Desarrollo de una plataforma para la evaluación de protocolos de seguridad descentralizados en IoT

Santiago Arias, Ana I. Gómez
Universidad Rey Juan Carlos
Calle Tulipán s/n
s.arias2020@alumnos,urjc.es
ana.gomez.perez@urjc.es

Cristian González García
Universidad de Oviedo
C/Federico García Lorca, nº18
gonzalezcristian@uniovi.es

Domingo Gómez-Pérez
Universidad de Cantabria
Avenida de los Castros s/n
domingo.gomez@unican.es

***Resumen*—El desarrollo del denominado Internet de las Cosas (IoT) presenta muchos retos en la actualidad, como por ejemplo en cuanto a la seguridad de sus datos. La privacidad de los datos requiere que los sistemas actuales aseguren la confidencialidad mientras se procesan datos de especial protección, bien sea en nuestros dispositivos, o en servidores externos. Esto requiere el desarrollo de protocolos criptográficos que permitan mantener la privacidad de los usuarios y a la vez sean transparentes en su ejecución en dispositivos heterogéneos de recursos limitados. En este trabajo se ha desarrollado un sistema multiplataforma descentralizado, que permita evaluar el rendimiento de protocolos criptográficos sobre dispositivos heterogéneos como se pueden encontrar en una red IoT. Como prueba de concepto se han elegido los protocolos de *Private Set Intersection (PSI)*, que permiten que las partes involucradas en una comunicación, puedan comprobar si comparten elementos en sus conjuntos de datos sin revelar información extra. Mediante esta plataforma hemos obtenido resultados iniciales sobre el rendimiento de dos criptosistemas previamente propuestos comparando diferentes esquemas en dispositivos de bajos recursos.**

***Index Terms*—Private Set Intersection, Internet of Things, evaluación de esquemas criptográficos**

I. Introducción

El desarrollo del Internet de las Cosas (IoT) ha producido un incremento significativo en el número de dispositivos que se conectan a Internet a través de redes, en general, descentralizadas. Los dispositivos IoT utilizados son muy heterogéneos, tanto en el hardware como en el software. En general, se trata de dispositivos de bajos recursos con conexión inalámbrica, empleados para automatizar y monitorizar procesos, ya sea en viviendas, ciudades o empresas. Esto requiere de esquemas de seguridad eficientes en los dispositivos adaptándose su seguridad a la heterogeneidad y ubicuidad de estas redes y los dispositivos de bajo poder computacional utilizados [1], como suelen ser microcontroladores (p.e. Arduino), diferentes smartphones de clases baja o media, microordenadores, o pulseras inteligentes. Por este motivo, es necesaria una arquitectura que permita la implementación de protocolos criptográficos basada en la necesidad de interoperabilidad, escalabilidad, seguridad, rendimiento y relevancia en aplicaciones del mundo real.

Generalmente, las plataformas IoT desarrolladas son centralizadas, como ocurre con el caso de Midgar propuesta inicialmente en 2014 [2], Xively, Sensorpedia, ThingSpeak, Carriots, AWS IoT, Azure IoT, etc. Se recomienda ver [3] para una revisión crítica actualizada de la literatura. Todas permiten el registro de diferentes dispositivos, el envío de datos a la red, y el envío de notificaciones desde ésta al dispositivo, o bien, la consulta de notificaciones pendientes en este último. Esto obliga a que toda la información esté centralizada y todo dispositivo tenga que cumplir los mismos estándares de conexión.

Una arquitectura de red descentralizada refleja la heterogeneidad y ubicuidad de los dispositivos, pero el desafío de este tipo de arquitecturas es la complejidad de la concurrencia, la red descentralizada y el registro de datos. Este reto permite complementar las demostraciones matemáticas de seguridad tradicionales con un uso más realista de la técnica en dispositivos con recursos limitados como se pueden encontrar en las nuevas redes IoT. Además, existen ventajas adicionales en una red descentralizada, como permitir realizar ≪Edge Computing≫ y ≪Fog Computing≫ de una forma más sencilla y dependiente del dispositivo final, así como una recogida de datos federada y con mayor privacidad [4].

