

UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Ingeniería Informática



**Protocolo resistente a trampas descentralizado para juegos de
adivinanza en redes**

Jamil Enrique Navarro Afanador

Profesor guía: Víctor Parada
Profesor co-guía: Carlos Ormeño

Trabajo de graduación en conformidad a los
requisitos para obtener el grado de Magíster en
Seguridad, Peritaje, y Auditoría de Procesos
Informáticos

Santiago – Chile
2018

© **Jamil Enrique Navarro Afanador, 2018**



Algunos derechos reservados.

Esta obra está bajo una **Licencia Creative Commons Atribución-NoComercial-SinDerivadas Chile 3.0**. Sus condiciones de uso pueden ser revisadas en:

<http://creativecommons.org/licenses/by-nc-nd/3.0/cl/>.

Resumen

Hoy en día Internet está pasando por un proceso de centralización. En la medida en la cual los gigantes de Internet se consolidan, crece su dominio sobre la información que reciben los cibernautas. Esto significa un riesgo de pérdida de privacidad para las personas, y una asimetría enorme entre estos conglomerados y sus usuarios. En relación a estos eventos, Tim Berners-Lee, creador de la web, manifestó sentirse devastado. Así mismo, indicó que actualmente está trabajando en un proyecto para “re-decentralizar la web”.

El problema escogido son las trampas en los videojuegos de red. La razón es que la prevalencia de las trampas en estos juegos ahuyenta a los jugadores honestos. Por eso la calidad de los videojuegos en red se mide tanto por su desempeño, como por su resistencia a las trampas. Y siendo los videojuegos de red una forma importante de compartir actualmente con familiares y amigos, las trampas se convierten en un obstáculo para la socialización.

Las estrategias de prevención de trampas han variado de forma progresiva. Inicialmente, los esfuerzos se centraron en ofuscar los datos en el cliente para evitar el acceso indebido a la información. La siguiente estrategia fue mantener el estado general del juego solamente en el servidor. El servidor recibe las jugadas de cada cliente, las consolida, y luego envía a los clientes sólo la información que necesitan. Las desventajas que se citan con mayor frecuencia de este enfoque son que no escala bien con un aumento de usuarios o de partidas, y que el gran consumo de recursos por parte del servidor implica un alto costo. En consecuencia, la transición actual es hacia arquitecturas donde los jugadores se comunican directamente entre sí, pero el servidor sigue siendo responsable de evitar las trampas. Este trabajo busca ir un paso más allá y promover una solución donde los jugadores se comuniquen de forma descentralizada, no dependan de un punto único de falla para la prevención de trampas, ni tengan que confiar montones de datos sobre sí mismos y sus actividades a un tercero. Es decir, se enmarca en este mismo espíritu del esfuerzo de Tim Berners-Lee.

Para ello, este trabajo propone un protocolo descentralizado para jugar partidas de juegos de adivinanza en red. En estos juegos, los jugadores deben adivinar un secreto. Los objetivos que se desean proteger son la integridad y confidencialidad del secreto desde que es seleccionado hasta que culmine la partida, la autenticidad de las jugadas durante la partida, y la validez de las respuestas a los intentos de adivinación. Algunos ejemplos de estos juegos son: El Ahorcado, Battleship, y Adivina Quién. Este trabajo se centra en el juego de El Ahorcado.

Palabras clave: juegos de adivinanza, resistencia a trampas, protocolo criptográfico.

Abstract

Nowadays the Internet is undergoing a centralization process. As Internet giants consolidate, their hold on user's information continues to grow. This means a loss of privacy for people, and an enormous asymmetry between these holdings and their users. Tim Berners-Lee, inventor of the World Wide Web, has said he feels devastated about these developments. He has also said that he is currently working on a project to "re-decentralize the Web".

The problem addressed is cheating in network video-games. The reason is cheating prevalence drives away honest players. This explains why network video-games are not only measured by their performance, but also by their cheat resistance. Since network video-games are an important way to share with family and friends, cheating becomes an obstacle to socialization.

Cheat prevention tactics have changed over time. Initial approaches focused on the obfuscation of data residing in the client, to protect it from users. The next strategy was storing the game state only in a server. Servers receive plays from each client, consolidate them, and then only send each client the information they need. Most cited disadvantages of this approach are that it does not scale, and the high costs the server generates. Therefore, games are currently being transitioned to architectures where players communicate directly with other players, but a server remains in charge of cheat prevention. This work seeks to go a step further in promoting systems that communicate peer-to-peer, rooting out single points of failure in cheat prevention, and preventing users from having to trust third parties with troves of information on themselves, and their activities. In other words, it follows the same spirit as Tim Berners-Lee's efforts.

Therefore, this work proposes a decentralized network protocol to play guessing games. In these games, players win by guessing a secret. The security objectives that must be protected are the integrity and confidentiality of the secret from the moment it's chosen until the game ends, the authenticity of plays during the game, and the validity of responses to guess attempts. Some examples of this games are: Hangman, Battleship, and Guess Who. This work focuses on Hangman.

Keywords: guessing games, cheat resistance, cryptographic protocol.

Dedicatoria

Este trabajo se lo dedico a mi mamá y a mi esposa, quienes han sido mis mayores apoyos en la vida, y me acompañaron en muchos de estos desvelos.

Agradecimientos

A Dios por todas las bendiciones que ha puesto en mi camino.

A mi co-tutor Carlos Ormeño, por su apoyo durante el desarrollo de este trabajo.

A los profesores de la USACH, quienes con su buena disposición, hicieron de este programa un aprendizaje de vida.

A mis compañeros de estudio del Magíster, quienes me acompañaron en esta travesía.

Tabla de contenido

| | | |
|----------|-----------------------------------------------------------|-----------|
| 1 | Introducción..... | 1 |
| 1.1 | Antecedentes y motivación..... | 1 |
| 1.2 | Descripción del problema..... | 3 |
| 1.3 | Solución propuesta..... | 4 |
| 1.3.1 | Características de la solución..... | 4 |
| 1.3.2 | Propósito de la solución..... | 5 |
| 1.4 | Objetivos y alcances del proyecto..... | 5 |
| 1.4.1 | Objetivo general..... | 5 |
| 1.4.2 | Objetivos específicos..... | 5 |
| 1.4.3 | Alcances..... | 5 |
| 1.4.3.1 | Alcance..... | 5 |
| 1.4.3.2 | Limitaciones..... | 5 |
| 1.5 | Metodologías y herramientas utilizadas..... | 6 |
| 1.5.1 | Metodología de desarrollo..... | 6 |
| 1.5.1.1 | Explicar el problema..... | 7 |
| 1.5.1.2 | Definir los requisitos..... | 7 |
| 1.5.1.3 | Diseñar y desarrollar el artefacto..... | 7 |
| 1.5.1.4 | Mostrar el artefacto..... | 7 |
| 1.5.1.5 | Evaluar el artefacto..... | 8 |
| 1.5.2 | Herramientas de desarrollo..... | 8 |
| 1.5.3 | Ambiente de desarrollo..... | 9 |
| 1.6 | Organización del documento..... | 9 |
| 2 | Marco teórico..... | 10 |
| 2.1 | Método de investigación..... | 10 |
| 2.2 | Juego “El Ahorcado”..... | 10 |
| 2.3 | Principios de diseño..... | 12 |
| 2.4 | Arquitectura de Seguridad..... | 13 |
| 2.4.1 | Dimensiones de Seguridad..... | 13 |
| 2.4.2 | Capas de seguridad..... | 14 |
| 2.4.3 | Planos de seguridad..... | 14 |
| 2.4.4 | Amenazas..... | 15 |
| 2.4.5 | Vulnerabilidades..... | 15 |
| 2.4.6 | Ataques..... | 16 |
| 2.5 | Conceptos criptográficos relevantes..... | 16 |
| 2.5.1 | Hash..... | 16 |
| 2.5.1.1 | Resistencia a colisiones..... | 16 |
| 2.5.1.2 | Protección de contraseñas con funciones <i>hash</i> | 17 |
| 2.5.1.3 | Problema de cumpleaños..... | 19 |
| 2.5.1.4 | Defensa en profundidad..... | 20 |
| 2.5.2 | Esquema de compromiso..... | 20 |
| 2.6 | Trabajos similares..... | 21 |
| 2.7 | Discusión..... | 22 |
| 2.7.1 | Dimensiones de seguridad..... | 23 |
| 2.7.2 | Capas de seguridad..... | 25 |
| 2.7.3 | Planos de seguridad..... | 25 |
| 2.7.4 | Amenazas contempladas..... | 25 |
| 2.7.4.1 | A1. El Seleccionador pierde el secreto..... | 26 |
| 2.7.4.2 | A2. Se pierde una respuesta del Seleccionador..... | 26 |
| 2.7.4.3 | A3. Se pierde un intento del Adivinador..... | 26 |
| 2.7.4.4 | A4. El Seleccionador altera el Secreto..... | 27 |

| | | |
|----------|---------------------------------------------------------------------------|-----------|
| 2.7.4.5 | A5. El Seleccionador recibe un intento falsificado..... | 27 |
| 2.7.4.6 | A6. El Adivinador recibe una respuesta falsificada..... | 27 |
| 2.7.4.7 | A7. El Adivinador repudia un intento..... | 27 |
| 2.7.4.8 | A8. El Seleccionador repudia una respuesta..... | 27 |
| 2.7.4.9 | A9. El Adivinador obtiene información indebida sobre el Secreto..... | 27 |
| 2.7.4.10 | A10. Se rompe la comunicación entre el Adivinador y el Seleccionador..... | 28 |
| 2.7.5 | Vulnerabilidades conocidas..... | 28 |
| 2.7.5.1 | V1. El Seleccionador no se compromete a un valor del Secreto..... | 28 |
| 2.7.5.2 | V2. El compromiso es el Secreto en claro..... | 28 |
| 2.7.5.3 | V3. El compromiso revela información sobre el Secreto..... | 29 |
| 2.7.5.4 | V4. El compromiso es un hash sin salt o el salt es conocido..... | 29 |
| 2.7.5.5 | V5. El compromiso es una firma sin nonce o el nonce es conocido..... | 29 |
| 2.7.5.6 | V6. El algoritmo de hash o firma es vulnerable a ataques de colisión..... | 30 |
| 2.7.5.7 | V7. Los mensajes no están firmados..... | 30 |
| 2.7.5.8 | V8. Los mensajes no tienen contexto de sesión o partida..... | 30 |
| 2.7.6 | Ataques relevantes..... | 30 |
| 2.7.6.1 | Ataque con tabla arco iris..... | 30 |
| 2.7.6.2 | Ataque de diccionario..... | 31 |
| 2.7.6.3 | Ataque de fuerza bruta..... | 31 |
| 2.7.6.4 | Ataque de Colisión..... | 32 |
| 2.7.6.5 | Falsificación de mensaje..... | 32 |
| 2.7.6.6 | Reenvío de mensaje..... | 32 |
| 2.7.6.7 | Saturación de la red..... | 32 |
| 2.7.6.8 | Desvío o impedimento de tráfico de red..... | 33 |
| 2.7.6.9 | Intercepción de tráfico de red..... | 33 |
| 2.7.6.10 | Distintas versiones de un Mensaje..... | 33 |
| 2.7.6.11 | Compromiso del cliente/cliente comprometido..... | 33 |
| 2.7.7 | Adaptación de procedimiento antitrampas actual..... | 33 |
| 2.7.7.1 | Debilidades del procedimiento antitrampas actual..... | 35 |
| 2.7.8 | Mejoras propuestas..... | 36 |
| 2.7.8.1 | Nivel Mínimo: Secreto..... | 37 |
| 2.7.8.2 | Nivel Intermedio: Posiciones, Elementos, o partes del Secreto..... | 38 |
| 2.7.8.3 | Nivel Máximo: Universo de movidas..... | 38 |
| 2.7.8.4 | Consideraciones sobre el tamaño del universo de movidas..... | 39 |
| 2.8 | Enfoques de desarrollo de la solución..... | 42 |
| 2.8.1 | Habilidades de los jugadores y tramposos..... | 43 |
| 2.8.2 | Arquitectura de red..... | 43 |
| 2.8.3 | Resguardo del estado de juego..... | 43 |
| 2.9 | Selección de enfoque de solución y justificación..... | 43 |
| 3 | Diseño del protocolo..... | 45 |
| 3.1 | Requisitos..... | 45 |
| 3.1.1 | Funcionales..... | 45 |
| 3.1.2 | No funcionales..... | 45 |
| 3.1.2.1 | Interoperabilidad..... | 45 |
| 3.1.2.2 | Portabilidad..... | 46 |
| 3.1.2.3 | Eficiencia..... | 46 |
| 3.1.2.4 | Arquitectura..... | 46 |
| 3.1.2.5 | Seguridad..... | 46 |
| 3.1.2.6 | Licenciamiento..... | 46 |
| 3.2 | Estructura de los Mensajes..... | 46 |
| 3.2.1 | Mensaje lets_play..... | 48 |
| 3.2.2 | Mensaje accept_play..... | 49 |
| 3.2.3 | Mensaje reject_play..... | 49 |
| 3.2.4 | Mensaje play_ack..... | 50 |

| | | |
|----------|---------------------------------------------------------------------------------|-----------|
| 3.2.5 | Mensaje nonce_start..... | 50 |
| 3.2.6 | Mensaje r_commit..... | 50 |
| 3.2.7 | Mensaje r_reveal..... | 50 |
| 3.2.8 | Mensaje w_commit..... | 51 |
| 3.2.9 | Mensaje w_ack..... | 51 |
| 3.2.10 | Mensaje make_a_try..... | 51 |
| 3.2.11 | Mensaje new_try..... | 51 |
| 3.2.12 | Mensaje reply..... | 52 |
| 3.2.13 | Mensaje r_ack..... | 52 |
| 3.2.14 | Mensaje game_ended..... | 52 |
| 3.2.15 | Mensaje result_validation..... | 53 |
| 3.2.16 | Mensaje result_ack..... | 53 |
| 3.2.17 | Mensaje validation_error..... | 53 |
| 3.2.18 | Mensaje unexpected_message..... | 53 |
| 3.2.19 | Mensaje quit..... | 54 |
| 3.3 | Secuencia de los Mensajes..... | 54 |
| 3.3.1 | Negociación de los parámetros de la partida..... | 54 |
| 3.3.2 | Generación de compromisos sobre la palabra..... | 56 |
| 3.3.3 | Bucle de juego..... | 60 |
| 3.3.4 | Cierre de la partida..... | 63 |
| 3.3.5 | Abandono de la partida..... | 63 |
| 3.3.6 | Validación de mensajes..... | 63 |
| 3.3.7 | Manejo de errores..... | 64 |
| 3.4 | Cifrado de la comunicación..... | 66 |
| 4 | Desarrollo del Prototipo..... | 67 |
| 4.1 | Funcionalidad a implementar..... | 68 |
| 4.1.1 | Primera fase..... | 68 |
| 4.1.2 | Segunda fase..... | 69 |
| 4.1.3 | Tercera fase..... | 69 |
| 4.1.4 | Cuarta fase..... | 70 |
| 4.1.5 | Quinta fase..... | 70 |
| 4.2 | Plan de trabajo..... | 70 |
| 4.3 | Arquitectura del prototipo..... | 71 |
| 4.4 | Resumen..... | 72 |
| 5 | Verificación y validación..... | 75 |
| 5.1 | Evaluación teórica de seguridad..... | 75 |
| 5.1.1 | C1. Los mensajes están firmados..... | 77 |
| 5.1.2 | C2. Los mensajes contienen el identificador del remitente..... | 77 |
| 5.1.3 | C3. Los mensajes están asociados a una partida específica..... | 78 |
| 5.1.4 | C4. Los mensajes del bucle de juego están asociados a una ronda específica..... | 79 |
| 5.1.5 | C5. Los nonces se generan de forma descentralizada..... | 80 |
| 5.1.6 | C6. Los mensajes contiene una marca de tiempo..... | 81 |
| 5.1.7 | C7. Están definidas las condiciones de validez de los mensajes..... | 81 |
| 5.1.8 | C8. Están definidas las acciones para el manejo de errores..... | 81 |
| 5.1.9 | C9. Los parámetros de juego se negocian de forma explícita..... | 82 |
| 5.1.10 | C10. La gestión de claves PGP es externa al protocolo..... | 82 |
| 5.1.11 | C11. La comunicación va sin cifrar..... | 82 |
| 5.2 | Experimento con el prototipo..... | 82 |
| 5.2.1 | Preparación..... | 83 |
| 5.2.2 | Variables de contexto que intervienen..... | 84 |
| 5.2.3 | Ejecución..... | 84 |
| 5.2.4 | Resultados..... | 85 |
| 5.2.4.1 | Mensajes intercambiados..... | 85 |
| 5.2.4.2 | Bytes transmitidos..... | 86 |

