partir de un input m se genera una salida c y d. Donde c es el compromiso

y d es el valor para abrir/validar ese compromiso, y el emisor del compromiso manda la salida c. En la fase de Reveal, el emisor envía la salida d de la anterior fase y el valor de entrada m con las que el receptor valida si acepta o no el compromiso mandado en la fase anterior.

2.3 Compilador de Abdalla et al

A continuación, se detalla el protocolo de establecimiento de secreto compartido para un grupo de participantes. Previo a entrar en detalle de las rondas del algoritmo, es necesario especificar que cada integrante tiene un identificador de usuario (UID) y que todos los participantes conocen todos los identificadores de los usuarios participantes en las rondas. Con dichos identificadores se puede establecer un orden dentro de los usuarios de tal manera que existe una arquitectura ordenada cíclica de participantes, donde el primer usuario tiene como participante previo el último de la lista. Dicha disposición de los usuarios se visualiza en la Ilustración 1 a continuación.

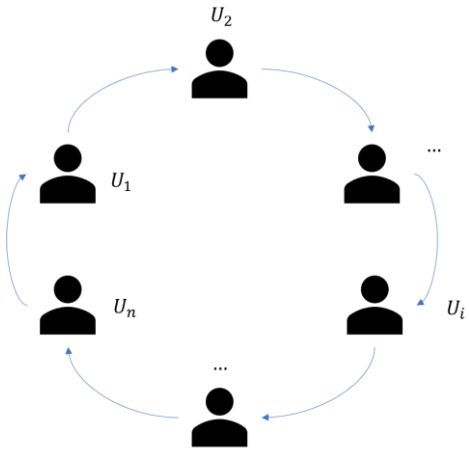

Ilustración 1 Arquitectura de los participantes

Round 0:

2-AKE: For $i = 1, \ldots, n$ execute 2-AKE(U_i, U_{i+1}).[a] Thus, each user U_i holds two keys $\overrightarrow{K}_i$, $\overleftarrow{K}_i$ shared with U_{i+1} respectively U_{i-1}.

Round 1:

Computation: Each U_i computes

$$X_i := \overrightarrow{K}_i \oplus \overleftarrow{K}_i$$

and chooses a random r_i to compute a commitment $C_i = C_\rho(i, X_i; r_i)$.

Broadcast: Each U_i broadcasts $M_i^1 := (U_i, C_i)$

Round 2:

Broadcast: Each U_i broadcasts $M_i^2 := (U_i, X_i, r_i)$

Check: Each U_i checks that $X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0$ and the correctness of the commitments. If at least one of these checks fails, set $\mathsf{acc}_i :=$ false and terminate the protocol execution.

Computation: Each U_i sets $K_i := \overleftarrow{K}_i$ and computes the $n-1$ values

$$K_{i-j} := \overleftarrow{K}_i \oplus X_{i-1} \oplus \cdots \oplus X_{i-j} \quad (j = 1, \ldots, n-1),$$

defines a master key

$$K := (K_1, \ldots, K_n, \mathsf{pid}_i),$$

and sets $\mathsf{sk}_i := F_{\mathrm{UH}(K)}(v_1)$, $\mathsf{sid}_i := F_{\mathrm{UH}(K)}(v_0)$ and $\mathsf{acc}_i :=$ true.

[a] All indices are to be taken in a cycle, i. e., $U_{n+1} = U_1$, etc.

Ilustración 2 Compilador de Abdalla et al. [1]

A continuación, se explica cada una de las rondas del protocolo que se introduce en la Ilustración 2.

En la Ronda 0, cada participante utiliza un protocolo 2-AKE para establecer una clave con su participante anterior y posterior. Es decir, cada usuario U_i, posee dos claves con sus participantes contiguos. $\overleftarrow{K}$ para el anterior y $\overrightarrow{K}$ para el posterior.