Respecto a las redes IoT descentralizadas, algunas de ellas utilizan Blockchain [5], añadiendo una capa de seguridad, pero haciendo que pierda rendimiento cuando escala y siendo costosa la implementación debido a cómo funciona esta tecnología. Otras, se centran únicamente en la parte de la autenticación, por ejemplo el uso de mecanismos de seguridad descentralizados y distribuidos como Oauth2, GDOI y GNAP [6] debido a que los protocolos de autenticación no son adecuados para una gran cantidad de dispositivos. También existen arquitecturas que proveen ventajas adicionales, como es por ejemplo tener multitenencia [7], mejorar el aprovechamiento de recursos y beneficios [4], o sobre cómo se debería realizar el descubrimiento de dispositivos [8]. También son populares la adaptación de arquitecturas descentralizadas libres en Industrial Internet of Things (IIoT) [9] como podría ser el uso de BitTorrent, WireGuard, BorgBackup, Syncthing, TOR, B.A.T.M.A.N., NetBSD Syslogd, o The Tangle. Se conocen muchos usos de estas redes, incluidos usos ilegales, y algunas han sido vulneradas o tienen puertas traseras, como es el caso de TOR [10].

Nuestra propuesta está enfocada en las capacidades de comunicación seguras entre dispositivos heterogéneos en una red descentralizada, no en dispositivos IoT específicos. El objetivo del sistema desarrollado es ofrecer una forma de demostrar vulnerabilidades y medir tiempos que puede ayudar a la comunidad criptográfica y a posibles implementaciones productivas de un software. Para testear la propuesta nos centramos en la evaluación de protocolos de Private Set Intersection (*PSI*) [11], que permiten aprender los elementos comunes que hay en dos conjuntos cifrados sin revelar

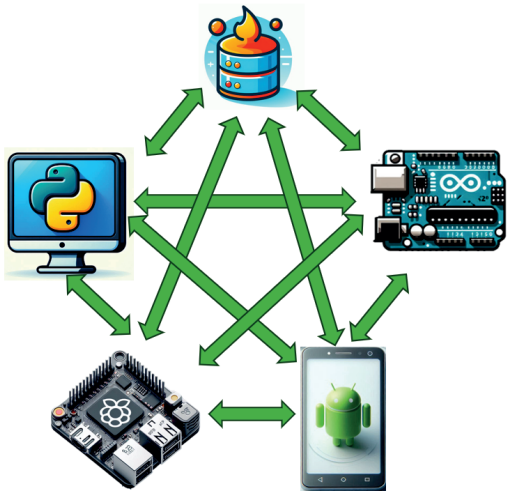

Figura 1: Comunicación entre dispositivos heterogéneos en una red IoT descentralizada

información adicional a ninguna de las partes implicadas.

La estructura del trabajo es la siguiente: La sección II describe la arquitectura y detalles de implementación de la plataforma así como una descripción de los esquemas PSI clásicos seleccionados para evaluar. Posteriormente, en la sección III se describen unos resultados iniciales de esta evaluación, para mostrar las capacidades de la plataforma. Finalmente se describen las conclusiones y trabajo futuro en la sección IV.

II. Arquitectura, Materiales y Métodos

II-A. Diseño de la plataforma

La red descentralizada que se propone se basa en servicios web que exponen una API REST y que utilizan el protocolo HTTP. Las aplicaciones Android están implementadas en Kotlin y Java y disponen de una interfaz gráfica para mostrar los dispositivos conectados a la misma red.

En la Figura 1 se muestra cómo se pueden comunicar de forma totalmente descentralizada los diferentes dispositivos entre ellos, así como en nuestro caso a una base de datos Firebase para registrar eventos, recursos consumidos y operaciones relativas a la investigación bajo petición. Además, la plataforma está diseñada para ser extensible a dispositivos de diferente capacidad de cómputo, como pueden ser PCs, microcontroladores, smartphones y microordenadores, siempre y cuando cumplan con el sistema de mensajes.

En esta versión inicial, tanto el sistema de servicio web como la plataforma Android permiten el descubrimiento de *peers* en redes de área local, o la posibilidad de hacer *ping* a otros usuarios de la red y la implementación de las primitivas básicas que dan soporte a un esquema PSI como la generación de claves criptográficas con un criptosistema o el cálculo de una intersección. La implementación modular permite la posible extensión a otras técnicas o variantes de PSI, las operaciones que se necesitan realizar por parte de los distintos criptosistemas, así como los criptosistemas en sí.

Queremos remarcar que el funcionamiento de la plataforma depende mucho de la concurrencia en el sentido de recepción de mensajes y ejecución de operaciones. No obstante, se implementan ejecutores prioritarios para hacer una gestión de la memoria lo más eficiente posible para los dispositivos.