| | |
|-----------------------------------------------------------|------------|
| 5.2.4.3 Tiempo de partida..... | 91 |
| 6 Conclusiones..... | 94 |
| 6.1 Conclusiones generales..... | 94 |
| 6.2 Sobre los objetivos..... | 95 |
| 6.2.1 Objetivo General..... | 95 |
| 6.2.2 Objetivos Específicos..... | 95 |
| 6.3 Trabajos futuros..... | 96 |
| 6.4 Reflexión personal..... | 97 |
| Glosario..... | 98 |
| Referencias Bibliográficas..... | 101 |
| Anexo A: Diagramas de estrategias antitrampas..... | 108 |
| Anexo B: Mensajes de partidas..... | 112 |
| Anexo C: Listado de palabras de prueba..... | 122 |

Índice de tablas

| | |
|----------------------------------------------------------------------------------------------------------------------|-----|
| Tabla 2.1: Matriz de dimensiones de seguridad y amenazas..... | 15 |
| Tabla 2.2: Matriz de amenazas genéricas y específicas..... | 26 |
| Tabla 2.3: Matriz de cruce de arquitecturas de red y resguardo del estado de juego..... | 44 |
| Tabla 3.1: Valor de las variables según el nivel de protección..... | 57 |
| Tabla 4.1: Cronograma de las fases de implementación del prototipo..... | 71 |
| Tabla 4.2: Mensajes utilizados por cada implementación..... | 73 |
| Tabla 4.3: Secuencia de mensajes utilizada por cada implementación..... | 74 |
| Tabla 5.1: Alineamiento de los controles utilizados con los principios de diseño para protocolos criptográficos..... | 76 |
| Tabla 5.2: Amenazas cubiertas por los controles aplicados..... | 76 |
| Tabla 5.3: Vulnerabilidades remediadas/resueltas por los controles..... | 77 |
| Tabla 5.4: Resumen de nivel de seguridad y clases a utilizar por corrida del experimento..... | 85 |
| Tabla 5.5: Resumen de mensajes intercambiados..... | 85 |
| Tabla 5.6: Resumen de bytes transmitidos..... | 87 |
| Tabla 5.7: Resumen de tamaño de mensaje (bytes/mensaje)..... | 88 |
| Tabla 5.8: Impacto de la firma en el tamaño promedio de los mensajes..... | 90 |
| Tabla 5.9: Resumen tiempo de partida..... | 91 |
| Tabla B.1: Mensajes de la partida "execute" (95), corrida 0..... | 112 |
| Tabla B.2: Mensajes de la partida "execute" (95), corrida 1..... | 113 |
| Tabla B.3: Mensajes de la partida "execute" (95), corrida 2..... | 113 |
| Tabla B.4: Mensajes de la partida "execute" (95), corrida 3..... | 115 |
| Tabla B.5: Mensajes de la partida "execute" (95), corrida 4..... | 118 |

Índice de figuras

| | |
|-------------------------------------------------------------------------------------------------------|-----|
| Figura 1.1: Vista general del marco de trabajo para investigación de Design Science..... | 7 |
| Figura 2.1: Bola de nieve..... | 10 |
| Figura 2.2: Diagrama de flujo <i>del</i> juego "El Ahorcado"..... | 11 |
| Figura 2.3: Arquitectura de seguridad para sistemas de comunicaciones extremo a extremo... | 13 |
| Figura 2.4: Arquitectura de seguridad reducida..... | 23 |
| Figura 2.5: Diagrama de adaptación de procedimiento antitrampas actual..... | 35 |
| Figura 2.6: Nivel de protección mínimo..... | 37 |
| Figura 2.7: Nivel de protección intermedio..... | 38 |
| Figura 2.8: Nivel de protección máximo..... | 39 |
| Figura 3.1: Diagrama de secuencia de mensajes para la generación de compromisos sobre la palabra..... | 59 |
| Figura 3.2: Diagrama de secuencia de mensajes para el bucle de juego..... | 62 |
| Figura 4.1: Diagrama de clases resumido..... | 71 |
| Figura 5.1: Detalle de mensajes intercambiados..... | 86 |
| Figura 5.2: Distribución por partida de los bytes transmitidos..... | 87 |
| Figura 5.3: Mensaje play_ack de la corrida 0 palabra 95 (execute)..... | 89 |
| Figura 5.4: Mensaje play_ack de partida corrida 3, palabra 95 (execute)..... | 89 |
| Figura 5.5: Impacto de la firma en el tamaño promedio de los mensajes..... | 90 |
| Figura 5.6: Distribución de tiempo de partida..... | 92 |
| Figura 5.7: Diagrama de caja tiempo de partida corridas 0 y 1..... | 92 |
| Figura 5.8: Diagrama de caja de tiempo de partida corridas 2, 3, y 4..... | 93 |
| Figura A.1: Estado de juego duplicado y arquitectura descentralizada..... | 108 |
| <i>Figura</i> A.2: Estado de Juego duplicado y ofuscado con arquitectura descentralizada..... | 109 |
| Figura A.3: Estado de Juego único y arquitectura cliente-servidor centralizada..... | 110 |
| <i>Figura</i> A.4: Estado de Juego único y arquitectura cliente-servidor descentralizada..... | 111 |
| Figura A.5: Estado <i>de</i> Juego dividido y arquitectura completamente descentralizada..... | 111 |

1 Introducción

En este capítulo se explica el problema y cómo se aborda en este trabajo. Primero se ofrece el contexto sobre los videojuegos y la importancia de que éstos sean resistentes a trampas. En particular, se justifica la búsqueda de soluciones que no dependan de un tercero de confianza. En segundo lugar, se definen los juegos de adivinanza, y sus elementos principales. Terceramente, se describen las características y el propósito del protocolo desarrollado en este trabajo. Seguidamente, se presentan los objetivos de este trabajo, y su alcance. A continuación, se listan las herramientas y los métodos utilizados en este trabajo. Finalmente, se presenta la estructura de este documento.

1.1 Antecedentes y motivación

Hoy en día Internet está pasando por un proceso de centralización (Tabora, 2018). Si bien se podría decir que esta tendencia comenzó con la consolidación a nivel de infraestructura de los ISPs y otros proveedores de servicios en línea, los verdaderos gigantes de internet han surgido junto con la tendencia de computación en la nube. En la medida en la cual los gigantes de Internet se consolidan, crece su dominio sobre la información que reciben los cibernautas. Esto significa un riesgo de pérdida de privacidad para las personas, ya que una misma empresa puede tener acceso a los contactos de una persona, sus comunicaciones personales, y sus intereses. Esta información luego es monetizada por estas compañías facilitando que los contenidos puedan ser dirigidos específicamente a cada usuario. Las aplicaciones de esta capacidad varían desde seleccionar los anuncios publicitarios con mayor probabilidad de que el usuario les haga clic, enviar contenido a los usuarios que los mantenga la mayor cantidad de tiempo frente a la pantalla, o personalizar los mensaje de forma que resulten más persuasivos para cada usuario.

Esto ha hecho que los silos de información vuelvan blancos cada vez más apetitosos. Por un lado, han sido víctimas directamente de ataques de robo masivo de datos de los usuarios (Armerding, 2018), incluyendo en algunos casos información o fotografías íntimas (McCormick, 2014). Por otra parte, han surgido industrias dedicadas a la manipulación de los algoritmos de estos servicios. Así, se tiene la optimización de contenidos para motores de búsqueda o SEO, del inglés *Search Engine Optimization*; los seguidores falsos en cuentas sociales para inflar la percepción de popularidad, y los perfiles falsos humanos o bots para posicionar (u ocultar) temas, o narrativas. Así, hoy en día cobra cada vez más relevancia respecto a las noticias falsas o Fake News. En particular, numerosos grupos considerados radicales o extremistas, reclutan a sus seguidores haciendo uso de estas facilidades. Otras organizaciones, utilizan

estos servicios como plataformas para llevar a cabo sus propias campañas de recolección masiva de datos, como ocurrió con Cambridge Analítica (Griffin, 2018).

La respuesta a este problema igual tiene su costo para los usuarios. Estas plataformas, sea por iniciativa propia, o amparadas por los Estados, tienen la posibilidad de excluir de la discusión pública segmentos completos de la sociedad. En algunos casos es sutil, y se baja la probabilidad de que ciertos contenidos sean presentados a los usuarios (Bond, 2018). En otros casos, se toma una resolución más fuerte y los contenidos son bloqueados o eliminados (Litpak, 2019). Finalmente, en algunos casos se prohíbe por completo la discusión de ciertos tópicos o se cierran las cuentas de los usuarios involucrados (Chappel & Tsiulcas, 2018) (Martineau, 2019). Estas prohibiciones pueden abarcar incluso países completos como es el caso de Irán, Cuba, o Corea del Norte.

En resumen, estos procesos ofrecen pocos recursos de apelación y transparencia a los usuarios. Esto resulta en una asimetría enorme entre estos conglomerados y sus usuarios. En relación a estos eventos, Tim Berners-Lee, creador de la web, manifestó sentirse devastado (Brooker, 2018). Además, se lamentó del control que tienen los grandes conglomerados de Internet sobre la información que reciben los cibernautas, y la pérdida de privacidad que ha significado para las personas. Por ello, argumenta que la centralización progresiva de la web atenta contra los intereses de la humanidad. Así mismo, indicó que actualmente está trabajando en un proyecto para “re-decentralizar la web”.

Este trabajo se enmarca en este mismo espíritu que la propuesta Tim Berners-Lee, de promover las soluciones descentralizadas que no tengan un punto único de falla, ni concentren tanta información. Para ello, se enfoca en una clase específica de videojuegos.

Se escogió este ámbito porque actualmente, los videojuegos además de ser un medio de entretenimiento, son una forma importante de socialización. Según la Entertainment Software Association (2016), en Estados Unidos, los videojuegos tienen una penetración en los hogares de un 63%, y la edad promedio de los jugadores es 35 años. De la misma forma, indica que la mayoría de los jugadores juegan en modo multijugador, y lo consideran una forma importante de compartir con sus familiares y amigos. En este sentido, Latinoamérica no se queda atrás, ya que tomada en conjunto, la región es el quinto mayor mercado de videojuegos del mundo, detrás de China, Estados Unidos, Japón y Corea (El Mercurio, 2018). Dentro de Latinoamérica, Chile se posiciona como un importante mercado (Montes, 2019), donde un 45% de la población juega algún videojuego (Quintero, 2018), y con una de las principales industrias de videojuegos de iberoamérica (Publimetro, 2018).

Las trampas siempre han sido una debilidad de los videojuegos (Yan & Randell, 2009). Pero en el contexto de los videojuegos masivos multijugador, la prevalencia de trampas ahuyenta a los jugadores honestos del juego (Yan, 2003). Esto las convierte en un obstáculo a la socialización. Por eso la calidad de los videojuegos se mide por su desempeño, y su resistencia a las trampas (Prather, Nix & Jessup; 2017).

En este sentido, la integridad en los videojuegos está cobrando mayor importancia. En 2016, varios adolescentes fueron arrestados en Japón por hacer trampa en juegos en línea (Geigner, 2016a), Corea del Sur aprobó una ley que criminaliza la elaboración de software que facilite hacer trampa (Geigner, 2016b), China aprobó una ley que obliga a las empresas de videojuegos a revelar las probabilidades de que ciertos ítems aparezcan en los *loot boxes*¹ o cajas de botín (Gartenberg, 2017).

Las estrategias de prevención de trampas han variado progresivamente. En un inicio, los esfuerzos se centraron en ofuscar los datos en el cliente para evitar el acceso indebido a la información (Yan, 2003). La debilidad de este enfoque proviene de que los datos están en el computador local. Un usuario suficientemente motivado y capacitado puede utilizar las técnicas apropiadas para revelar esta información. Posteriormente, se intentó sacar de los clientes el estado general del juego y dejarlo solamente en el servidor (Yan, 2003). Este servidor quedaba encargado de recibir las jugadas de cada cliente, consolidarlas, y luego enviar a los clientes solamente la información a la que debían acceder. Sin embargo, esta estrategia también tiene desventajas. Gran parte de la investigación en este área se ha concentrado en el alto consumo de recursos del servidor, y la poca escalabilidad de esta solución (Fan, Trinder & Taylor, 2010). Por esta razón, hoy se están desarrollando formas de evitar pasar los datos al servidor, buscando descentralizar los datos en los clientes. No obstante, las arquitecturas descentralizadas se siguen considerando más vulnerables a las trampas que las centralizadas (Yahyavi & Kemme; 2013). Sin embargo, hoy la visión del servidor como un bastión seguro está obsoleta. Además de las implicaciones que negativas que un enfoque centralizado puede tener para los usuarios, se debe tomar en cuenta también que su viabilidad está afectada por ser los juegos en línea son el mayor blanco de ataques de denegación (Hickey, 2017).

1.2 Descripción del problema

Los juegos de adivinanza son aquellos cuyo objetivo es adivinar un Secreto. Para esto, los jugadores pueden tomar dos roles posibles: Seleccionador y Adivinador. El Seleccionador escoge el secreto y responde las preguntas del Adivinador. El Adivinador juega haciendo

¹ Ítem que al ser abierto revela aleatoriamente otros ítems. En algunos casos, el juego puede revelar cierta información probabilista. Su venta ha sido señalada como una forma de apuesta (Perks, 2016)

preguntas o intentos que ayudan a esclarecer el secreto, hasta que descubre el secreto (victoria) o se queda sin jugadas (derrota). Uno de los ejemplos más comunes es “El Ahorcado”, el cual se detalla en el capítulo dos. Otros juegos de adivinanza incluyen juegos de mesa comerciales como “Adivina quién?” (Hasbro Inc, 2015a) o *Battleship* (Hasbro Inc, 2015b), en los cuales, ambos jugadores ejercen alternadamente de Seleccionadores y Adivinadores en una misma partida.