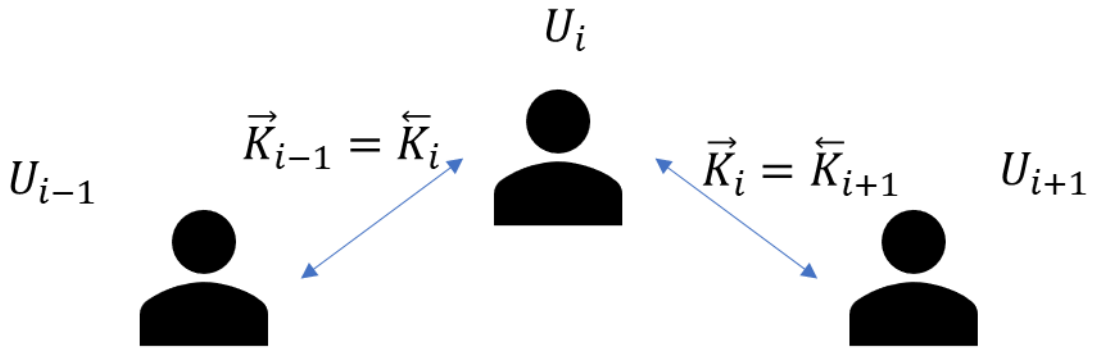

Ilustración 3 Ronda 0

Una vez finalizada la ronda anterior, cada participante calcula un valor X_i a partir del operador XOR bit a bit de las dos claves que almacena. A continuación, se genera un numero aleatorio r_i,

y a partir de los dos valores que se acaban de generar y el valor i (orden en la lista), se genera el compromiso del usuario i. Por último, en esta ronda cada usuario emite un mensaje compuesto por su identificación de usuario y su compromiso en abierto.

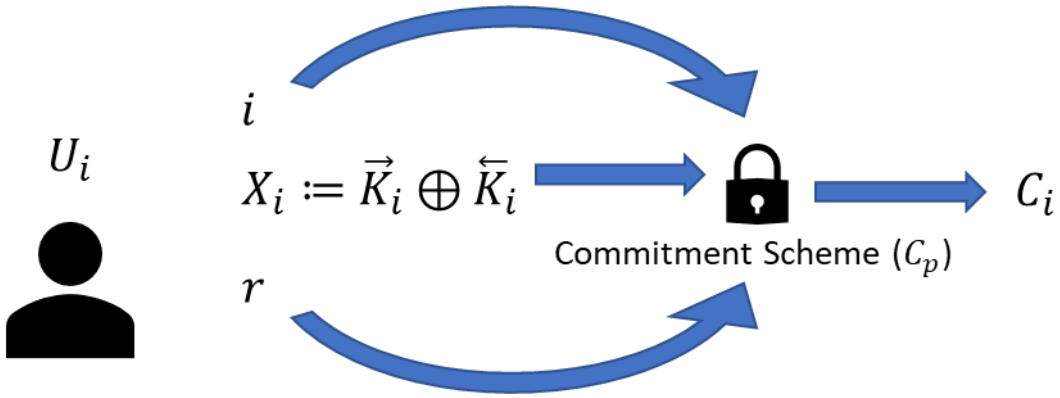

Ilustración 4 Cálculos de la ronda 1

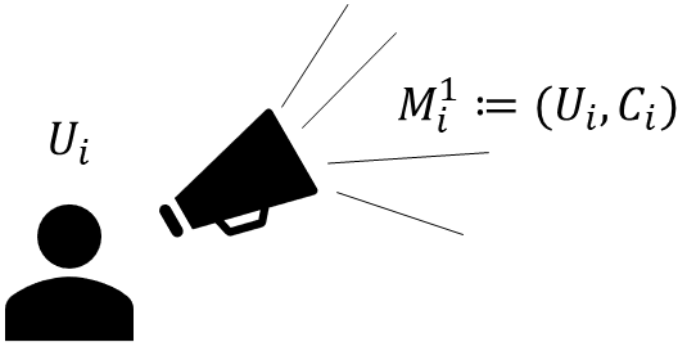

Ilustración 5 Emisión del mensaje de la ronda 1

En la última ronda, cada usuario emite un mensaje compuesto del identificador de usuario el valor X_i y el numero aleatorio r_i con el que se calculó el compromiso en la ronda anterior. A partir de los mensajes de todos los usuarios, cada participante hace la comprobación que la operación XOR de todos los valores X_i es 0 y que los compromisos se han calculado con los valores X y r que se han emitido en la segunda ronda. Si alguna de las comprobaciones anteriores resulta falsa, el protocolo se para y devuelve un valor Falso como variable acc. En caso de continuar, cada participante calcula los valores K_j utilizando la operación XOR utilizando el valor K_i y las variables X_k $\forall k \in \{i, i+1, \dots, j\}$. Por último, se calcula una clave de sesión (sk_i) y un id de sesión (sid_i) a partir de dos funciones seleccionadas de una familia de funciones $Hash$ y de las claves K calculadas anteriormente.