Para el desarrollo de la red IoT se han utilizado las siguientes tecnologías:

- Flask [12] soporta la creación de servicios web y API REST.
- Waitress [13] implementa las funcionalidades del servidor Web Server Gateway Interface (WSGI) para desplegar los servicios web de Flask de forma que simule un entorno productivo.
- ZMQ [14] envía mensajes entre dispositivos en la red descentralizada, facilitando la comunicación.
- Firebase [15] como base de datos en tiempo real utilizada para registrar los recursos consumidos, tiempos de operación y otros datos relevantes, junto a Crashlytics como herramienta de informes de fallos utilizada en la aplicación Android para monitorizar y solucionar problemas.

El funcionamiento general es el siguiente: cuando el sistema se inicia, por defecto se levanta un nodo para escuchar y añade a los compañeros que se hayan preseleccionado, pudiendo apagarlo luego para levantar otro nodo que escuche en otro puerto distinto al original. Para la comunicación entre dispositivos se utilizan los socket *DEALER* de ZMQ, teniendo uno por dispositivo y permitiendo que podamos enviar mensajes. Para recibir mensajes se utiliza el socket *ROUTER* de ZMQ, que queda vinculado a la interfaz de red con la dirección de red de área local del dispositivo. Las aplicaciones, si disponen de la autenticación necesaria, se conectan también a la Firebase Realtime Database para enviar los registros. Se puede interactuar con el servicio web desde un navegador o mediante peticiones HTTP a la API REST.

A continuación se detallan tanto el funcionamiento, las operaciones y las capacidades de la plataforma.

Para la configuración inicial, cuando se inicia el nodo, el sistema crea un socket *ROUTER* que se configura con el valor de la IP local del dispositivo y un puerto específico para escuchar en la red. Este socket *ROUTER* actúa como punto de entrada para las conexiones entrantes.

Posteriormente la conexión entre dispositivos se realiza a través de mensajes de descubrimiento. Al iniciar, cada nodo crea un socket *DEALER* y envía un mensaje de *DISCOVERY* al dispositivo que quiere conocer. Los nodos receptores, al recibir dicho mensaje verifican si ya conocen al nodo remitente. Si no lo conocen, crean un socket *DEALER* para establecer una conexión directa con el nuevo nodo, respondiendo con un *DISCOVERY ACK*. Si ya lo conocía, le mandará una ≪prueba de vida≫ para indicarlo y que el nodo solicitante cree un socket *DEALER* permanente. Este mecanismo permite que los nodos se identifiquen y conecten entre sí fácilmente, añadiendo los nuevos nodos a su lista de dispositivos conocidos y asegurando una red interconectada.

Para manejar los mensajes se está utilizando un sistema de colas de ZMQ. En un socket *ROUTER* cada nodo se encarga de recibir y encolar los mensajes entrantes y, cuando hay un mensaje pendiente en la cola del *ROUTER*, este se desencola y se procesa según el tipo de mensaje. Se priorizan los mensajes que requieren menor capacidad de cómputo, como puede ser la respuesta a un *ping* o a un *DISCOVERY*, frente a mensajes

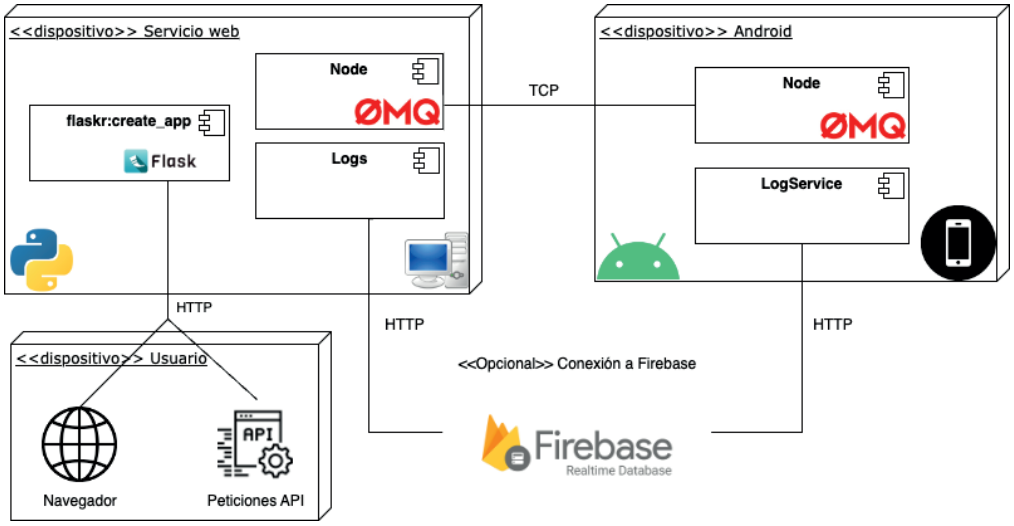

Figura 2: Diagrama de bloques que muestra el funcionamiento de la plataforma. Se muestran los servicios Web basados en Flask, como la conexión de dispositivos Android y la API REST expuesta. El sistema permite la conexión a una base de datos para el volcado de datos para su procesamiento.