Las trampas son aquellas acciones que dan una ventaja al jugador e infringen las normas del juego. Para Neumann, Prigent, Varvello & Suh (2007) los tipos de trampas se corresponden con los principios de la triada de la Seguridad de la Información (confidencialidad, integridad, y disponibilidad). Se afecta la confidencialidad cuando el Adivinador obtiene información que no le corresponde tener sobre el secreto. Se afecta la integridad cuando el Seleccionador cambia el Secreto durante la partida. Se altera la disponibilidad cuando alguno de los jugadores busca retrasar o colapsar la partida para no perder. Adicionalmente, las trampas pueden afectar la autenticidad de la comunicación, cuando alguien emite una jugada falsa a nombre de otro jugador, o hace un *replay attack*, es decir, reenvía una jugada válida anterior (Yahyavi & Kemme, 2013)

¿Cómo prevenir las trampas en videojuegos de adivinanza sin recurrir a un tercero de confianza? ¿Cuál es el desempeño de una solución de este tipo?

1.3 Solución propuesta

1.3.1 Características de la solución

- Debe permitir que el Adivinador verifique que el juego fue justo. Es decir, que se mantuvo la integridad del Secreto, y que la respuesta a sus intentos fue apropiada.
- No debe entregar al Adivinador ninguna información adicional sobre el Secreto. Esto quiere decir, que se mantuvo confidencialidad del Secreto, y no se alteró la dificultad de adivinarlo.
- Debe preservar la estructura descentralizada del juego. Esto significa que no se debe depender de un tercero que opere como custodio del Secreto o como certificador de las jugadas.
- Debe ser resistente a clientes maliciosos. Esto quiere decir, que la seguridad se encuentra en el protocolo, y no depende de la implementación del cliente adverso.
- La comunicación no debe ser vulnerable a impostores. Los intentos del Adivinador y las respuestas del Seleccionador deben estar autenticados.

- Debe servir para comunicar clientes de juego a través de una red.

1.3.2 Propósito de la solución

Permitir que las personas puedan participar en juegos de adivinanza, con la confianza de que se están respetando las reglas del juego, sin que haga falta un tercero de confianza que lo certifique.

1.4 Objetivos y alcances del proyecto

1.4.1 Objetivo general

OG. Diseñar un protocolo resistente a trampas descentralizado para juegos de adivinanza en redes.

1.4.2 Objetivos específicos

OE1. Sintetizar las dificultades específicas asociadas llevar a cabo juegos de adivinanzas de forma descentralizada.

OE2. Definir los requisitos que debe cumplir un protocolo para que sea resistente a trampas y descentralizado.

OE3. Diseñar y desarrollar un protocolo para juegos de adivinanza que cumpla con los requisitos definidos.

OE4. Demostrar la funcionalidad del protocolo mediante la implementación de un prototipo de juego de adivinanza.

OE5. Evaluar el desempeño del protocolo mediante la repetición de experimentos controlados con el prototipo.

1.4.3 Alcances

1.4.3.1 Alcance

El protocolo a desarrollar debe proteger la integridad y confidencialidad del Secreto desde que es seleccionado hasta que culmine la partida, la autenticidad de las jugadas durante la partida, y la validez de las respuestas del Seleccionador.

1.4.3.2 Limitaciones

La solución cubre las capas de sesión, presentación, y aplicación del Modelo OSI. Se presume que los protocolos en las capas inferiores estén orientados a conexión y se encargarán de

transmitir de forma confiable los datos entre los clientes. En este sentido, las capas inferiores quedan fuera del alcance de este trabajo. En consecuencia, la solución no evita trampas que dependan del protocolo de red, o del tipo de conexión, ni ataques de denegación de servicio contra la infraestructura de red. Por otro lado, el protocolo no abarca la interacción humana. Por consiguiente, no puede impedir que los jugadores se coludan, extraigan información de su oponente mediante ingeniería social, o gestionen de forma incorrecta sus claves criptográficas.

Para este trabajo, no sólo se da por sentado que el jugador puede alterar el cliente, sino que no se considera necesaria la existencia de un cliente “original”. Por ser sistemas abiertos, para que dos clientes se conecten, sólo es necesario que ambos implementen versiones compatibles del protocolo. Por lo tanto, no se puede deducir nada adicional sobre el funcionamiento interno de los mismos. Esto quiere decir que no se puede detectar ni prevenir que alguno de los clientes preste asistencia avanzada al jugador como análisis estadístico, optimización de jugadas, o cualquier otro servicio que pueda resultar ventajoso. Más aún, la solución tampoco permite distinguir entre un usuario humano y un aplicación automatizada (o *bot*).

1.5 Metodologías y herramientas utilizadas

1.5.1 Metodología de desarrollo

Para Johannesson & Perjons (2014) Design Science (ciencia del diseño) es el estudio de los artefactos creados para resolver problemas prácticos. Este método fue concebido para la creación de sistemas informáticos, pero puede ser aplicado para trabajos de investigación. Los autores detallan un marco de trabajo cuyas actividades son Explicar el Problema, Definir Requisitos, Diseñar y Desarrollar el Artefacto, Demostrar el Artefacto, y Evaluar el Artefacto. Para este trabajo, el artefacto se trata de un protocolo resistente a trampas descentralizado para juegos de adivinanza en redes. En la Figura 1 Vista general del marco de trabajo para investigación de Design Science, se pueden apreciar estas actividades.

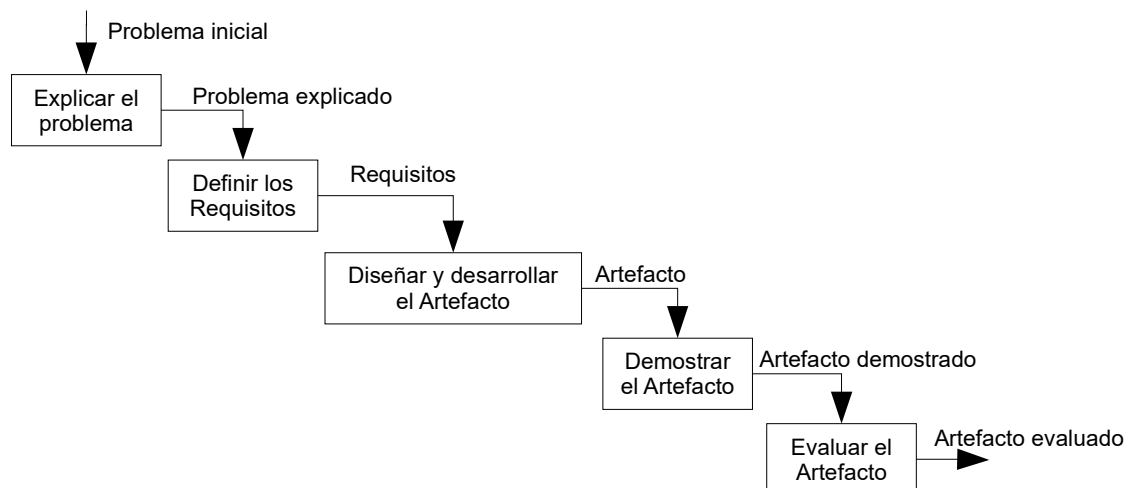


Figura 1.1: Vista general del marco de trabajo para investigación de Design Science.

Fuente: Johannesson & Perjons (2014)

1.5.1.1 Explicar el problema

Se analiza el problema inicial, sus causas y consecuencias, con la finalidad de obtener un problema explicado (Johannesson & Perjons, 2014). Para ello se llevará a cabo una revisión de la literatura relevante. Esta revisión se hará utilizando el método Bola de Nieve (Wohlin, 2014).

1.5.1.2 Definir los requisitos

En esta etapa se transforma el problema en necesidades que debe satisfacer la solución. Los requisitos pueden ser funcionales, estructurales, y ambientales (Johannesson & Perjons, 2014). Los requisitos que debe cumplir el protocolo se obtendrán de la revisión de la literatura.

1.5.1.3 Diseñar y desarrollar el artefacto

A través del diseño se determina la estructura y funcionalidad del artefacto, y luego se crea un artefacto que cumpla con los requisitos (Johannesson & Perjons, 2014). En esta etapa se desarrollará el protocolo siguiendo principios reconocidos para el diseño de protocolos de comunicación. Abadi & Needham (1995) establecen once principios que se deben tener en cuenta para el diseño de protocolos criptográficos.

1.5.1.4 Demostrar el artefacto

En esta fase se realiza una “prueba de concepto” del artefacto para confirmar que funciona, aplicándolo en una situación real (Johannesson & Perjons, 2014). En este orden de ideas, se demostrará la factibilidad del protocolo mediante un prototipo. Se utilizará la metodología de Desarrollo Rápido de Aplicaciones (McConnell, 1996) para implementar el protocolo en un cliente para jugar un juego de adivinanza.

1.5.1.5 *Evaluar el artefacto*

Se mide el cumplimiento de los requisitos, y qué tan bien se resuelve el problema (Johannesson & Perjons, 2014). Se evaluará la funcionalidad del protocolo mediante la repetición de experimentos controlados con el prototipo. Wohlin, Runeson, Höst, Ohlsson, Regnell, & Wesslén (2012) establecen que los pasos para efectuar un experimento son definir el alcance, planificar, ejecutar, analizar e interpretar, y presentar y empaquetar. Estos pasos se aplicarán de la siguiente forma:

1. Definir el alcance: El propósito del experimento es comprobar si el protocolo propuesto permite que dos jugadores desarrollen una partida de un juego de adivinanza. Se desea medir el número de partidas desarrolladas a completitud, el tiempo que toman, y el tráfico de red generado.
2. Planificar: El experimento se realizará utilizando bots que implementen el protocolo. Se definirá un listado de palabras y se monitoreará la evolución de las partidas con dichas palabras. Este experimento se realizará con un computador personal.
3. Ejecutar: Se registrarán los tiempos de duración de las partidas, y se capturará el tráfico de red generado. Se validará que haya congruencia entre los datos recolectados.
4. Analizar e interpretar: Se efectuará un análisis estadístico sobre los datos recopilados.
5. Presentar y empacar: Se documentarán los resultados obtenidos y las conclusiones alcanzadas.

Por otro lado, la evaluación de seguridad del protocolo se realizará de forma teórica, utilizando como apoyo la literatura de este área. En particular, se verificará que se sigan los once principios de diseño de Abadi & Needham (1995) mencionados previamente.

1.5.2 *Herramientas de desarrollo*

Para la elaboración de este trabajo se utilizaron las siguientes herramientas:

- Hardware: computadores portátiles personales
- Sistema operativo: Windows 7 o Ubuntu 14.04 LTS.
- Herramientas de investigación: Google Scholar y los recursos de la Biblioteca Virtual de la USACH.
- Suite ofimática: LibreOffice versión 6.
- Diagramación: Bizagi Modeler versión 3.2 o LibreOffice.

- Desarrollo: Python versión 3, Notepad++ o Gedit.

1.5.3 Ambiente de desarrollo

Este trabajo se desarrollará en el domicilio del autor, y en el Departamento de Informática de la USACH.

1.6 Organización del documento

Esta tesis contiene seis capítulos. En el primer capítulo se presenta una vista general del problema y de las posibles soluciones. De la misma forma, se establecen los objetivos de este trabajo y los métodos a utilizar. Seguidamente, en el segundo capítulo se resumen los conceptos necesarios para comprender este trabajo. A continuación, en el capítulo tres se enumeran los requisitos que debe cumplir el protocolo diseñado, los principios que se deben seguir para su diseño, y el diseño resultante. Luego, en el cuarto capítulo se detalla el prototipo que implementa el protocolo, incluyendo la funcionalidad implementada, el plan de trabajo para el desarrollo, la arquitectura del prototipo, y una reseña sobre el resultado final. Posteriormente, en el capítulo cinco se incluye el resultado de la evaluación teórica del protocolo, y del experimento con el prototipo. En el capítulo 6 se encuentran las conclusiones de este trabajo. Más adelante, en las secciones finales se encuentra el glosario de los términos especializados utilizados en este trabajo, se listan las referencias utilizadas para elaborar este trabajo, y se incluyen los apéndices.

2 Marco teórico

2.1 Método de investigación

Para esta investigación, se buscaron artículos relevantes utilizando el método bola de nieve. Wohlin (2014) define un procedimiento para llevar a cabo este método. El primer paso de este procedimiento es inicializar la bibliografía con un conjunto semilla de artículos relevantes. Seguidamente, se buscan de forma iterativa nuevos artículos, relacionados con la bibliografía que ya se tiene, utilizando las referencias. Cuando se buscan fuentes que son citadas por algún trabajo que ya está en la bibliografía, se habla de bola de nieve de retroceso. Por el contrario, cuando se buscan fuentes que citan trabajos que están en la bibliografía, se denomina bola de nieve de avanzada. En la Figura 2.1 Bola de nieve, se puede observar un diagrama sobre estos dos actividades.



Figura 2.1: Bola de nieve

Fuente: Elaboración Propia, 2018.

Para inicializar la bibliografía, se utilizó Google Scholar², y se combinaron términos de búsqueda de tres categorías distintas: aquellos relacionados con los juegos de adivinanza, otros con la arquitectura de los videojuegos, y otros relacionados con seguridad de la información. Los términos relacionados con juegos de adivinanza incluyen “guessing game”, “hangman”, “battleship”, y “guess who”. Por otra parte, los términos relacionados a los videojuegos incluyen “multiplayer game”, “network game”, “peer-to-peer”, y “p2p”. Finalmente, los términos relacionados a la seguridad incluyen “secure”, “pgp”, “signature”, y “zero-knowledge proof”. Una vez iniciada la búsqueda se descubrió la relevancia de otros términos dentro de este campo como “cheat-proof”, “cheat”, “cheat detection”, y “commitment scheme”.

2.2 Juego “El Ahorcado”

El Ahorcado es un juego de adivinanza de dos jugadores. Uno de los jugadores toma el rol de Seleccionador y el otro jugador toma el rol de Adivinador. El Seleccionador escoge una palabra

2 <https://scholar.google.com>

como Secreto, y le indica al Adivinador cuántas letras tiene dicha palabra. El Adivinador tiene una cierta cantidad máxima de intentos fallidos para adivinar el Secreto. Un intento del adivinador consiste en preguntar al Seleccionador si el Secreto contiene una determinada letra. El Adivinador intenta adivinar las letras del Secreto hasta que acierte todas las letras del Secreto o agote su máximo de intentos fallidos. Si la respuesta es afirmativa, el Seleccionador debe indicar qué posiciones en la palabra ocupa dicha letra. En cambio, si la respuesta es negativa, el Seleccionador lo indica al Adivinador y le resta uno de sus posibles intentos fallidos o “vidas”. La forma tradicional de llevar la cuenta de cuántas vidas le quedan al Adivinador es ir dibujando con cada intento fallido el patíbulo u horca, y luego las partes del cuerpo ahorcado. Cuando el patíbulo y el cuerpo están completos, se dice que el Adivinador perdió o está “ahorcado” (Sweigart, 2015/2008). En la Figura 2.2 se puede apreciar un diagrama de flujo de alto nivel para el juego “El Ahorcado”.