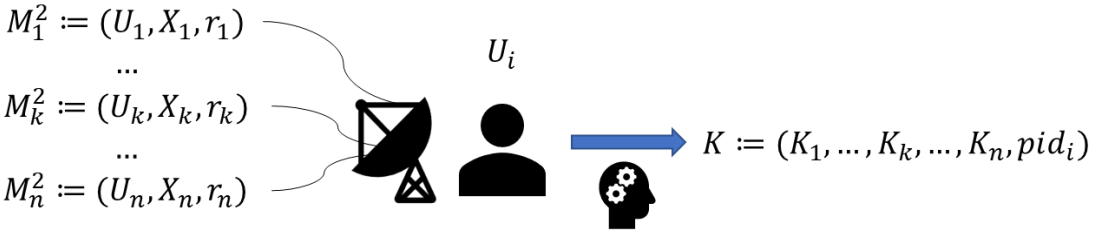

Ilustración 6 Ronda 2

3 Metodología de desarrollo

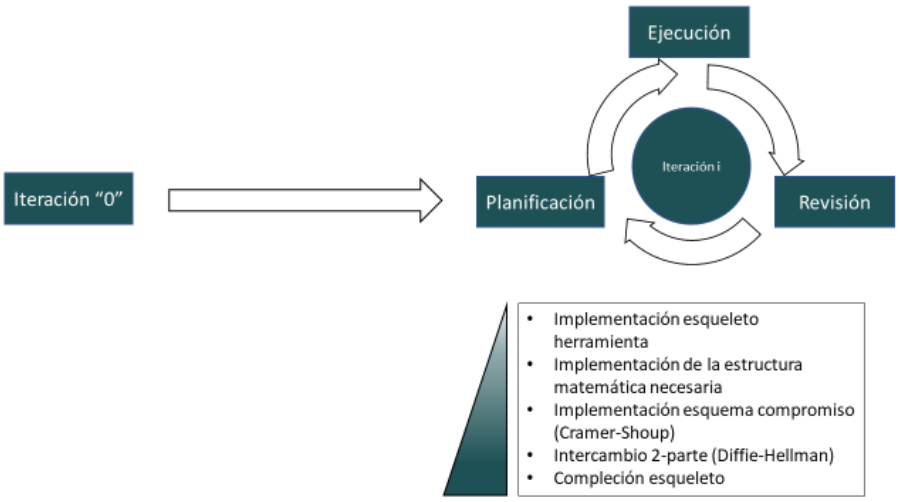

Ilustración 7 Esquema de la metodología de trabajo realizada

Para el desarrollo del trabajo se ha seguido de una metodología por iteraciones de dos semanas. Donde cabe destacar una primera iteración o "0" donde se obtuvieron los conocimientos teóricos necesarios para implementar el protocolo descrito en [1].

En la primera iteración, para el correcto entendimiento del protocolo, adicionalmente a la publicación [1], se realizó la lectura Cramer Shoup [2] y [3], herramienta necesaria para la construcción del protocolo.

El resto de iteraciones, se componían de 3 fases. Primero se planificaba las tareas a llevar a cabo durante la iteración, posteriormente se ejecutaban estas y por último se revisaban y planificaba la siguiente iteración.

Debido a que el trabajo era sobre todo implementación, se realizó una implementación del protocolo principal en base a funciones no implementadas (implementación esqueleto herramienta) y a partir de esa estructura se fue iterando construyendo el resto de herramientas necesarias para su completo funcionamiento.

4 Implementación

Antes de entrar en detalle con la implementación, es necesario especificar una serie de consideraciones tomadas para la implementación.

Para la implementación del protocolo se ha decidido utilizar el lenguaje Python debido a conocimiento previo del lenguaje, y a las herramientas y librerías de dicho lenguaje que ha facilitado la implementación del protocolo.

Se ha tomado como hipótesis, que anterior al inicio de la ronda 1 del protocolo, no hay ningún adversario en el canal de comunicación entre los participantes.

Con respecto a la familia de Hash de la ronda 2, se ha elegido los Hash sha3_512 y sha_3_224 para el cálculo de la clave de sesión y del id de sesión respectivamente.