que requieren más procesamiento como la recepción de un archivo JSON.

Las ventajas de tener un socket *ROUTER* en cada nodo permite que cada dispositivo escuche independientemente a otros nodos, facilitando la gestión de múltiples conexiones y mejorando el rendimiento al responder a los mensajes recibidos sin recibir información del estado del resto de dispositivos. De forma similar, tener un socket *DEALER* para cada dispositivo conocido en la red permite mandar los mensajes de forma independientemente y optimiza el rendimiento de la red. Finalmente el uso de socket *ROUTER* para la recepción y encolamiento de mensajes, permite una gestión ordenada del tráfico de datos.

La flexibilidad y escalabilidad del sistema viene dada por ZMQ y por las características de la arquitectura descentralizada que se ha desarrollado. En la Figura 2 se muestra un diagrama de bloques con el funcionamiento de alto nivel de la plataforma.

II-B. Esquemas de PSI evaluados

En esta subsección se introducen los esquemas evaluados en esta primera versión de la plataforma, elegidos por su interés en redes IoT. Se introducen los fundamentos teóricos necesarios para facilitar la comprensión de los experimentos realizados. Los protocolos de PSI constituyen un caso particular del problema de la Computación Segura Multiparte (SMPC), que es una familia de protocolos para el cálculo de una función matemática sobre una serie de entradas de datos que permanecen privadas y revelan solo el resultado de la operación. En este caso la función matemática es la intersección de los dos conjuntos de datos provistos en la entrada.

Una definición más formal es la siguiente: Alice y Bob tienen su conjunto de datos, los ≪conjuntos secretos≫:

$$A = \{a_1, \cdots, a_n\}, B = \{b_1, \cdots, b_m\}.$$

Alice quiere conocer $A \cap B$, sin revelarle ninguna información adicional a Bob en el proceso. Dependiendo de las variantes seleccionadas del protocolo a implementar, se puede hacer que Bob tampoco pueda conocer detalles de la intersección, con varios escenarios reflejados en la Tabla I. Para este problema, se asume que cada elemento es una cadena de bits con poca entropía por tener una estructura muy marcada, i.e. números de teléfono, direcciones de correo electrónico o palabras de un determinado idioma.

Tabla I: Modelo de compartición de la información entre Alice y Bob donde a, b, c y d son bits cualesquiera limitados por la restricción $a \vee b = \texttt{True}$ (al menos uno conoce la intersección) y $|A|$ representa la cardinalidad del conjunto A, i.e. el número de elementos.

	$A \cap B$	$\lvert A \cap B \rvert$	$\lvert A \rvert$	$\lvert B \rvert$
Alice	a	a	✓	d
Bob	b	b	c	✓

Teóricamente se ha demostrado que no existen protocolos seguros para PSI en cualquier variante de información desvelada (cualquiera de las dos partes o ambas descubren la intersección; se revela o no para cada parte el tamaño del conjunto ajeno) en un escenario en el que un atacante tenga acceso a recursos computacionales infinitos [16].

Aunque existen diferentes variantes que han sido propuestas para PSI, nos centraremos en dos esquemas: Private Matching (PM) y basado en dominio para realizar las pruebas de rendimiento sobre la plataforma desarrollada. Queremos remarcar que los modelos de seguridad usados se basarán en modelos de adversarios ≪semi-honestos≫ donde se requieren medidas adicionales que comprueben que alguna parte no está siendo engañosa, las partes no se desvian del protocolo de manera no maliciosa, o se obtienen ventajas no deseadas, sin poder subvertir el protocolo en sí.