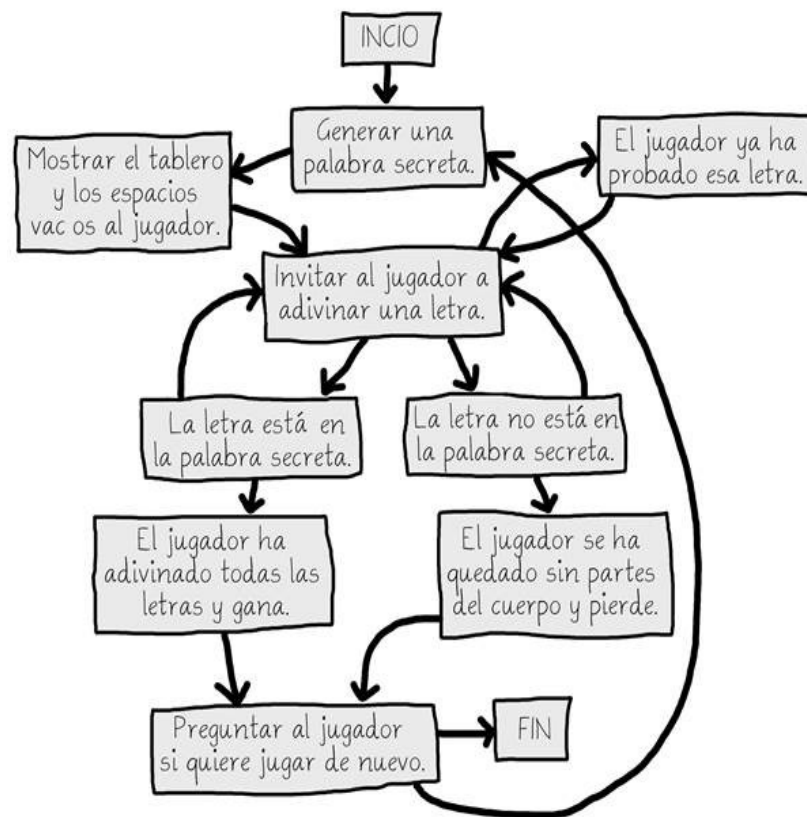


Figura 2.2: Diagrama de flujo del juego "El Ahorcado".

Fuente: Sweigart (2015/2008)

2.3 Principios de diseño

A continuación, se listan once principios que se deben tener en cuenta para el diseño de protocolos criptográficos (Abadi & Needham, 1995), los cuales servirán como buenas prácticas de referencia en este trabajo:

- P1. "Todo mensaje debe decir lo que quiere decir: la interpretación del mensaje debe depender solamente de su contenido. Debería ser posible escribir una oración directa, en lenguaje natural³, describiendo su contenido – aunque si existe un formalismo disponible, también es válido".
- P2. "Las condiciones para que un mensaje sea accionable deben estar definidas claramente, de forma que cualquiera que revise un diseño sepa si son válidos o no".
- P3. "Si la identidad de un participante es esencial para el significado de un mensaje, resulta prudente mencionar explícitamente al participante dentro del mensaje".
- P4. "Debe estar claro para qué se usa el cifrado. Cifrar no es totalmente económico, y no preguntarse para qué precisamente se está usando puede ocasionar redundancia. Cifrar no es sinónimo de seguridad, y su mal uso puede provocar errores".
- P5. "Cuando un participante firma algo que ya fue cifrado, no se debe inferir que dicho participante conoce el contenido del mensaje. Por otro lado, es apropiado inferir que el participante que firma un mensaje y luego lo cifra por privacidad, conoce el contenido del mensaje".
- P6. "Debe estar claro cuáles propiedades se están suponiendo sobre los *nonces*⁴. Algo que sirva para asegurar la sucesión temporal podría no servir para asegurar la asociación – y quizás es mejor establecer la asociación de otra forma".
- P7. "El uso de una cantidad predecible (como el valor de un contador) puede servir para garantizar la novedad, a través de un intercambio de desafío y respuesta. Pero para que una cantidad predecible sea efectiva, debería estar protegida de forma que un intruso no pueda simular un desafío y luego repetir la respuesta".
- P8. "Si las marcas de tiempo se usan como garantía de frescura por referencia al tiempo absoluto, entonces la diferencia de hora local entre los relojes de varias máquinas debe ser mucho menor que la antigüedad permitida para que un mensaje se considere válido. Además, el mecanismo de mantenimiento de la fecha y hora en todas partes pasa a ser parte de la base de computación confiable".
- P9. "Una clave puede haber sido usada recientemente, por ejemplo para cifrar un *nonce*, y aún así ser muy antigua, y estar posiblemente comprometida. El uso reciente de una clave no hacen que ésta luzca mejor de lo que lo haría de no ser así".
- P10. "Si se usa una codificación para presentar el significado del mensaje, entonces debería ser posible determinar cuál codificación está siendo utilizada. En el caso común

3 Nota del traductor: el texto original dice "una oración directa en inglés".

4 Nonce: Número único que se utiliza una sola vez en un protocolo, y que no debe repetirse (Mollin, 2007)

donde la codificación depende del protocolo, debería ser posible deducir que el mensaje pertenece a este protocolo, y de hecho a una ejecución particular del protocolo, y saber su número en el protocolo”.

P11. “El diseñador del protocolo debería saber de cuáles relaciones de confianza depende su protocolo, y por qué la dependencia es necesaria. Las razones para que ciertas relaciones de confianza sean aceptables deberían ser explícitas aunque estén basadas en un criterio o una política en lugar de la lógica”.

2.4 Arquitectura de Seguridad

La UIT (2003) elaboró una arquitectura de seguridad para sistemas de comunicación extremo a extremo. Su propósito es establecer los equipos y actividades de red a proteger, las amenazas que los afectan, y el tipo de protección que requieren. Esta arquitectura está estructurada en tres componentes: dimensiones, capas, y planos. En la Figura 2.3 Arquitectura de seguridad para sistemas de comunicaciones extremo a extremo, se puede observar la relación entre estos componentes. Adicionalmente, se considerarán las vulnerabilidades, amenazas y ataques relevantes al problema estudiado.

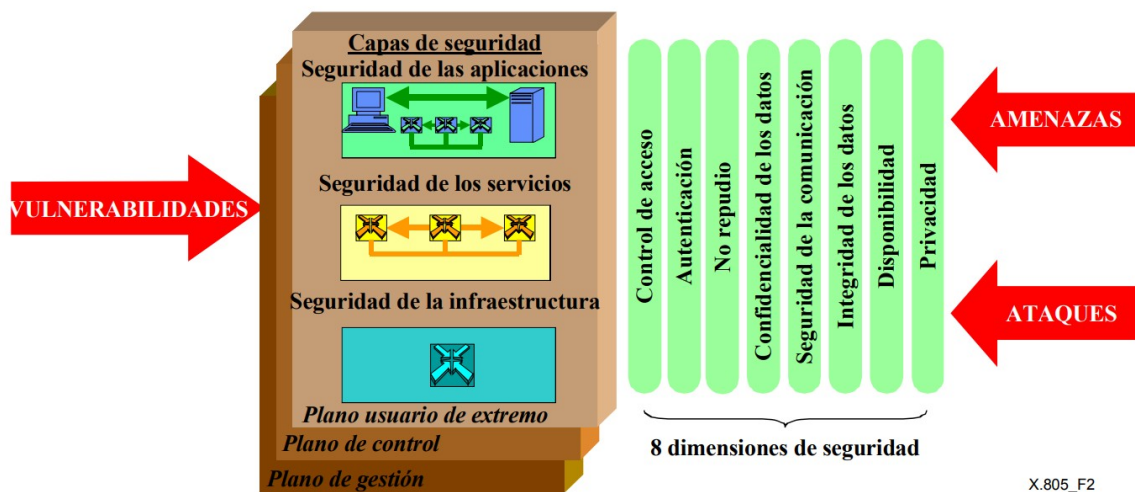


Figura 2.3: Arquitectura de seguridad para sistemas de comunicaciones extremo a extremo.

Fuente: UIT (2003)

2.4.1 Dimensiones de Seguridad

La UIT (2003) define una dimensión de seguridad como “un conjunto de medidas de seguridad que responden a un determinado aspecto de la seguridad de red”. Así mismo, lista ocho de estos conjuntos los cuales son:

1. Control de acceso: Evita que se utilicen recursos de red sin autorización.
2. Autenticación: Identifica a los participantes en la comunicación.

3. No repudio: Evita que alguien realice una acción y luego lo niegue.
4. Confidencialidad de los datos: Evita que quienes no estén autorizados accedan al contenido de los datos.
5. Seguridad de la comunicación: Asegura que la comunicación sólo transita por los puntos extremos autorizados.
6. Integridad de los datos: Resguarda los datos de la creación, modificación, borrado, o reactuación indebidos.
7. Disponibilidad: Permite el acceso a los elementos de red cuando sean requeridos. Prevé la recuperación en caso de incidentes.
8. Privacidad: Protege la información que puede ser descubierta espionando el tráfico de red.

2.4.2 Capas de seguridad

La UIT (2003) define una jerarquía de tres capas de seguridad, donde la capa inferior “potencia” a las capas superiores. Estas capas no deben confundirse con las capas del modelo OSI. A continuación, se detallan las tres capas de seguridad de la arquitectura, de abajo hacia arriba:

1. Seguridad de la Infraestructura: “comprende los dispositivos de transmisión de la red y los elementos de red que están protegidos por las dimensiones de seguridad”. “Los componentes de esta capa son, por ejemplo, los encaminadores, los centros de conmutación y los servidores, así como los enlaces de comunicación entre estos encaminadores, centros de conmutación y servidores.”
2. Seguridad de los servicios: “La capa de seguridad servicios se utiliza para proteger a los proveedores de servicio y a sus clientes, que están expuestos unos y otros a amenazas contra la seguridad.”
3. Seguridad de aplicaciones: “tiene que ver con la seguridad de las aplicaciones de red a las que acceden los clientes de proveedores de servicios”. “Hay cuatro objetivos de ataques contra la seguridad en esta capa: el usuario de la aplicación, el proveedor de la aplicación, los programas intermedios de terceros que intervienen como integradores (por ejemplo, servicios de albergue en la web) y el proveedor del servicio”.

2.4.3 Planos de seguridad

La UIT (2003) recomienda que se separen los eventos correspondientes a actividades distintas de red. De esta forma, si un servicio recibe una alta demanda de los usuarios, esto no debe

impedir que los administradores puedan gestionar el servicio. Para ello, establece los tres planos asociados a las actividades que se pueden realizar. Los planos de seguridad son:

1. Gestión: protege las funciones de Operaciones, administración, mantenimiento y configuración.
2. Control: Protege las actividades que permiten conmutar o encaminar de forma eficiente los datos en la red, distribuir la carga, u otro tipo de señalización. Generalmente se trata de comunicaciones de máquina a máquina.
3. Usuario de extremo: Se trata de la seguridad de los clientes que se conectan a la red y la utilizan.

2.4.4 Amenazas

Es definida por la Asociación Española de Normalización y Certificación (2014) como la “causa potencial de un incidente no deseado, el cual puede ocasionar daño a un sistema o a una organización”. La UIT (2003) también indica la relación entre estas dimensiones de seguridad y una lista de amenazas de seguridad. En la Tabla 2.1 Matriz de dimensiones de seguridad y amenazas, se muestran estas relaciones.

Tabla 2.1: Matriz de dimensiones de seguridad y amenazas

| | | Amenazas de Seguridad | | | | |
|--------------------------|------------------------------|--------------------------------------------------|------------------------------------------|----------------------------------------------------------------|---------------------------|---------------------------|
| | | Destrucción de información y/o de otros recursos | Corrupción o modificación de información | Robo, supresión o pérdida de información y/o de otros recursos | Revelación de información | Interrupción de servicios |
| Dimensiones de seguridad | Control de acceso | ✓ | ✓ | ✓ | ✓ | |
| | Autenticación | | | ✓ | ✓ | |
| | No repudio | ✓ | ✓ | ✓ | ✓ | ✓ |
| | Confidencialidad de datos | | | ✓ | ✓ | |
| | Seguridad de la comunicación | | | ✓ | ✓ | |
| | Integridad de los datos | ✓ | ✓ | | | |
| | Disponibilidad | ✓ | | | | ✓ |
| | Privacidad | | | | ✓ | |

Fuente: UIT (2003)

2.4.5 Vulnerabilidades

Es definida por la Asociación Española de Normalización y Certificación (2014) como una “debilidad de un activo o control que puede ser explotado por una o más amenazas”,

2.4.6 Ataques

Es definida por la Asociación Española de Normalización y Certificación (2014) como una “tentativa de destruir, exponer, alterar, robar o acceder sin autorización o hacer uso no autorizado de un activo”.

2.5 Conceptos criptográficos relevantes

2.5.1 Hash

Una función *hash* toma una entrada de largo variable y devuelve un valor pseudo-aleatorio de largo fijo (Burnett & Paine, 2001). Según Katz & Lindell (2015), estas funciones se usan tradicionalmente para construir tablas *hash*, las cuales permiten la búsqueda de elementos almacenados en tiempo constante. En este contexto de estructuras de datos, si la función *hash* H tiene un rango de tamaño N , entonces el elemento x se almacena en la fila $H(x)$ de una tabla de tamaño N . Se dice que hay una colisión si dos elementos x y x' , son distintos ($x \neq x'$), pero tienen el mismo valor *hash* ($H(x) = H(x')$). En estos casos, la fila correspondiente a $H(x)$ almacena tanto x como x' , con lo cual aumenta el tiempo de búsqueda para dicha fila. Por lo tanto, es deseable que la función *hash* produzca la menor cantidad de colisiones.

En el contexto criptográfico, Katz & Lindell (2015) indican que la resistencia a colisión, más que deseable es un requisito. Esto se debe en gran medida, a que en el dominio de las estructuras de datos, las colisiones se pueden considerar una ocurrencia accidental, mientras que en la criptografía, se parte de la existencia de un adversario cuyo objetivo es causar colisiones intencionalmente. Algunas funciones de *hash* criptográfico notables incluyen MD5, y la familia de funciones SHA (SHA1, SHA2, SHA3).

2.5.1.1 Resistencia a colisiones

Katz & Lindell (2015) explican que para funciones *hash* donde la cardinalidad del dominio es mayor a la del rango, necesariamente deben haber colisiones. Esta observación es consistente con el Principio del Palomar, de Dirichlet. Dicho esto, la resistencia a colisiones se refiere a que dichas colisiones sean difíciles de hallar. En este sentido, se dice que una función *hash* H es resistente a colisiones, si la probabilidad de que un adversario encuentre una colisión en tiempo polinomial es despreciable.