Los parámetros para la generación de las claves publicas AKE y de los compromisos son compartidas por todos, por lo que solo uno de los participantes genera dichos parámetros y los comparte con el resto.

A continuación, se detalla la estructura de clases de la solución implementada.

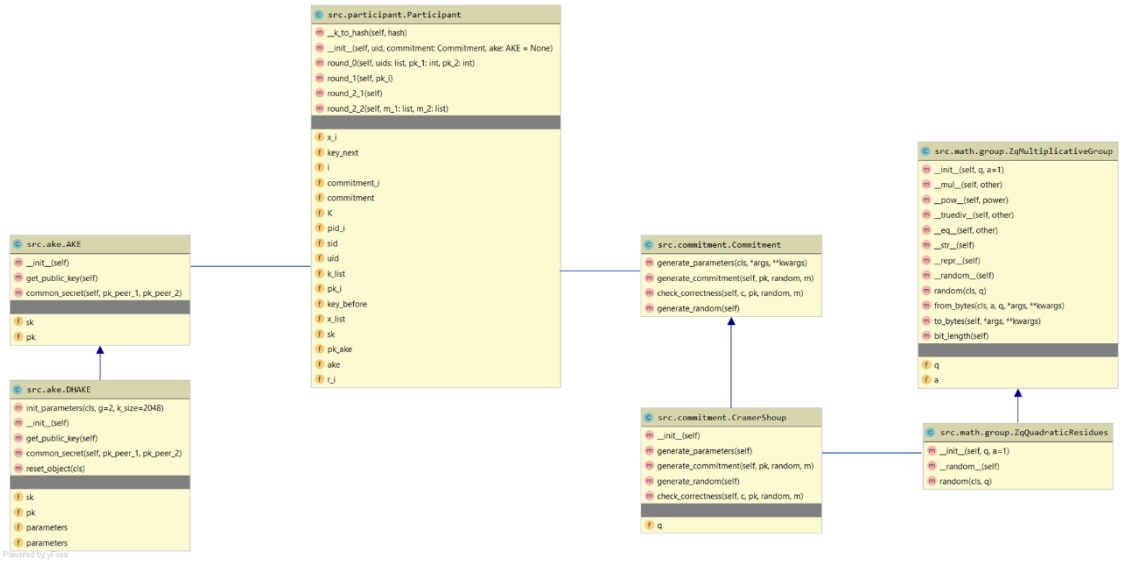

Ilustración 8 Diagrama de clases del código implementado

Como se puede visualizar en la ilustración 3, para el proyecto se ha diseñado de manera que tanto la implementación del AKE como del generador de compromisos (Commitment) sean fácilmente intercambiable con otros algoritmos equivalentes. Debido a esto, se ha implementado las clases AKE y Commitment como dos clases con métodos abstractos y se deja la implementación de dichos métodos a una subclase que implemente un algoritmo especifico.

Adicional a la consideración anterior, en la ilustración 3 se muestra que la clase principal del código es Participant, que es la que implementa los pasos del protocolo mediante diferentes métodos que son llamados en la función main del proyecto, quien a su vez gestiona si la simulación del protocolo es monoproceso o multiproceso. Además, dicha la clase Participant instancia las clases DHAKE y CramerShoup que son las dos clases que implementan los algoritmos de AKE y de generación de compromisos respectivamente.

A continuación, se detallará las clases anteriormente mencionadas.

4.1 Implementación AKE

Como se ha dicho anteriormente, la clase AKE tiene todos sus métodos abstractos de tal manera que hace de interfaz que deben tener las subclases que implementen las distintas herramientas a usar en el protocolo.