II-B1. Private Match Intersection (PM): Este esquema requiere el uso de criptografía homomórfica e implementa Oblivious Polynomial Evaluation (OPE) [17], que sigue los pasos que explicaremos a continuación:

Se calculan los coeficientes de un polinomio que tiene como

raíces de los elementos de A

$$p(A) = \prod_{i=1}^{n} (X - a_i) = \sum_{i=0}^{n} p_i X^i, \ \forall i = 0, \cdots, n.$$

Estos coeficientes se cifran obteniendo $E(p_i)$ y se envía a B. Una vez recibidos, B evalúa el polinomio anterior con los elementos de su conjunto secreto, aprovechando las propiedades del cifrado homomórfico.

$$E(p(b_j)), \forall j \in 0, \cdots, m.$$

Tomando una r aleatoria, B utiliza la clave pública de A para cifrar los siguientes datos datos:

$$E_j = E(r \cdot p(b_j) + b_j)$$

Finalmente, A recibe el resultado E_j y descifra la intersección

$$D_j = D(E_j), \forall j \in 0, \cdots, m$$

Entonces puede comprobar si $D_j \in A$, siguiendo que

$$D(E(r \cdot p(b_j) + b_j)) = b_j \Leftrightarrow p(b_j) = 0 \Leftrightarrow b_j \in A$$

Este esquema permite que B no conozca el resultado de la evaluación cifrando la información también a posibles atacantes. Una posible debilidad es que existe probabilidad no nula de que haya una colisión si $b \notin X$ tal que $E(r \cdot p(b) + b)$ coincide con el cifrado de un elemento de X.

La variante de cardinalidad OPE-C permite obtener el tamaño de la intersección por el siguiente cálculo por parte de B, $E(r \cdot p(b_j) + 0^+)$. La parte A cuenta las cadena de ceros recibidas para recuperar el número de elementos en común. Es importante cambiar el orden de envío, bien de la evaluación o del resultado, para que no se revelen datos posicionales.

II-B2. Intersecciones calculadas según el dominio: Una alternativa cuando se trabaja con dispositivos IoT es calcular las intersecciones aprovechando propiedades homomórficas de la operación multiplicación. Esto requiere que A y B compartan un mismo dominio de datos D, esto es $A, B \subset D$. Es un esquema sencillo para dominios suficientemente pequeños como por ejemplo un termostato inteligente que regule la temperatura entre varios sensores de un edificio, sin desvelar los valores particulares. Una implementación se propuso como protocolo sencillo para el rastreo de contagiados por COVID [18] .

El proceso se inicia con A codificando un 0 o un 1, dependiendo de si los elementos del dominio están en su conjunto de datos.

$$f_A(x) = \begin{cases} E(0) & \text{si } x \notin A \\ E(1) & \text{si } x \in A \end{cases} \quad \forall x \in D$$

Después, A envía el resultado a B, que recibe los datos. En función de si el elemento está en su conjunto secreto, cifra un 0 con la clave pública de A si no pertenece o multiplica el valor del elemento por el escalar dos si pertenece a su conjunto.

$$f_B(x) = \begin{cases} E(0) & \text{si } x \notin B \\ E(f_a(y) \cdot 2) & \text{si } x \in B, \forall y \in A \end{cases}$$

Finalmente, A recibe la composición de estas dos funciones f_A y f_b para todos los elementos del dominio D. Al descifrar los datos $Dec_A(x) = 2$ si $x \in A \cap B$ y 0 en otro caso.

Este esquema revela más información que el anterior, dado que A puede inferir información sobre elementos de B que no estén en su conjunto. También puede convertirse en algo ineficiente si el conjunto de datos sobre el que se opera es grande y depende más del rendimiento del esquema criptográfico, que tiene que tener condición de probabilístico.

III. Resultados

Se han seleccionado dos criptosistemas clásicos y probabilistas con propiedades homomórficas en sus operaciones para realizar la comparación.

Paillier es un criptosistema de clave pública que tiene las propiedades de ser homomórfico, aditivo, y probabilístico [19]. Su seguridad se apoya en la dificultad de calcular residuos para números grandes basado en el denominado problema ≪Decisional Composite Residuosity Assumption (DCRA)≫. En concreto, la dificultad computacional del problema se establece en que dado un número entero z y un parámetro no primo n, hay que decidir si existe un y tal $z = y^n \mod n^2$ [19].

Para la generación de claves criptográficas, se eligen dos primos "grandes" (p y q) a partir de los cuales se calcula $\lambda = \mathrm{mcm}(p-1, q-1)$, donde la función mcm representa el mínimo común múltiplo. Para la clave pública se toma $n = pq$, y $g \in \mathbb{Z}^*_{n^2}$, esto es en el conjunto de enteros sin factores primos en común con n^2, haciendo módulo con n^2.