Adicionalmente, Katz & Lindell (2015) indican que existen requisitos de seguridad más débiles que la resistencia a colisiones, los cuales pueden ser suficientes en algunos casos. Se trata de la resistencia a segunda preimagen, y de la resistencia a preimagen. Toda función *hash* H que sea resistente a colisiones es resistente a segunda preimagen, y toda función *hash* H resistente a segunda preimagen es resistente a preimagen. A continuación se brinda más detalle sobre estas propiedades:

- Resistencia a segunda preimagen o a colisión con el objetivo: Si, teniendo x , es infactible para un adversario encontrar un x' tal que $x \neq x'$, y $H(x) = H(x')$.
- Resistencia a preimagen: Si, teniendo y , es infactible para un adversario encontrar un x tal que $H(x) = y$.

2.5.1.2 *Protección de contraseñas con funciones hash*

Katz & Lindell (2015) mencionan que uno de los usos más comunes e importantes de las funciones *hash* es la protección de contraseñas. Es decir, en lugar de almacenar las contraseñas en texto claro, los sistemas aplican una función *hash* y almacenan dicho resultado. Para autenticar a los usuarios posteriormente, una vez que éstos introduzcan una contraseña, el sistema aplica la misma función *hash* sobre el valor introducido, y compara el resultado con el valor almacenado. De esta forma se evita que un atacante, que obtenga acceso a los datos del sistema, pueda recuperar la contraseña original x (ver 2.7.5.2 V2. El compromiso es el Secreto en claro), ya que sólo obtendría $H(x)$, el resultado de aplicar la función *hash* H sobre dicha contraseña. Es importante evitar revelar la contraseñas de los usuarios, ya que esto podría afectar no sólo al sistema en cuestión, sino incluso otros sistemas. Es decir, si un atacante obtiene la contraseña en claro de un usuario, puede acceder indebidamente al sistema atacado, y a cualquier otro sistema donde el usuario esté utilizando la misma contraseña.

Sin embargo, Katz & Lindell (2015) indican que si la contraseña es escogida dentro de un espacio reducido D de posibilidades, entonces un atacante podría encontrar la contraseña x a partir de $H(x)$, en un promedio de $|D|/2$ intentos (la mitad de la cardinalidad de D). Más concretamente, dan como ejemplo que si las posibles contraseñas son palabras en inglés, un atacante puede compilar un diccionario D de palabras en inglés ($|D| \approx 80.000$), y hallar una preimagen en un promedio de aproximadamente 40.000 intentos. Para ello, teniendo $H(x)$, un atacante puede iterar sobre la lista de palabras en D , y por cada $pw_i \in D$ calcular $H(pw_i)$, hasta que encuentre un pw_i tal que $H(pw_i) = H(x)$ (ver 2.7.6.2 Ataque de diccionario). Esta debilidad ocurre porque la propiedad de resistencia a preimagen sólo dice que es difícil invertir un *hash* $H(x)$ cuando x se escoge de forma uniforme sobre un dominio grande (como $\{0,1\}^n$). No dice nada cuando x se escoge de otro espacio o de otra forma.

Si en lugar de una palabra en inglés, el usuario escoge una secuencia alfanumérica (letras mayúsculas, minúsculas, y números) de ocho caracteres, se tienen 62^8 combinaciones posibles (aproximadamente $2,18 \times 10^{14}$). En este caso, un atacante podría generar secuencialmente cada una de las combinaciones, y hallar la preimagen en un promedio de $1,09 \times 10^{14}$ intentos. Para ello, teniendo $H(x)$, un atacante debe iterar sobre las posibles combinaciones, y a cada combinación c , aplicarle la función *hash* H , hasta que encuentre una c tal que $H(c) = H(x)$ (ver 2.7.6.3 Ataque de fuerza bruta). Este ataque requiere muchos más intentos, lo cual se traduce un mayor consumo de cómputo para calcular los valores *hash*.

Sin embargo, el costo de estos ataques podría tomarse como un costo hundido, de forma que cada ataque posterior tenga un costo adicional marginal. Katz & Lindell (2015) explican que el preprocesamiento le permitiría a un adversario invertir incluso funciones aleatorias a una mayor velocidad que haciendo una búsqueda exhaustiva. Para el mismo ejemplo de las contraseñas alfanuméricas de 8 caracteres, un atacante puede generar con antelación las 62^8 combinaciones distintas, aplicarles la función *hash* H , y almacenar en cada renglón de una tabla la combinación utilizada y el valor obtenido. Luego, cada vez que el atacante obtenga una lista de *hash* de las contraseñas de los usuarios de un servidor, basta con que compare éstos contra la tabla preprocesada, para obtener una preimagen (ver 2.7.6.1 Ataque con tabla arco iris). Esta operación de búsqueda es mucho menos costosa que la de cómputo de valores *hash*.

Para mitigar estos riesgos, Katz & Lindell (2015) afirman que se utilizan principalmente dos estrategias. La primera consiste en utilizar funciones *hash* más lentas, ya esto afecta desproporcionadamente a los atacantes. Por ejemplo, sea H_1 una función *hash* que demora un microsegundo en calcularse, y H_2 otra función *hash* pero que demora un milisegundo en calcularse. Para un usuario legítimo, aunque haya una diferencia de un factor de mil entre ambas funciones *hash*, el impacto entre un microsegundo y un milisegundo no es significativa. Sin embargo, retrasar un ataque por un factor de mil, si resulta efectivo como protección. Por ejemplo, para un atacante generar una tabla arco iris de 62^8 combinaciones con H_2 demoraría unos 218.340.105.585 segundos, o casi 7.020 años, mientras que utilizando H_1 se demoraría sólo 7 años.

El mundo de las criptomonedas⁵ ofrece ejemplos concretos sobre este punto. Esto se debe a que se trata de una comunidad que está particularmente interesada en la velocidad de los cálculos de *hashes*, ya que para usualmente se requiere del cálculo de *hashes* que cumplan ciertas características para generarlas o “minarlas”. Como referencia, utilizando un CPU se pueden calcular dentro del orden de los miles de hash por segundo (Kh/s), mientras que los

5 Algunas criptomonedas notables son Bitcoin (Nakamoto, 2008), Zcash (Bitansky, Canetti, Chiesa & Tromer; 2012), Ethereum (Buterling, 2014), y Monero (van Saberhagen, 2013)

GPU generan millones de hash por segundo (Mh/s), y con hardware especializado o ASIC⁶ se pueden procesar billones de hash por segundo (Th/s) (Techkle, 2018). Esta diferencia en la capacidad de cómputo explica que los mineros de Bitcoin hayan pasado de utilizar CPU a usar GPU, y luego hayan migrado a ASIC. Esto a su vez ha significado que se han concentrado los actores en el ecosistema de Bitcoin, ya que este aumento en el capital necesario para participar, ha marginado a los pequeños actores (99 bitcoins, 2019). Para mantener un ecosistema descentralizado, otras criptomonedas han tenido que migrar a algoritmos *hash* que no tengan soporte de las ASIC, como es el caso de Ethereum (N. V., 2019) y de Monero (Wilmoth, 2018).

La segunda estrategia es el uso de *salt*. Burnett & Paine (2001) lo definen como un valor aleatorio que se concatena a una contraseña antes de aplicarle una función *hash* con la finalidad de evitar ataques de precomputación (tablas arco iris). Katz & Lindell (2015) explican el procedimiento que debe seguir un sistema cuando el usuario ingresa su contraseña *pw* para ser almacenada. En lugar de almacenar directamente $H(pw)$, el sistema debe generar primero un *salt* *s*, consistente en un valor aleatorio grande. Luego con el *salt* *s* genera el *hash* salteado *h_{pw}* a partir de $H(s, pw)$. En el sistema se almacenan los valores de *s* y *h_{pw}*. Cuando un usuario desea autenticarse posteriormente, e ingresa *pw'*, el sistema recupera el *salt* *s* almacenado, y verifica que *h_{pw}* coincida con $H(s, pw')$. Idealmente, se debe utilizar un *salt* distinto para cada contraseña. Como un atacante no puede saber con antelación el *salt* que tendrá cada contraseña, no es factible la precomputación de tablas arco iris. En este caso, si las contraseñas fuesen de 8 caracteres alfanuméricos, un atacante debe realizar un ataque de diccionario o fuerza bruta por cada contraseña que desee descifrar. Es decir por cada combinación *c*, toma el *salt* *s*, y verifica si $H(s, c)$ coincide con *h_{pw}*.

2.5.1.3 Problema de cumpleaños

Katz & Lindell (2015) explican el problema de cumpleaños. Este problema plantea: Si hay *q* personas en una habitación, y los nacimientos se distribuyen de forma uniforme y aleatoria, ¿cuál es la probabilidad de que dos personas cumplan años el mismo día? Por el Principio del Palomar se sabe que con al menos 367 personas, está garantizado que esto ocurra. Pero se puede lograr que la probabilidad de que esto ocurra sea mayor a 50%, con tan sólo 23 personas, ya que $q = \Theta(\sqrt{365})$. Es decir, dada una función criptográfica de *m* bits, su rango tiene una cardinalidad de 2^m , y la probabilidad de hallar colisiones es aproximadamente 50% luego de $2^{m/2}$ intentos. En consecuencia, si se quiere que un algoritmo de hash criptográfico sea resistente a ataques de colisión de *T* intentos, se requiere que el valor hash tenga una longitud de $2 \log T$ bits.

6 Circuito Integrado de Aplicación Específica del inglés *Application-Specific Integrated Circuit*.

Adicionalmente, Katz & Lindell (2015) plantean que estas colisiones no necesariamente corresponden a mensajes significativos. Sin embargo, indican que este ataque puede ser adaptado para hallar colisiones significativas. Por ejemplo, un adversario desea hallar una colisión, para una función hash cuyo valor tiene m bits, y dicha colisión debe ser entre un mensaje halagador x y otro denigrante z . Para ello debe generar una lista X de \sqrt{m} variaciones halagadoras y otra Z de \sqrt{m} variaciones denigrantes. Para generar estas listas se puede ayudar en cambios en la redacción, sinónimos, espaciado, etc. Luego debe buscar si existen $x \in X$ y $z \in Z$, tal que $H(x) = H(z)$.

Este trabajo puede ser facilitado si existen vulnerabilidades en la función *hash* en cuestión. En su tesis de magíster, Stevens (2007) trabaja sobre este problema enfocado en la función *hash* MD5, resultando en tres entregables. El primero es un algoritmo rápido para hallar colisiones sin ninguna propiedad específica. El segundo entregable es un algoritmo que, dados dos mensajes m_1 y m_2 , puede construir dos apéndices b_1 y b_2 tal que $MD5(m_1 || b_1) = MD5(m_2 || b_2)$. Además, entrega una implementación de estos ataques. En un trabajo posterior, se muestra un ataque llamado Nostradamus, donde se generaron 12 archivos PDF distintos, cuyo valor hash es igual (Stevens, Lenstra & de Weger, 2007). Cada uno de los 12 archivos contiene una predicción distinta sobre el ganador de la elección presidencial estadounidense de 2008. Finalmente esta misma línea de investigación resulta en la creación de dos certificados de firma electrónica X.509, para identidades distintas, pero que emiten firmas idénticas (Stevens, Sotirov, Appalbaum, Lenstra, Molnar, Osvik & de Weger, 2009).

2.5.1.4 *Defensa en profundidad*

Cuando discuten los códigos para la autenticación de mensajes basados en hash (HMAC), Katz & Lindell (2015) explican que en lugar de suponer que la función hash, que se usa internamente, es resistente a colisiones, estos mecanismos suponen que dicha función es débilmente resistente a colisiones. Esta suposición significó que se tomaron precauciones adicionales en el algoritmo utilizado. El beneficio de este enfoque se evidenció ya que la función HMAC basada en MD5 siguió siendo robusta, aún cuando ya se conocían vulnerabilidades de MD5. En este sentido, las precauciones adicionales, sirvieron de mitigación mientras los desarrolladores migraron de forma programada las implementaciones de HMAC, a otras que hicieran uso de funciones *hash* más seguras.

2.5.2 *Esquema de compromiso*

Damgård (1999) indica que los esquemas de compromiso permiten que una serie de participantes se comprometan con unos determinados valores, que pueden ser revelados

posteriormente. Estos métodos deben contar con dos propiedades, ser ocultantes⁷ y vinculantes⁸. Es necesario que sean ocultantes, porque la idea es que ningún participante sepa el valor de otro participante antes de que éste lo revele. Y deben ser vinculantes porque tampoco es deseable que algún participante pueda cambiar el valor al cual se comprometió. Además, cuentan con dos fases, la fase de compromiso y la de revelación.

Katz & Lindell (2015) indican que se puede construir un esquema de compromiso a partir de una función hash H . Para comprometerse con un mensaje m , El emisor escoge un número aleatorio uniforme r , y genera un compromiso c aplicando la función hash H , tal que $c = H(m, r)$. Luego puede revelar el compromiso enviando m y r , con lo cual el receptor puede validar que $H(m, r)$ coincide con el c que recibió previamente. Este esquema oculta el valor de m , ya que al ser r uniformemente aleatorio, no se revela nada sobre m , a menos que el atacante calcule específicamente $H(m, r)$. Por otra parte, este esquema es vinculante para el emisor si H es resistente a colisiones.

2.6 Trabajos similares

Baughman et al (2007), explican algunos de los mecanismos utilizados para hacer trampas en los juegos multijugadores. Para esto, distinguen entre los juegos centralizados, donde toda la información de los clientes pasa a través de un servidor antes de llegar a otro cliente; y los juegos descentralizados donde la información es transmitida directamente entre los clientes. A ésta última modalidad también se le conoce como peer-to-peer o p2p. A pesar de que Baughman et al (2007) valoran la coordinación que permiten los servidores, reconocen la escalabilidad de los modelos descentralizados. Aún así consideran que la descentralización aumenta la vulnerabilidad a las trampas. El principal aporte de este trabajo son dos protocolos a prueba de trampas orientados a juegos en tiempo real, uno denominado *Lockstep*, donde todos los jugadores van al unísono; y otro denominado Sincronización Asíncrona (AS en inglés), donde sólo se sincronizan estrechamente las jugadas si hay potencial de interacción. Estos protocolos utilizan estrategias de pruebas de conocimiento cero.

Cronin, Filstrup & Jamin (2003), parten de la premisa de que para ciertos juegos en tiempo real, las interrupciones afectan más la experiencia de juego que las inconsistencias. En particular, hace referencia al género de disparos en primera persona (FPS por sus siglas en inglés). Compara *Lockstep* (Baughman et al, 2007), *Lockstep* Canalizado (Lee, Kozlowski, Lenker & Jamin, 2003), de canal adaptable. En este artículo proponen el protocolo de tubería deslizante.

7 En inglés *hiding*

8 En inglés *binding*

Chambers, Feng, Feng & Saha (2005) elaboran un protocolo para mitigar la revelación de información en juegos de estrategia en tiempo real (RTS por sus siglas en inglés). Este protocolo funciona a través del uso de compromisos. Los autores explican la idea general del protocolo usando como ejemplo el juego *Battleship*. En ese caso, cada uno de los jugadores concatena un *salt* al valor de su propio tablero, una vez están ubicados los barcos. A este valor le aplica una función de *hash* criptográfico, y el resultado lo envía a su adversario como compromiso. Al final del juego, se intercambian los valores en claro del tablero y la *salt*, y comprueban que el *hash* y las jugadas son correctos. Este esquema es aplicado por los autores para imponer el cumplimiento de la niebla de guerra a los demás jugadores.