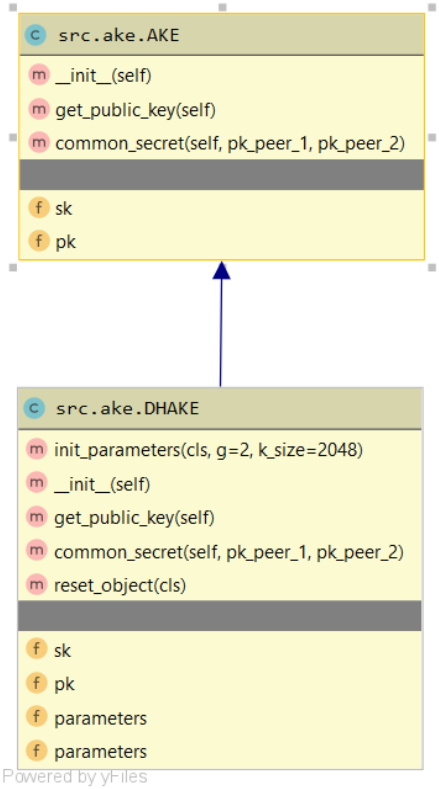

Ilustración 9 Detalle de la clase AKE y subclase

Dicha clase solo tiene dos métodos además del método de clase "__init__". El método "get_public_key" debe generar una clave publica que será compartida con los otros usuarios en la ronda 0 del protocolo. El otro método abstracto, "common_secret", es el encargado de generar a partir de las dos claves públicas de los usuarios adyacentes, las claves $\overleftarrow{K}$ para el anterior y $\vec{K}$ para el posterior.

Para la simulación, se ha desarrollado la clase DHAKE que implementa los métodos abstractos anteriores basándose en el cifrado Diffie-Hellman implementado en la librería de Python cryptography [4]. Debido a que los parámetros del AKE son compartidos, se ha decidido implementar una variable de clase *parameters* de tal manera que todas las instancias compartan la instancia de dicho atributo ya que es el más costoso de generar con respecto a tiempo.

4.2 Implementación Commitment

Al igual que en la implementación del AKE, la clase Commitment tiene métodos abstractos que deben implementar las subclases. El método "generate_parameters" es el encargado de generar los parámetros a partir de los que, junto con el mensaje, se generará el compromiso. Como hemos visto en la sección **¡Error! No se encuentra el origen de la referencia.**, a la hora de generar el compromiso es necesario generar un valor aleatorio, para ello, la clase Commitment tiene como método abstracto "generate_random" que debe ser implementado en las subclases.

“generate_commitment” es el método encargado de generar el compromiso a partir del mensaje, de un valor aleatorio y los parámetros necesarios. El último de los métodos abstractos, es “generate_random” el cual es el encargado de validar que un compromiso se ha generado a partir de un determinado mensaje y valor aleatorio.

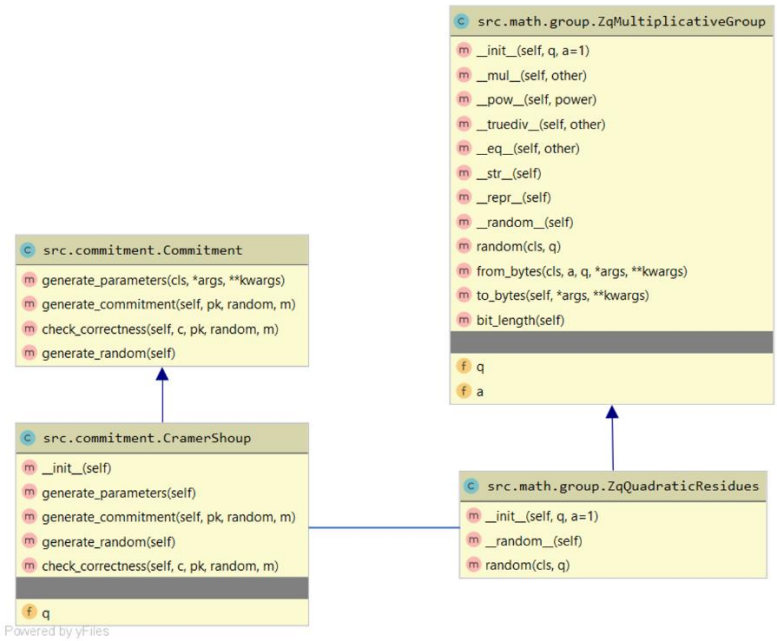

Ilustración 10 Detalle de la clase Commitment y relaciones

Para la implementación de la herramienta de compromisos se ha implementado a partir de la herramienta Cramer Shoup. Dicha herramienta se ha implementado a partir de la especificación que se detalla en [2]. Para dicha herramienta ha sido necesario implementar las herramientas matemáticas bajo las clases “ZqMultiplicativeGroup” y “ZqQuadraticResidues”.

4.3 Implementación Participant

Como se ha comentado con anterioridad, la clase Participant simula el comportamiento de los usuarios del protocolo. Para ello, se ha implementado métodos con nombres asociados a las rondas del protocolo para su trazabilidad.