Damgård–Jurik [20] es una generalización del criptosistema de Paillier. A diferencia de Paillier, que tiene un espacio de cifrado $\mathbb{Z}_{n^2}$, Damgård–Jurik amplía el espacio de cifrado a $\mathbb{Z}_{n^{s+1}}$, donde s es un entero positivo. Esto quiere decir que Damgård–Jurik proporciona una mayor flexibilidad pudiendo parametrizar *s* como factor de expansión, y una mayor seguridad al trabajar en espacios de cifrado mayores. Esto hace que abarque un mayor abanico de aplicaciones en criptografía que Paillier.

Para la configuración de las pruebas se han utilizado varios dispositivos, en particular un Mac Studio (Apple M1 Max - 10 núcleos a 3.228 GHz - 32 GB), MacBook Pro (Apple M1 Pro - 8 núcleos a 3.228 GHz - 16 GB) y un Samsung Galaxy S21 Ultra. Se han utilizado implementaciones de terceros en Python para Paillier [21] y Damgård-Jurik [22], mientras que se han desarrollado implementaciones propias en Android.

Para la configuración de los parámetros de prueba se ha elegido un dominio de 500 elementos, con conjuntos de datos de 50 elementos aleatorios. Ambos criptosistemas se han configurado con un tamaño de clave de 2.048 bits y un factor de expansión de $s = 2$ para Damgård-Jurik. Con estos valores se han realizado 1.000 peticiones y se han hallado los valores medios de tiempo y memoria ocupada para cada etapa del protocolo (cifrado de datos, evaluación de la intersección y descifrado junto con la obtención del resultados de conjunto de la intersección. La Tabla II resume los resultados temporales obtenidos cuando un dispositivo Android solicita el protocolo PSI contra el servicio web del servidor. Tanto el protocolo OPE como su variante de cálculo de la cardinalidad OPE-C, muestran valores menores para ambos criptosistemas, que el esquema de dominio. Esto es debido a que este último opera sobre todo el dominio de datos, siendo más ineficiente en todas las operaciones. Estas

Tabla II: Comparación de tiempo en segundos cuando un dispositivo Android inicia un PSI contra un servicio web para los tres pasos de codificación y cifrado del conjunto de datos (Cifrado), operaciones de intersección (Evaluación) y obtención del resultado de la operación PSI (Descifrado) .

Cifrador	PSI	Cifrado	Evaluación (Servidor)	Descifrado
Paillier	Dominio	18,37	19,59	36,51
	OPE	2,29	7,57	2,64
	OPE-C	2,09	7,27	2,44
Damgård-Jurik (s=2)	Dominio	77,11	64,83	78,84
	OPE	8,02	5,38	7,31
	OPE-C	8,56	4,06	7,86