Pittman & GauthierDickey (2013) elaboran el protocolo Match+Guardian, y una serie de algoritmos que permiten jugar juegos de cartas coleccionables en redes descentralizadas. Una de las características de los juegos de cartas coleccionables que mencionan los autores, y que los diferencian de los videojuegos FPS y RTS, es que el turno de cada jugador no tiene una duración definida, sino que depende de convenciones sociales. La base de su trabajo es la generación segura de números aleatorios, para lo cual se basan en el protocolo de lanzamiento de monedas desarrollado por Blum (1983). En este sentido, definen protocolos seguros para diversas operaciones básicas de estos juegos, como generar un mazo base, crear un mazo de juego, seleccionar una carta al azar dentro del mazo, verificar la legalidad de la carta jugada, y pasar un mazo base a otro jugador. Silva & Simplicio (2017) profundizan en el problema de los juegos de cartas coleccionables y plantean el protocolo SecureTCG que contempla más de dos jugadores, y es menos verboso que Match+Guardian. Una adaptación de estos protocolos a juegos de otros tipos se ve en Rice (2014). En este trabajo se elabora un protocolo seguro para generar aleatoriamente el terreno de los mapas en juegos de estrategia.

Stoughton et al (2014) compara tres implementaciones seguras de *Battleship*. Para ello, se apoya en lenguajes de programación seguros, compatibles con el control de flujo de información (IFC por sus siglas en inglés). Las tres implementaciones se basan en una arquitectura cliente/servidor. La primera implementación usa Concurrent ML y requiere un árbitro confiable. La segunda implementación utiliza Haskell con la biblioteca LIO. La tercera utiliza Concurrent ML con control de acceso para evitar la necesidad de un árbitro confiable.

2.7 Discusión

Se puede estructurar una comparación de los trabajos mencionados anteriormente con los objetivos de esta tesis, utilizando como base la arquitectura de seguridad presentada. En la Figura 2.4 Arquitectura de seguridad reducida, se puede observar los elementos que se consideraron relevantes en este trabajo.

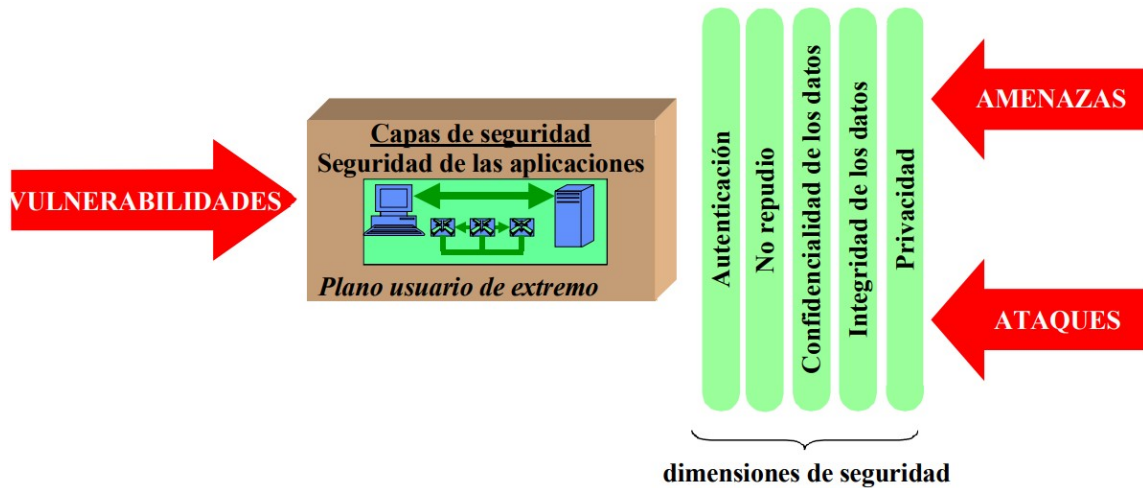


Figura 2.4: Arquitectura de seguridad reducida.

Fuente: Adaptado de UIT (2003)

2.7.1 Dimensiones de seguridad

Ninguno de los protocolos de sincronización revisados se encarga del control de acceso. Por un lado, se considera fuera del alcance de la sincronización de jugadas el control de acceso de los usuarios, es decir la gestión de usuarios registrados, suspendidos, morosos, etc. Por otro lado, el control de acceso de los jugadores a determinados elementos del juego, se debería considerar responsabilidad de la lógica del juego, que valida las movidas. En el contexto de este trabajo, se da por sentado que los jugadores tienen completo control sobre el cliente de juego, y no se contempla la gestión de otra infraestructura. En este sentido, esta dimensión no aplica.

Los juegos en tiempo real (JTR), los juegos de cartas coleccionables (JCC), y los juegos de adivinanza (JA) tienen necesidades similares de autenticación de las jugadas. Es decir, todos se preocupan de alguna forma de evitar que un impostor suplante la identidad de un jugador, falsificando una jugada, o reenviando una movida previamente capturada. En general, se usa una combinación de firma electrónica, *hash* y *nonce*. En los JA, se puede intentar falsificar un intento del Adivinador o una respuesta del Seleccionador.

De la misma forma, en los JTR, JCC, y JA es relevante evitar que los jugadores repudien sus jugadas. En los JA, puede tratarse de que el Adivinador repudie un intento o el Seleccionador repudie una respuesta.

En los JTR es necesario mantener la confidencialidad de todas las jugadas aunque sólo brevemente, ya que la información pierde vigencia. La duración usual es de una "ronda", donde los usuarios se comprometen a una jugada, y luego revelan sus jugadas. En cambio, en los JCC y JA la confidencialidad puede durar toda la partida. Por ejemplo, en los JCC es importante

mantener la confidencialidad del orden y la composición del mazo durante toda la partida, aunque se vayan revelando las cartas jugadas. En el caso de los JA, la jugada importante del Seleccionador es escoger el Secreto, pero el resto de las jugadas no necesitan ser confidenciales. El Adivinador puede enviar sus intentos en claro al Seleccionador, y el Seleccionador también puede responder en claro.

La seguridad de la comunicación no se considera relevante para este trabajo. Por un lado, los protocolos de sincronización de partidas son de capas superiores a la conmutación (capa 2 del Modelo OSI) y el encaminamiento (capa 3 del Modelo OSI). Por otro lado, no resulta necesario para los JA, ya que no se considera que las respuestas y los intentos sean confidenciales, y el secreto permanece en el cliente del Seleccionador, sin transitar.

En lo correspondiente a la integridad de los datos, los requisitos de los JA y JCC son mayores que los de JTR. Baughman et al (2007) se preocupan de garantizar la correctitud del desarrollo del juego, por ello rechazan de plano hacer aproximaciones como navegación por estima, y consideran impráctico hacer vuelta atrás en el estado del juego. Sin embargo, Cronin et al (2003) argumentan que en los JTR pueden hacerse aproximaciones con navegación por estima, u otros ajustes posteriores al estado de juego, a fin de mantener una mayor fluidez en la partida. En los JCC y JA no es viable hacer vuelta atrás de jugadas, ni aproximarlas, ya que esto arruina el juego. En el caso de los JCC no se podría validar si el mazo fue alterado. En los JA no se podría validar si el Seleccionador modificó el Secreto. Esto implica, además, que en un JA no puede corregirse la partida detectando la trampa después del hecho, ya que una vez materializada, se arruina la partida.

La disponibilidad no es un factor crítico en los JA y los JCC, como sí lo es en los JTR. En los JTR los jugadores juegan simultáneamente. Por eso, el mínimo retraso puede afectar la fluidez de la partida. En los JCC y JA las movidas se hacen por turnos, y la duración de cada turno depende de convenciones sociales. Es decir, las prioridades para los JA son contrarias a lo planteado por Cronin et al (2003) para los JTR, ya que en los JA las inconsistencias arruinan el juego, y las interrupciones pueden tolerarse dentro de límites razonables.

Ninguno de los protocolos revisados se preocupa de la privacidad de los datos. La Ley 19.268 (1999) define los datos personales como aquellos “relativos a cualquier información concerniente a personas naturales, identificadas o identificables”, y define los datos sensibles como “aquellos datos personales que se refieren a las características físicas o morales de las personas o a hechos o circunstancias de su vida privada o intimidad, tales como los hábitos personales, el origen racial, las ideologías y opiniones políticas, las creencias o convicciones religiosas, los estados de salud físicos o psíquicos y la vida sexual”. En el caso de los JA no se

considera indispensable el intercambio de datos personales durante la partida, ya que los usuarios podrían identificarse por pseudónimos, y no está previsto el ingreso, transmisión, ni almacenamiento de datos sensibles. Por consiguiente, se considera que esta dimensión no aplica.

2.7.2 Capas de seguridad

Ni los protocolos revisados ni el realizado en este trabajo son responsables de la Seguridad de la Infraestructura, ya que no protegen dispositivos como conmutadores o encaminadores. Por otro lado, los clientes de juego, sean del Seleccionador o del Adivinador, no prestan servicios a otros usuarios, por lo que tampoco aplica la Seguridad de los Servicios. En cambio, al tratarse de aplicaciones que acceden la red, si aplica la capa de Seguridad de Aplicaciones.

2.7.3 Planos de seguridad

Ni los protocolos revisados ni el realizado en este trabajo contemplan la operación, administración, mantenimiento o configuración del juego, por lo tanto no aplica el plano de Gestión. Por otro lado, los mensajes de control que puedan enviar los clientes para mantener la sincronización de los estados de juego tienen una naturaleza distinta a lo planteado para el plano de control de la red, por lo tanto estos mensajes no se consideran parte de este plano. Finalmente, los clientes que implementan los protocolos de juego calzan dentro del plano de Usuario de extremo.

2.7.4 Amenazas contempladas

Al tomar las amenazas genéricas definidas en UIT (2003) (ver sección 2.4.4) y adaptarlas a los JA, se pueden obtener amenazas más específicas. En esta sección se explicará en qué consisten cada una de las siguientes amenazas específicas:

- A1. El Seleccionador pierde el secreto.
- A2. Se pierde una respuesta del Seleccionador.
- A3. Se pierde un intento del Adivinador.
- A4. El Seleccionador altera el Secreto.
- A5. El Seleccionador recibe un intento falsificado.
- A6. El Adivinador recibe una respuesta falsificada.
- A7. El Adivinador repudia un intento.
- A8. El Seleccionador repudia una respuesta.

- A9. El Adivinador obtiene información indebida sobre el Secreto.
- A10. Se rompe la comunicación entre el Adivinador y el Seleccionador.

En la Tabla 2.2 Matriz de amenazas genéricas y específicas, se muestra la relación entre las amenazas genéricas y las específicas a los JA.

Tabla 2.2: Matriz de amenazas genéricas y específicas

| | Amenazas específicas | | | | | | | | | |
|--------------------|----------------------------------------------------------------|----|----|----|----|----|----|----|----|-----|
| | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 |
| Amenazas genéricas | Destrucción de información y/o de otros recursos | ✓ | ✓ | ✓ | | | | | | |
| | Corrupción o modificación de información | | | | ✓ | ✓ | ✓ | ✓ | | |
| | Robo, supresión o pérdida de información y/o de otros recursos | ✓ | ✓ | ✓ | | | | | | |
| | Revelación de información | | | | | | | | ✓ | |
| | Interrupción de servicios | | | | | | | | | ✓ |

Fuente: Elaboración Propia, 2018.

2.7.4.1 A1. El Seleccionador pierde el secreto

El Seleccionador no es capaz de revelar el Secreto porque lo ha perdido u olvidado. Es indistinguible de que el Seleccionador rehúse revelar el Secreto al Adivinador. Es responsabilidad del cliente Seleccionador almacenar seguramente el Secreto, y protegerlo de eliminación no autorizada. Por tratarse de un protocolo descentralizado, no se cuenta con un tercero de confianza que almacene el Secreto en un fideicomiso, por lo cual es irrecuperable.

2.7.4.2 A2. Se pierde una respuesta del Seleccionador

El Seleccionador envía una respuesta al Adivinador, pero éste no la recibe. Deja al Adivinador esperando. Debe ser posible para el Adivinador solicitar al Seleccionador que reenvíe la respuesta a su intento. Es importante que una respuesta reenviada no se interprete como una respuesta posterior.

2.7.4.3 A3. Se pierde un intento del Adivinador

El Adivinador envía un intento al Seleccionador, pero éste no la recibe. Deja al Seleccionador esperando. Debe ser posible para el Seleccionador solicitar al Adivinador que reenvíe su intento. Es importante que un intento reenviado no se interprete como un intento posterior.

2.7.4.4 A4. El Seleccionador altera el Secreto

El Seleccionador responde sobre la base de un Secreto, sino de un conjunto de palabras. Una vez que comienza la partida decide cuáles respuestas resultan más ventajosas a los intentos del Adivinador, con lo cual se van reduciendo las palabras en su conjunto.

2.7.4.5 A5. El Seleccionador recibe un intento falsificado.

El Seleccionador recibe un intento que no fue enviado por el Adivinador. Si el Seleccionador lo toma como válido, el atacante podría forzar al Adivinador a cometer errores, o realizar movidas subóptimas.

2.7.4.6 A6. El Adivinador recibe una respuesta falsificada.

El Adivinador recibe una respuesta que no fue enviada por el Seleccionador. Si el Adivinador la toma como válida, el atacante podría causar que el Adivinador realice intentos sobre respuestas potencialmente falsas, y podría hacer parecer que el Seleccionador envió una respuesta deshonesta.

2.7.4.7 A7. El Adivinador repudia un intento.

El Seleccionador recibe un intento y luego el Adivinador rechaza haberlo realizado. Si el Seleccionador responde antes de recibir el mensaje de repudio, se plantea un problema. Esto se debe a que si el Seleccionador acepta el repudio del intento, entonces el Adivinador tendrá información adicional sobre el Secreto gracias a la respuesta enviada (ver A9). Por otro lado, si el Seleccionador rechaza el repudio, el Adivinador tendría que aceptar un intento que repudió (ver A5).

2.7.4.8 A8. El Seleccionador repudia una respuesta.

El Adivinador recibe una respuesta y luego el Seleccionador rechaza haberla realizado. Si el Adivinador envía un intento antes de recibir el mensaje de repudio, se plantea un problema. Esto se debe a que si el Adivinador acepta el repudio de la respuesta, entonces deberá asumir el intento que acaba de enviar sobre la base de información potencialmente falsa (ver A6). Por otro lado, si el Adivinador rechaza el repudio, el Seleccionador podría ser acusado de dar una respuesta que repudió (ver A4).

2.7.4.9 A9. El Adivinador obtiene información indebida sobre el Secreto

El Adivinador obtiene información sobre el Secreto que no debería tener. Puede tratarse de una revelación total o parcial del Secreto, o de información adicional que le permita reducir el espacio de intentos a considerar, o los posibles valores del Secreto. Por ejemplo: la palabra

completa, una letra y posición, una letra sin posición, la cantidad de vocales en la palabra, la existencia de letras repetidas, etc.