Ilustración 11 Detalle de la clase Participant

En el método "round_0" el participante recibe como inputs la lista de los *uid* de todos los usuarios del protocolo y las claves K de los usuarios anterior y posterior. En dicha ronda, al igual que se especifica en la sección **¡Error! No se encuentra el origen de la referencia.**, se calcula las variables $\overleftarrow{K}$ y $\vec{K}$, cuyo equivalente en el código tiene por nombre *key_before* y *key_next* respectivamente.

El siguiente paso, es la ronda 1 que está implementada en el método "round_1", que recibe como input los parámetros para generar el compromiso. En esta función, cada participante utiliza el método "generate_random" de la clase Commitment para generar un valor aleatorio, y con ello el compromiso. Al final de la ronda, cada participante debe emitir en abierto el compromiso generado. Para simular dicho comportamiento, el método devuelve el compromiso generado como output y es el código que llama al método de encargado de emitir el mensaje.

La última ronda se divide en dos métodos, "round_2_1" y "round_2_2". La primera función, "round_2_1", tiene por función ejecutar la parte de broadcast de la ronda 2 del protocolo. Dichos mensajes son recibidos como input junto con los mensajes de la función "round_1" en el método "round_2_2". Este último método, comprueba que los compromisos se han generado correctamente. Si la comprobación es correcta, el método calcula las claves X_k de los usuarios y por ultimo las variables *sk* y *sid* que son la clave de sesión y el id de sesión respectivamente.

4.4 Simulación del protocolo

Como se ha comentado al principio de la sección 4, la clase “Participant” es aquella que tiene implementado el trabajo de cálculo del protocolo. Para la simulación del protocolo se ha diseñado dos modos de ejecución que se muestra a continuación:

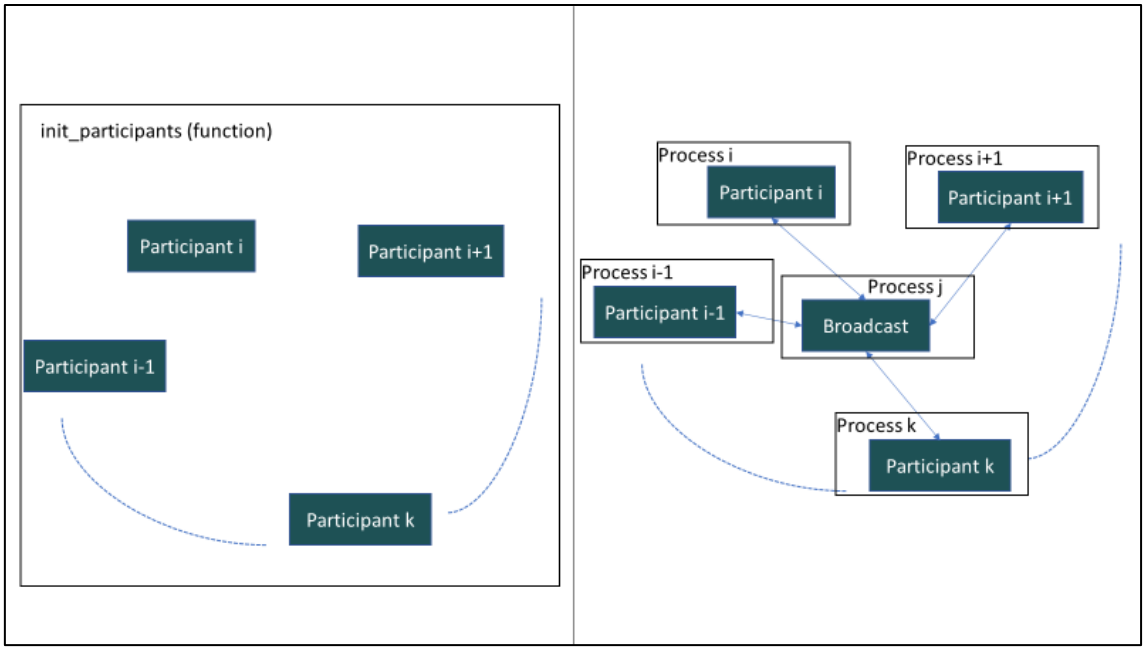

Ilustración 12 Diferencia entre los dos modos de ejecución

Como se puede ver en la ilustración, el modo de ejecución de la izquierda, simulación simple, es una ejecución de principio a fin dentro de una función en si misma que instancia a los participantes y simula ser el broadcast pasando a cada objeto los mensajes después de acabar las rondas. Esta versión es un modo de ejecución simplificado que permite validar el correcto funcionamiento tanto de las herramientas como del protocolo.