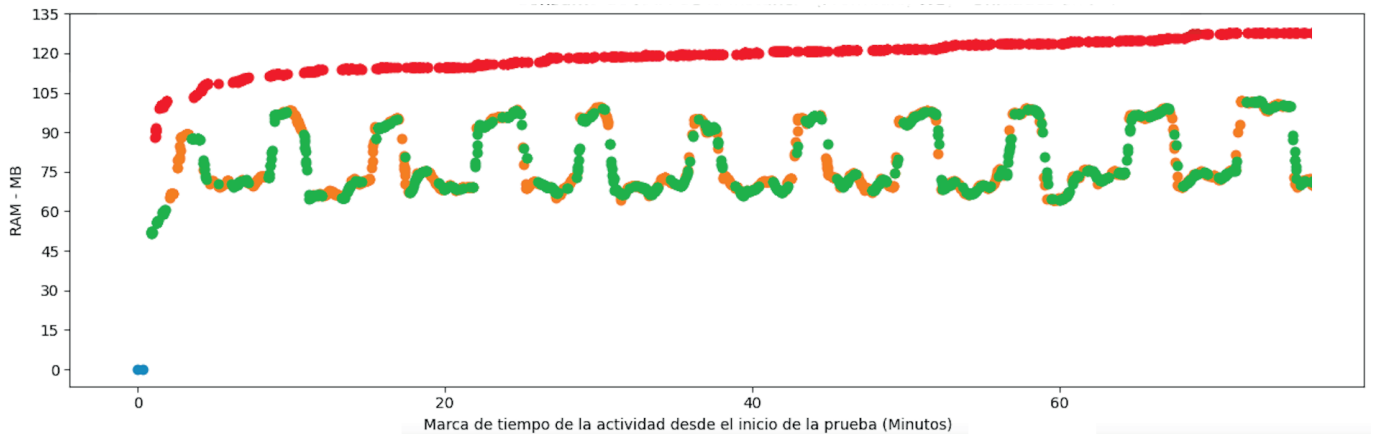

Figura 3: Uso de RAM en la ejecución de una prueba sobre dos equipos Mac (Alice y Bob) para 1000 peticiones en un intervalo temporal. Se ha medido interacciones sobre las etapas de Generación de clave (azul), codificación y cifrado de datos (verde), operaciones de evaluación (rojo) y operaciones de descifrado (naranja)

.

simulaciones iniciales muestran peor rendimiento en el caso de Damgård-Jurik en general.

La plataforma también puede realizar mediciones de memoria en relación con la gestión de las peticiones. Los resultados se pueden ver en la Figura 3, que representa una medición realizada con el servidor web sobre equipos Mac para el criptosistema de Paillier. En este caso la liberación de la memoria se realiza periódicamente, siendo la correspondiente al caso de generación de clave casi nula debido a que la precisión del sistema de medición no ha sido suficiente para el intervalo temporal.

IV. Conclusiones y trabajo futuro

La plataforma presentada está diseñada para que la API REST y la interfaz de los servicios web provean de un sistema sencillo para probar esquemas criptográficos sobre redes IoT descentralizadas. La propuesta implementada es extensible para la evaluación del rendimiento de implementaciones criptográficas sobre una red de dispositivos heterogéneos. Esta plataforma también aporta un sistema de registros robusto, que reporta cada evento relevante a una base de datos para su extracción y estudio. Con esto se cumple el objetivo de ofrecer un banco de pruebas descentralizado.

Como prueba de concepto se han realizado experimentos en los que se han comparado dos esquemas PSI, que utilizan dos criptosistemas: Paillier y Damgård-Jurik con distintas configuraciones, ofreciendo medidas sobre los tiempos de ejecución y el consumo de RAM, que parecen desaconsejar Damgård-Jurik para su uso en este tipo de redes, así como el esquema de cifrado en dominio, salvo en aplicaciones concretas. La interpretación de estos resultados coincide con la creencia en la comunidad científica, es decir, su utilidad es limitada usando la versión estándar. Ruan et al. [23] ofrecen una mejora sobre Paillier que no extienden a Damgård-Jurik en una variedad de escenarios semihonestos y que sería interesante extender para realizar una comparación de tiempos. Posteriormente, otros autores [24] han seguido con más comparaciones entre ambos, pero limitándose a constatar el rendimiento del cifrado en computadores.

Es necesario realizar más pruebas de rendimiento sobre la plataforma para poder caracterizar completamente el funcionamiento de los esquemas propuestos debido a que las pruebas se han hecho para comprobar su correcto funcionamiento. La última versión de la plataforma se puede encontrar en Github con su parte en Android[1], y otro para el desarrollo del servicio Web[2]. Se facilita una guía de instalación y uso de cada parte de la plataforma.

Como mejoras futuras se está trabajando en la extensión de la plataforma para añadir más dispositivos IoT, así como el soporte a otros esquemas y protocolos criptográficos. Para la extensión referente a los protocolos criptográficos estamos investigando implementaciones de criptosistemas resistentes a la computación cuántica como NTRU [25], Cheon-Kim-Kim-

[1]Repositorio Android: https://github.com/4rius/APP_PSI
[2]Repositorio Python: https://github.com/4rius/WS_PSI

Song (CKKS) [26] o Brakerski-Fan-Vercauteren (BFV) [27]. Respecto a la parte de dispositivos IoT, estamos probando herramientas de emulación con buena interoperabilidad y de código libre.

También se está trabajando en mejoras de uso y rendimiento de la plataforma como la sustitución de la actual base de datos Firebase por una alternativa más eficiente y escalable para el guardado de experimentos y su posterior análisis. Finalmente se está estudiando la generación de una imagen de Docker para la parte de servicios web que permita que la instalación sea más sencilla de utilizar para investigadores interesados en su uso.

AGRADECIMIENTOS

Los autores agradecen los comentarios de los revisores que han contribuido sustancialmente a la mejora del trabajo.