2.7.4.10 A10. Se rompe la comunicación entre el Adivinador y el Seleccionador.

No hay comunicación de red entre el Adivinador y el Seleccionador. Puede ser que al menos uno de los dos pierda conexión a la red, o que no se puedan encaminar los paquetes entre las redes donde están conectados.

2.7.5 Vulnerabilidades conocidas

En esta sección se explican ciertas vulnerabilidades que pueden afectar implementaciones de JA. Estas vulnerabilidades son consecuencia de la ausencia de controles, o la aplicación de controles defectuosos. A continuación se listan:

- V1. El Seleccionador no se compromete a un valor del Secreto
- V2. El compromiso es el Secreto en claro
- V3. El compromiso revela información sobre el Secreto.
- V4. El compromiso es un *hash* sin *salt* o el *salt* es conocido
- V5. El compromiso es una firma sin *nonce* o el *nonce* es conocido
- V6. El algoritmo de *hash* o firma es vulnerable a ataques de colisión
- V7. Los mensajes no están firmados
- V8. Los mensajes no tienen contexto de sesión o partida

2.7.5.1 V1. El Seleccionador no se compromete a un valor del Secreto

Si el Seleccionador no envía al Adivinador un compromiso sobre el valor del Secreto, la partida queda trivialmente expuesta a A4. Esto se debe a que el Seleccionador tiene libertad absoluta para cambiar de Secreto sin que el Adivinador pueda detectarlo.

2.7.5.2 V2. El compromiso es el Secreto en claro

Si el Seleccionador envía al Adivinador, el Secreto en claro como compromiso, se evita A4. Pero queda trivialmente expuesto a A9, ya que el compromiso le revela al Adivinador el Secreto. En los enfoques de “seguridad a través de la oscuridad” se confía en que el cliente del Adivinador ofusca esta información. Para en este trabajo se considera inapropiado utilizar este enfoque, ya que se desea que la seguridad esté en el protocolo, no en la implementación de los clientes.

2.7.5.3 V3. *El compromiso revela información sobre el Secreto.*

Si el Seleccionador envía al Adivinador, un compromiso resultante de manipular el Secreto de una forma criptográficamente insegura (como ROT13, base64, sustitución simple, etc), se evita A4. Sin embargo, el Adivinador puede analizar el compromiso y obtener el Secreto, quedando así expuesto el Secreto a A9. El mismo principio aplica a la aplicación inadecuada de las funciones criptográficas, que permita deducir si hay letras repetidas, o alguna otra característica del Secreto.

2.7.5.4 V4. *El compromiso es un hash sin salt o el salt es conocido*

Si el Seleccionador aplica una función *hash* al Secreto, y lo envía al Adivinador como compromiso, se evita A4. Sin embargo, el Secreto queda expuesto a A9, a través de ataques con tablas arco iris⁹ (ver sección 2.7.6.1). Es decir, el Adivinador podría tomar un diccionario de palabras y precalcular el *hash* de cada una de ellas, luego al recibir el *hash* de compromiso sólo tendría que compararlo con las entradas de la tabla para determinar si alguna de ellas es el Secreto. Si no cuenta con una tabla arco iris, o se usó *salt* pero lo conoce el Adivinador, éste puede llevar a cabo un ataque de diccionario (ver sección 2.7.6.2) calculando en el momento el *hash* de cada palabra en el diccionario, concatenada con el *salt* si lo lleva y es conocido. En teoría, el Adivinador también podría llevar a cabo un ataque de fuerza bruta (ver 2.7.6.3), pero éste consume más tiempo y recursos. Para llevarlo a cabo, el Adivinador debe generar todas las combinaciones posibles de letras, aplicarle la función *hash* a cada combinación, concatenada con el *salt* si lo lleva y es conocido, y comparar el resultado con el compromiso recibido.

2.7.5.5 V5. *El compromiso es una firma sin nonce o el nonce es conocido*

Si el Seleccionador firma el Secreto, y envía la firma al Adivinador como compromiso, se evita A4. Sin embargo, el Secreto queda expuesto a A9, a través de ataques de diccionario (ver sección 2.7.6.2). Es decir, el Adivinador podría tomar un diccionario de palabras y verificar secuencialmente si alguna de estas palabras se corresponde con el texto en claro firmado, es decir el Secreto. En teoría, el Adivinador también podría llevar a cabo un ataque de fuerza bruta (ver 2.7.6.3), pero éste consume más tiempo y recursos. Para llevarlo a cabo, el Adivinador debe generar todas las combinaciones posibles de letras, validar la firma con cada combinación y revisar si alguna se corresponde con el texto en claro firmado.

⁹ En inglés: *Rainbow table*.

2.7.5.6 V6. El algoritmo de hash o firma es vulnerable a ataques de colisión

Si el algoritmo de *hash* o firma es vulnerable a ataques de colisión (ver la sección 2.7.6.4), entonces el Secreto está expuesto a A4. Es decir, si H es una función vulnerable de *hash* criptográfico, el Seleccionador puede buscar dos palabras w_1 y w_2 , dos *salts* s_1 y s_2 , y calcular un compromiso c , tal que $w_1 \neq w_2$ y el resumen criptográfico $c = H(s_1, w_1) = H(s_2, w_2)$. Esto le permitiría al Seleccionador enviar un compromiso c que no es vinculante, ya que puede decidir durante la partida si le conviene más w_1 o w_2 . Al final de la partida, el Adivinador recibiría la palabra y el *salt* que hayan convenido más al Seleccionador, y no podría detectar ni demostrar que hubo trampa en la partida.

2.7.5.7 V7. Los mensajes no están firmados

Si el Adivinador no firma sus intentos, es trivial falsificarlos (A5). De la misma forma, si el Seleccionador no firma sus respuestas, es trivial falsificarlas (A6). De no establecerse un algoritmo criptográficamente seguro para la firma, los intentos de detectar falsificaciones pueden resultar en que los mensajes sean repudiados (A7 y A8).

2.7.5.8 V8. Los mensajes no tienen contexto de sesión o partida

Aún estando firmados los mensajes, es posible que la partida quede expuesta a jugadas a interferencia de agentes maliciosos (A5 y A6) y los problemas relacionados de repudio (A7 y A8). Esto se debe a que un Espía puede escuchar los mensajes y compromisos de la partida y reenviarlos dentro de la misma partida, o en partidas subsiguientes, donde son tomados como válidos. Para evitar esto, los mensajes firmados deben incluir datos como el identificador de la partida o el correlativo de jugadas en la partida, de forma que la jugada no sea válida en un futuro. Por otro lado, la falta de contexto puede ocasionar errores de ordenamiento de los mensajes, o que un mensaje reenviado se interprete como mensaje nuevo (A2, A3).

2.7.6 Ataques relevantes

En esta sección se explican los principales ataques que pueden ser efectuados por un atacante para afectar el normal desenvolvimiento de las partidas de JA.

2.7.6.1 Ataque con tabla arco iris

Este ataque permite obtener la preimagen de una función de *hash* criptográfico. Se caracteriza porque el atacante elabora con antelación una tabla arco iris, conocida en inglés como *rainbow table*. Para ello, el Adivinador podría tomar un diccionario de palabras, precomputar el *hash* de cada una de ellas, y almacenar en una tabla tanto la palabra como el valor *hash* resultante.

Luego, si recibe un valor *hash* del Secreto como compromiso, sólo tendría que compararlo con las entradas de la tabla para determinar si alguna de ellas es la preimagen buscada. Según lo explicado en la sección 2.5, la contramedida más común es agregar un *salt* al *hash*.

2.7.6.2 *Ataque de diccionario*

Este ataque permite obtener la preimagen de una función de *hash* criptográfico. En caso de recibir como compromiso un valor *hash* (sin *salt*) del Secreto como compromiso, una alternativa al uso de tablas arcoiris es calcular los valores *hash* se hace durante el ataque, y no previamente. Es decir, el Adivinador podría tomar un diccionario de palabras y calcula secuencialmente el valor *hash* de cada una de ellas, hasta que encuentre una palabra cuyo valor *hash* coincida con el recibido. Si el compromiso fuese una firma criptográfica sobre la palabra en claro, en lugar de un valor *hash* regular, aplica esta misma lógica, ya que el Adivinador podría validar secuencialmente la firma recibida contra un diccionario de palabras hasta encontrar una palabra que sea validada con la firma.

En el caso de que el Adivinador reciba un valor como compromiso un valor *hash* con *salt* del Secreto, el ataque con tabla arcoiris es inviable. Sin embargo, si el atacante conoce el *salt*, puede calcular secuencialmente el valor *hash* de cada palabra incluyendo el *salt* conocido, hasta que encuentre una palabra cuyo valor *hash*, al incluir el *salt* conocido, coincida con el valor *hash* recibido como compromiso. En este sentido, para que el *salt* proteja efectivamente de este ataque al Secreto, el *salt* debe ser desconocido para el Adivinador durante el desarrollo de la partida. El Adivinador podría conocer el *salt* si el Seleccionador lo revela antes de que culmine la partida, o no se selecciona el *salt* usando mecanismos aleatorios robustos.

2.7.6.3 *Ataque de fuerza bruta*

Tal como se explica en la sección 2.5, un ataque de fuerza bruta consume mucho tiempo y recursos. Para llevarlo a cabo, se deben generar secuencialmente todas las combinaciones posibles de preimagen, aplicarle la función *hash* a cada combinación y comparar el resultado con el *hash* a romper. El Adivinador puede utilizar este ataque si se recibe como compromiso un valor *hash*, independientemente de que se utilice o no *salt*, o si el compromiso es una firma criptográfica.

Si para compromiso se usó *salt* o no, lo que cambia es la complejidad del ataque, por el número de combinaciones a evaluar. Como línea base de comparación, se tiene que un diccionario de palabras en inglés podría tener unas 80.000 entradas (8×10^4). Por otra parte, si no se usa *salt* o el *salt* es conocido, la palabra a adivinar tiene ocho letras, y el idioma es inglés, por fuerza bruta se pueden generar 26^8 combinaciones, esto es más de 208 mil millones de combinaciones o

$2,08 \times 10^{11}$. Al introducir *salt*, se deben tener en cuenta los posibles valores que éste puede tomar. En particular, un número de 32 bits tiene 2^{32} posibles valores, es decir, más de 4,29 mil millones ($4,29 \times 10^9$) de combinaciones. En este sentido, las posibles combinaciones para palabras comunes en inglés, utilizando un número de 32 bits como *salt* son $8 \times 10^4 \times 4,29 \times 10^9$, es decir más de $3,43 \times 10^{14}$. Por otro lado, las posibles combinaciones de palabras con alfabeto inglés de ocho letras y un número de 32 bits como *salt* son $2,08 \times 10^{11} \times 4,29 \times 10^9$, es decir más de $8,92 \times 10^{20}$.

2.7.6.4 *Ataque de Colisión*

Según lo visto en la sección 2.5, consiste en buscar dos preimágenes w_1 y w_2 , y dos *salts* s_1 y s_2 , tal que $w_1 \neq w_2$ y el *hash* $h = H(s_1, w_1) = H(s_2, w_2)$. Así, un atacante puede burlar la integridad que ofrece el *hash*, ya que puede alternar entre w_1 y w_2 , sin ser detectado.

Esto le permitiría al Seleccionador enviar un compromiso h que no lo ata, y así tiene la opción durante la partida de decidir si conviene más w_1 o w_2 . Al final de la partida, el Adivinador recibiría la palabra y *salt* que hayan convenido más al Seleccionador, y no podría detectar ni demostrar que hubo trampa en la partida.

2.7.6.5 *Falsificación de mensaje*

Un atacante elabora un mensaje y hace parecer que proviene de uno de los jugadores. De ser necesario, puede falsear la dirección de origen del mensaje, u otros identificadores. Si los mensajes son firmados, requiere la clave privada del jugador suplantado. Se puede utilizar para falsificar intentos del Adivinador, respuestas del Seleccionador, o interferir con el compromiso sobre el Secreto.

2.7.6.6 *Reenvío de mensaje*

También conocido en inglés como *replay attack*. Un atacante captura un mensaje legítimo proveniente de uno de los jugadores, y lo reenvía posteriormente sin modificación. Puede enviarlo para la misma partida o una partida posterior. El ataque se consuma si el receptor interpreta el mensaje reenviado como un mensaje nuevo, y no como uno antiguo.

2.7.6.7 *Saturación de la red*

Un atacante satura el ancho de banda, o capacidad de procesamiento de la red. Al impedir la comunicación entre los jugadores, evita que éstos puedan llevar a cabo la partida. Es conocido comúnmente como ataque de denegación de servicio o DoS. Como ya fue mencionado en la sección 1.1, los juegos en línea son el mayor blanco de ataques de denegación (Hickey, 2017).

2.7.6.8 *Desvío o impedimento de tráfico de red*

Un atacante afecta la conmutación o encaminamiento del tráfico de red. Al bloquear el tráfico de red, o desviarlo para que no llegue al destinatario adecuado, se impide la comunicación entre los jugadores, por lo cual evita que éstos puedan llevar a cabo la partida.

2.7.6.9 *Intercepción de tráfico de red*

Un atacante inspecciona el tráfico de red. Podría ser uno de los jugadores con la finalidad de obtener información que las aplicaciones de juego intercambian por la red, pero que no muestran a los usuarios. Podría ser un tercero, espiando el transcurso de la partida con la finalidad de obtener información sobre la partida, sobre el comportamiento de las aplicaciones, el uso de firmas electrónicas, o sobre los jugadores.

2.7.6.10 *Distintas versiones de un Mensaje*

Un jugador malicioso envía versiones distintas de un mensaje. Un Adivinador podría enviar dos intentos distintos para una misma ronda. Un Seleccionador podría enviar dos respuestas distintas para un mismo intento. Esto afecta la integridad del juego, y se presta para que el jugador malicioso repudie las versiones menos convenientes del mensaje que envió, quedándose con la más conveniente.

2.7.6.11 *Compromiso del cliente/cliente comprometido*

Un jugador malicioso altera una implementación del cliente, implementa un cliente malicioso, intercepta el tráfico de red, o accede a los datos en memoria, con la finalidad de visualizar información que el cliente maneja pero que se confía que le oculte al jugador. En un enfoque de “seguridad a través de la oscuridad” se podría confiar en que el cliente del Adivinador ofusca información sobre el Secreto. Para este trabajo se considera inapropiado utilizar este enfoque, ya que se desea que la seguridad esté en el protocolo, no en la implementación de los clientes.