El modo de ejecución de la derecha o “realista” simula la comunicación de los participantes mediante un servidor broadcast al cual se conectan los participantes. Actualmente, el proceso main del repositorio levanta un proceso servidor el cual ofrece que todos aquellos que se conecten a él pueden mandar un mensaje que se retransmite al resto de procesos conectados. Por otro lado, el proceso principal (main), levanta n procesos que instanciarán a un objeto “Participant” y procesa la información recibida del broadcast ejecutando el método correspondiente de la instancia. Este modo permite una simulación de n usuarios que ejecutan el protocolo para generar la clave de sesión y el id de sesión.

5 Resultados

En esta sección se analizará los tiempos de ejecución de la implementación para cada una de sus simulaciones. Dichas simulaciones se han realizado en un equipo local, por lo que los usuarios y el servidor de las ejecuciones realizadas bajo la tipología simulación realista, se encuentran en procesos independientes, pero de un mismo equipo físico.

En la Ilustración 13, se visualiza el tiempo necesario para la ejecución del protocolo respecto al número de participantes involucrados en dicha ejecución. En azul se representa cada una de las ejecuciones realizadas bajo la simulación simple. Por otro lado, en rojo, se visualizan las ejecuciones de la simulación realista.

Como se puede apreciar, el tiempo que necesita la simulación simple es siempre superior a las ejecuciones realistas para el mismo número de participantes. Esto se debe a que la ejecución simple simula todos los participantes bajo un mismo proceso, de tal manera que no se paralelizan los cálculos que realiza cada usuario del protocolo. Por otro lado, la simulación simple escala peor cuanto más participantes tiene el protocolo.

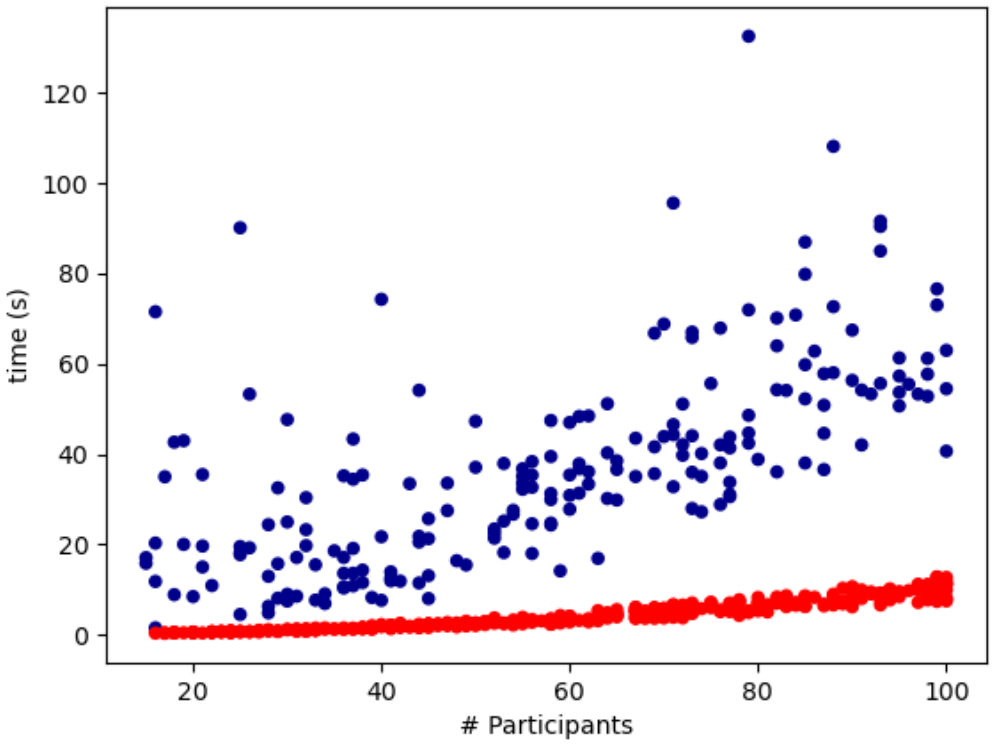

Ilustración 13 Grafico tiempo ejecución respecto al núm. de usuarios

En la Ilustración 14, se hace foco en las ejecuciones de la simulación realista. Como se puede observar, este tipo de simulaciones tampoco tiene una tendencia lineal respecto al numero de

participantes, pero debido a que los usuarios se simulan en procesos diferentes, los cálculos se paralelizan, lo cual permite tiempos razonables para la ejecución del protocolo.