Domingo Gómez-Pérez está parcialmente financiado por el proyecto "PROTOCOLOS SEGUROS EN REDES DESCENTRALIZADAS.(AYUDA FINANCIADA CONTRATO PROGRAMA GOB CANTABRIA - UC)", además esta publicación es parte de la CÁTEDRA UNIVERSIDAD DE CANTABRIA-INCIBE DE NUEVOS RETOS EN CIBERSEGURIDAD, financiada por la Unión Europea NextGeneration-EU, Plan de Recuperación, Transformación y Resiliencia, a través de INCIBE.

REFERENCIAS

[1] C. Maple, "Security and privacy in the Internet of Things," *Journal of cyber policy*, vol. 2, no. 2, pp. 155–184, 2017.

[2] C. G. García, J. P. Espada, E. R. N. Valdez, and V. García-Díaz, "Midgar: Domain-specific language to generate smart objects for an Internet of Things platform," in *2014 Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 7 2014, pp. 352–357.

[3] C. G. García, D. Meana-Llorian, V. García-Diaz, A. C. Jiménez, and J. P. Anzola, "Midgar: Creation of a graphic domain-specific language to generate smart objects for Internet of Things scenarios using model-driven engineering," *IEEE Access*, vol. 8, pp. 141 872–141 894, 2020.

[4] J. Mocnej, W. K. Seah, A. Pekar, and I. Zolotova, "Decentralised IoT architecture for efficient resources utilisation," *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 168–173, 2018, 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018.

[5] C. Y. Kim, B. C. Ng, J. Sahni, N. Samian, and W. K. Seah, "Blockchain network platform for IoT data integrity and scalability," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. IEEE, 10 2023, pp. 162–171.

[6] M. Mukhandi, "Decentralised and scalable security for IoT devices," in *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. ACM, 11 2021, pp. 411–412.

[7] S. Cherrier, Z. Movahedi, and Y. M. Ghamri-Doudane, "Multi-tenancy in decentralised IoT," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 12 2015, pp. 256–261.

[8] R. Kurte, Z. Salcic, and K. I.-K. Wang, "Decentralised global service discovery for the Internet of Things," *Sensors*, vol. 24, p. 2196, 3 2024.

[9] S. Plaga, N. Wiedermann, S. D. Anton, S. Tatschner, H. Schotten, and T. Newe, "Securing future decentralised industrial IoT infrastructures: Challenges and free open source solutions," *Future Generation Computer Systems*, vol. 93, pp. 596–608, 4 2019.

[10] Q. Tan, X. Wang, W. Shi, J. Tang, and Z. Tian, "An anonymity vulnerability in tor," *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2574–2587, 2022.

[11] D. Morales, I. Agudo, and J. López, "Private set intersection: A systematic literature review," *Computer Science Review*, vol. 49, p. 100567, 2023.

[12] Pallets, "Flask version 3.0," https://flask.palletsprojects.com/en/3.0.x/, 2023, accessed: April 2024).

[13] Pylons, "Waitress version 3.0," https://github.com/Pylons/waitress, 2024, accessed: April 2024).

[14] B. E. Granger and M. Ragan-Kelley, "PyZMQ," https://github.com/zeromq/pyzmq, 2012, accessed: April 2024).

[15] Google, "Firebase," https://firebase.google.com/docs/database/, 2023, accessed: April 2024).

[16] P. D'arco, M. I. Gonzalez Vasco, A. Pérez del Pozo, C. Soriente, and R. Steinwandt, "Private set intersection: New generic constructions and feasibility results." *Advances in mathematics of communications*, vol. 11, no. 3, 2017.

[17] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, pp. 1–19.

[18] OpenMined, "CovidAlert," https://github.com/OpenMined/covid-alert, 2020, accessed: April 2024).

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[20] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography*, K. Kim, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 119–136.

[21] CSIRO's Data61, "Python Paillier library," https://github.com/data61/python-paillier, 2013.

[22] N. Boucher, L. Govedič, P. Saowakon, and K. Swanson, "Python Damgard Jurik library," https://github.com/cryptovoting/damgard-jurik, 2019.

[23] O. Ruan, Z. Wang, J. Mi, and M. Zhang, "New approach to set representation and practical private set-intersection protocols," *IEEE Access*, vol. 7, pp. 64 897–64 906, 2019.

[24] G. Dimitoglou and C. Jim, "Performance evaluation of partially homomorphic encryption algorithms," in *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2022, pp. 910–915.

[25] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *International algorithmic number theory symposium*. Springer, 1998, pp. 267–288.

[26] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[27] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Paper 2012/144, 2012. [Online]. Available: https://eprint.iacr.org/2012/144