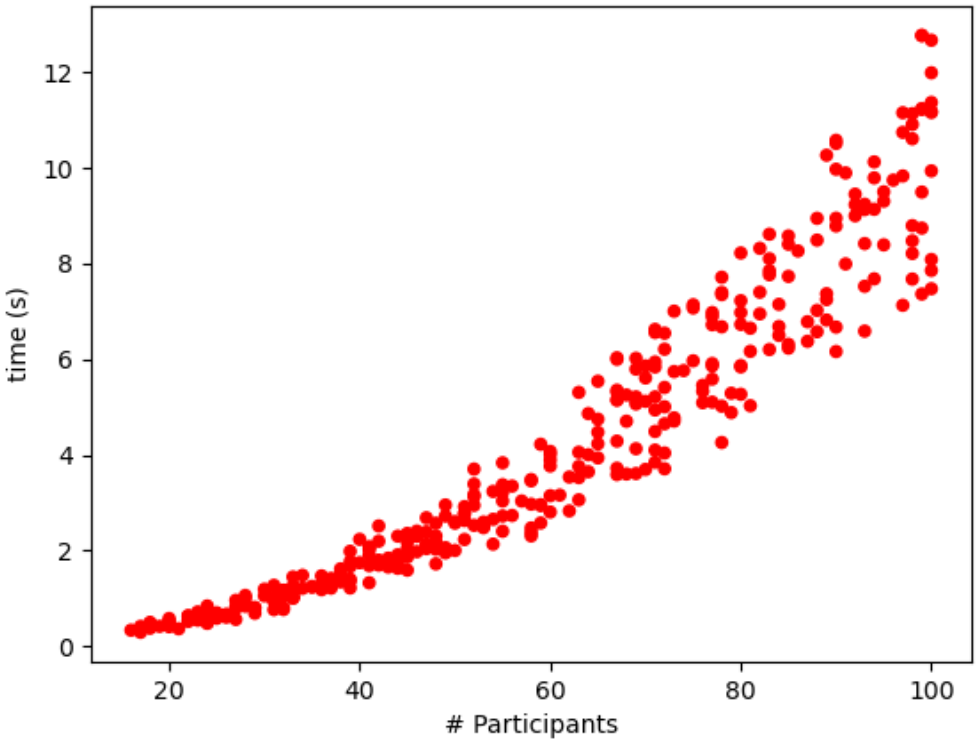

Ilustración 14 Grafico tiempo ejecución respecto al núm. de usuarios. Simulación realista

6 Conclusiones

Como se ha especificado en la sección 4, la implementación del protocolo se ha diseñado en base a interfaces. Este diseño del código, permite mayor facilidad de adaptación del protocolo a otras herramientas criptográficas.

El trabajo desarrollado tiene una futura evolución en la cual se adapten nuevas construcciones criptográficas para construir esquemas de grupo post-cuantico y analizar su aplicabilidad real realizando simulaciones realistas para distinto número de participantes.

7 Bibliografía

[1] M. Abdalla, J.-M. Bohli, M. I. González Vasco and R. Steinwandt, "(Password) Authenticated Key Establishment:From 2-Party To Group," 2007.

[2] A. Hänninen, «The Cramer-Shoup Public-Key,» 2006.

[3] S. Ulrick, «Implementation of Cramer-Shoup Cryptosystem,» 2017.

[4] «pyca/cryptography,» [En línea]. Available: https://cryptography.io/en/latest/.

[5] «The Blavatnik School of Computer Science,» 11 2011. Available: http://www.cs.tau.ac.il/~iftachh/Courses/FOC/Fall11/Slides/Commitments.pdf.

[6] «SymPy's documentation,». Available: https://docs.sympy.org/latest/index.html.

[7] «NYU Computer Science,» 12 2008. Available: https://cs.nyu.edu/courses/fall08/G22.3210-001/lect/lecture14.pdf.