Un atacante afecta el funcionamiento de una implementación del cliente a fin de generar alguna desventaja al jugador, extraer información, provocar un mal funcionamiento, etc. Es responsabilidad del cliente Seleccionador almacenar seguramente el Secreto, y protegerlo de eliminación no autorizada. Esto queda fuera del alcance del protocolo

2.7.7 *Adaptación de procedimiento antitrampas actual*

A la luz de lo investigado, sólo Chambers et al (2005) y Stoughton et al (2014) tratan sobre la prevención de trampas en juegos de adivinanza. Stoughton et al (2014) se enfoca completamente en la prevención de trampas en partidas de Battleship, pero utiliza arquitecturas cliente/servidor, y la protección depende de módulos de confianza implementados con lenguajes

de programación seguros. Este trabajo no especifica el protocolo utilizado para la comunicación. Chambers et al (2005), en cambio, sí menciona un mecanismo a prueba de trampas para Battleship, en redes descentralizadas. Sin embargo, se trata de una mención pasajera, ya que ese artículo se enfoca en un problema distinto: la niebla de guerra en juegos en tiempo real. En este sentido, este trabajo considera el protocolo mencionado en Chambers et al (2005) como el estado del arte, aunque dicho trabajo no especifica cuáles mensajes se deben transmitir, ni su contenido, ni su secuencia. Para adaptar el mecanismo propuesto por Chambers et al (2005) a un protocolo de juego para “El Ahorcado” se deben sustituir los elementos equivalentes:

- Secreto: se sustituye el tablero de posición de los barcos con la palabra a adivinar.
- Rondas: siguen consistiendo en que el Adivinador hace una movida, y recibe una respuesta del Seleccionador. En Battleship las movidas consisten en lanzar un misil a una casilla determinada, y la respuesta indica si se atinó a un barco o no; en “El Ahorcado” las movidas consisten en decir una letra del abecedario, y la respuesta es revelar cuáles casillas de la palabra contienen dicha letra.
- Roles: en *Battleship* ambos jugadores se alternan como Adivinadores del tablero enemigo y Seleccionadores del tablero propio; en cambio, en “El Ahorcado” los jugadores sólo ejercen un rol durante la partida, o seleccionan la palabra o la adivinan.

El procedimiento adaptado quedaría como:

1. El Seleccionador escoge una palabra w como Secreto.
2. El Seleccionador escoge un número s como *salt*.
3. El Seleccionador genera el compromiso c aplicando una función de *hash* criptográfico H al Secreto y el *salt*. Es decir, $c = H(s, w)$
4. El Seleccionador envía c al Adivinador.
5. Transcurre la partida de “El Ahorcado”.
6. El Seleccionador envía al Adivinador el *salt* s y la palabra en claro w .
7. El Adivinador aplica la misma función de *hash* criptográfico H al Secreto w y *salt* s , y verifica que $c = H(s, w)$.

En la Figura 2.5 Diagrama de adaptación de procedimiento antitrampas actual, se puede observar como la secuencia de acciones durante la partida afecta el estado de juego conocido por cada jugador.



|  Seleccionador | | Adivinador  | |
|-------------------------------------------------------------------------------------------------|----------------------------------------|------------------------------------------------------------------------------------------------|------------------------|
| Conoce | Acción | Acción | Conoce |
| Función <i>hash</i> H. | Inicio de la partida. | | Función <i>hash</i> H. |
| H, w. | Escoge palabra secreta w. | | H. |
| H, w, s. | Escoge <i>salt</i> s. | | |
| H, w, s, c. | Calcula compromiso c. $c = H(s, w)$ | | |
| | Envía c. | | H, c. |
| | Transcurre partida | | H, c, w, s. |
| | Envía w y s. | | |
| | | Verifica que $c = H(s, w)$ | |

Figura 2.5: Diagrama de adaptación de procedimiento antitrampas actual

Fuente: Elaboración Propia, 2018.

2.7.7.1 Debilidades del procedimiento antitrampas actual

Una gran limitación del procedimiento antitrampas actual es que no cubre el transcurso del juego sino sólo el resultado final. Por un lado, esto implica que el Adivinador no tiene forma de saber si las respuestas que ha recibido son correctas hasta que finalice la partida. Pero también significa que no se puede verificar ni la integridad ni la autenticidad de la comunicación durante el juego (V7), es decir de los intentos del Adivinador y de las respuestas del Seleccionador.

Adicionalmente, incluso los mensajes que sí están cubiertos por el protocolo sufren de ciertas vulnerabilidades. Estas vulnerabilidades emanan de la autonomía del Seleccionador para decidir unilateralmente todos los elementos del compromiso c y la ausencia de identificadores de sesión, jugadores, o partida en el mensaje. Por lo cual hay ciertas propiedades que el Adivinador no puede verificar sobre los mensaje que recibe en el paso 4 y 6, tales como:

- Frescura¹⁰: El Adivinador no puede determinar si el compromiso c que recibió se generó, a partir de w y s, hace un instante o hace años. Esto se debe a que el mensaje no contiene un *nonce*, sello de tiempo, o algún mecanismo similar (V8). Esto a su vez, abre la puerta a que el Seleccionador pueda precomputar colisiones (V6) porque no tiene restricción de tiempo para generar el compromiso.

¹⁰ Frescura (del inglés *freshness*): que se generó luego del comienzo de la ejecución actual del protocolo. (Ling & Chen, 2012). Se contrapone a la ranciedad o *staleness*.

- Destinatario: El Adivinador no puede verificar para quién fueron redactados los mensajes recibidos ya que no forma parte del mensaje, no están cifrados con su clave pública, no llevan identificador de sesión, ni usan algún mecanismo similar (V8).
- Remitente: el Adivinador no puede verificar la identidad de quien envía estos mensajes, ya que no incorporan firma electrónica, un identificador de sesión, o algún mecanismo similar (V8).

En este sentido un tercero malicioso puede efectuar ataques de reenvío o falsificación de mensajes (A5, A6), y tanto el Adivinador como el Seleccionador pueden repudiar sus acciones (A7, A8). Dependiendo del algoritmo de *hash*, un Seleccionador con tiempo y recursos puede alterar el secreto (A4) mediante un ataque de colisión. Como no se especifica si el protocolo de juego, no se puede determinar si hay peligro de que los mensajes reenviados se malinterpreten (A2, A3).

2.7.8 Mejoras propuestas

La primera mejora propuesta es incorporar de forma explícita la necesidad de que los mensajes que intercambian los jugadores vayan firmados. Se considera que éste es el punto de partida, ya que la firma electrónica asegura integridad, autenticidad, y no repudio de los mensajes. Es decir, sin esta protección, no se pueden distinguir entre mensajes legítimos y mensajes falsificados, ni tampoco se puede detectar si el mensaje fue modificado por un intermediario. Cualquier otra protección que se establezca, si no se cuenta con firma electrónica, puede ser replicada o falsificada por un atacante.

En segundo lugar, se debe agregar suficiente contexto a los mensajes para evitar errores de repetición, y ataques de reenvío. Estos podrían ocurrir si un mensaje de otra partida anterior es interpretado por el receptor, fuera de contexto, como si fuese un mensaje nuevo. Además, puede ocurrir con mensajes de la misma partida, pero de una ronda o etapa anterior.

Para mitigar el riesgo de que el Seleccionador precompute compromisos ambiguos sobre la palabra, se propone incorporar un control de frescura sobre los compromisos. De esta forma el Adivinador puede tener mayor confianza en que el compromiso se realizó recientemente, lo cual le da una mayor seguridad computacional al esquema de compromiso. Es decir, se limita la capacidad del seleccionador de precomputar colisiones.

Adicionalmente, se propone definir niveles diferenciados de protección antitrampas. Por un lado, está el nivel de protección mínimo, donde no hay verificaciones durante la partida, sino sólo una verificación final. Por otro lado, está el nivel de protección máximo, donde toda jugada es verificada durante la partida. Adicionalmente, se cuenta con un nivel intermedio, donde sólo se

verifican algunas jugadas. No se considera necesario definir niveles adicionales, ya que cualquier opción que tenga más de una verificación final, pero no verifique todas las jugadas, es una variante del nivel intermedio.

Contar con niveles diferenciados, y no un único nivel de protección máxima, permite que el protocolo se pueda adaptar a una mayor cantidad de contextos. De esta forma se ajusta mejor a las diferencias de necesidad de protección, de las características del juego de adivinanza, y de las capacidades de la plataforma donde se implementa el juego. El nivel de protección máximo tiene precondiciones que podrían no ser satisfechas por todos los juegos de adivinanza, o todas sus variantes, en cuyo caso quedan disponibles los niveles mínimo e intermedio (ver sección 2.7.8.4). En otros casos, las limitantes de la plataforma (capacidad de cómputo, ancho de banda, memoria, etc.) podrían hacer que sólo sea viable el nivel de protección mínimo. Finalmente, algunas partidas o competencias podrían necesitar un estándar de protección mayor que otras (por ejemplo: una partida amistosa vs un torneo).

2.7.8.1 Nivel Mínimo: Secreto

El nivel mínimo es una versión fortificada del protocolo de Chambers et al (2005). El Seleccionador se compromete a un Secreto, se desarrolla la partida, y al final revela el compromiso. Para proteger la autenticidad de la partida, se deben firmar todos los mensajes que envíen los jugadores: el compromiso, los intentos, las respuestas, y la revelación final. Por otro lado, se debe agregar suficiente contexto a los mensajes para evitar errores de repetición, ataques de reenvío, y ataques de colisión. Este nivel requiere menos recursos (almacenamiento, cómputo, y ancho de banda).

Para “El Ahorcado”, el Secreto corresponde a la palabra. De esta forma, el Adivinador puede verificar que la partida fue justa, pero sólo una vez que ha culminado. En la Figura 2.6 Nivel de protección mínimo, se observa que el elemento protegido es la palabra completa.



Figura 2.6: Nivel de protección mínimo

Fuente: Adaptado de <https://www.proprofs.com/games/word-games/hangman/>

2.7.8.2 Nivel Intermedio: Posiciones, Elementos, o partes del Secreto

La idea en este nivel es que el Seleccionador se compromete a ciertos valores parciales del Secreto. Las partes del Secreto o elementos a utilizar dependen del juego. El objetivo es poder revelar compromisos durante el transcurso de la partida, para que el Adivinador puede verificar parcialmente que le han jugado limpio. Otra ventaja de este enfoque, es que aumenta la dificultad computacional de los ataques de colisión. Es importante que las partes del Secreto sean independientes entre sí, de forma de que no se aporte información extra al Adivinador. Esta opción utiliza más recursos que el nivel anterior, pero ofrece una protección más detallada.

Para “El Ahorcado” la palabra puede ser dividida en letras. En este sentido el Seleccionador se compromete a cada una de las letras que conforman la palabra. Por lo tanto, para una palabra de cinco letras, se requieren cinco compromisos. Si en un intento el Adivinador falla, el Seleccionador responde negativamente. Por el contrario, si el Adivinador acierta, el Seleccionador responde afirmativamente y revela el compromiso correspondiente. Debe tenerse cuidado de que si una letra se repite en la palabra, los compromisos en cada posición deben ser distintos, de forma de que el Adivinador no sepa con antelación que hay alguna letra repetida. En la Figura 2.7 Nivel de protección intermedio, se observa que los elementos protegidos son las letras de la palabra.

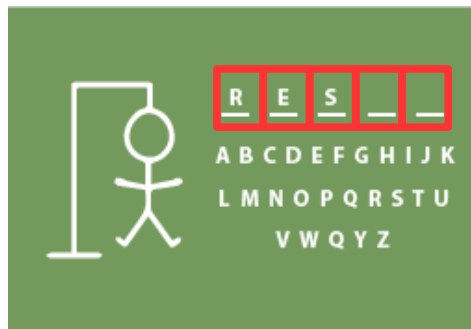


Figura 2.7: Nivel de protección intermedio

Fuente: Adaptado de <https://www.proprofs.com/games/word-games/hangman/>

2.7.8.3 Nivel Máximo: Universo de movidas

La idea en este nivel es que el Seleccionador pueda comprometerse a una determinada respuesta para cada una de las movidas posibles del Adivinador. De esta forma se agrega una protección aún mayor que en nivel anterior, ya que el Adivinador puede verificar el desarrollo justo de la partida durante todas las jugadas, no sólo aquellas que descubren una de las partes del Secreto. Ésta es la opción que consume más recursos de cómputo, pero también es la que ofrece la mayor protección.

Adicionalmente, este nivel de protección tiene como requisito fuerte que las movidas en el juego sean enumerables y finitas. En el caso de “El Ahorcado” el universo de movidas del Adivinador es el abecedario. No hay ningún intento que pueda hacer fuera de ese conjunto de letras. El Seleccionador debe generar veintiséis compromisos si se utiliza el abecedario inglés o veintisiete si se utiliza el abecedario castellano. La respuesta a cada una de estas movidas es una lista con las posiciones que ocupa esa letra en la palabra. Cuando el Adivinador intenta con una letra, el Seleccionador revela la respuesta y el Adivinador verifica inmediatamente si ésta coincide con el compromiso hecho. En la Figura 2.8 Nivel de protección máximo, se ilustra que los compromisos se realizan sobre la respuesta a las letras del abecedario.

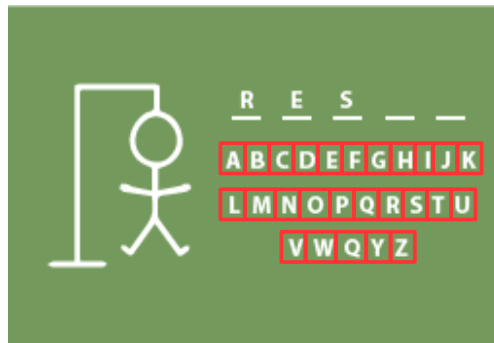


Figura 2.8: Nivel de protección máximo

Fuente: Adaptado de <https://www.proprofs.com/games/word-games/hangman/>

2.7.8.4 Consideraciones sobre el tamaño del universo de movidas

No todas las variantes de JA tienen un universo de movidas enumerable y finito. En particular, en un juego como “20 preguntas” (Wizkid, 1997), el seleccionador escoge un objeto o sustancia, y el adivinador puede hacer un máximo de 20 preguntas cerradas (sí o no) sobre dicho objeto o sustancia, con la finalidad de adivinar de qué se trata. El nivel de protección mínimo se logra trivialmente, generando un compromiso sobre el secreto como un todo. Para lograr un nivel de protección intermedio, el Seleccionador podría generar compromisos sobre ciertas características fundamentales del objeto como: si es animal, vegetal, o mineral; si está vivo o muerto; su color; etc. Sin embargo, no es posible lograr un nivel de protección máximo, porque las preguntas que hace el adivinador son de su libre elección, y no habría como anticipar todas las posibles preguntas para generar un compromiso sobre las respuestas.

En otros casos, se da que el universo de movidas de un JA cambia drásticamente de una variante a otra. Una posible causa es que la respuesta a los intentos, bajo algunas variantes de JA, puede ser producto de operaciones lógicas como conjunción o disyunción sobre las propiedades del Secreto. En este caso, hay que tener presentes los teoremas de De Morgan (Macbeth, Razumiejczyk, Crivello, Bolzán, Pereyra Girardi & Campitelli; 2014): la negación de

una conjunción es la disyunción de las negaciones¹¹, y la negación de la disyunción es la conjunción de las negaciones¹². Esto significa que las respuestas afirmativas y negativas no necesariamente revelan información sobre el Secreto de forma simétrica, debido a sus respectivas reglas lógicas de absorción y de unidad.

Más específicamente, si el intento es una conjunción, y la respuesta es afirmativa, el jugador puede saber que todos los componentes de la conjunción tienen valor verdadero¹³; pero si la respuesta es negativa, sólo puede saber que al menos uno de los componentes de conjunción tiene valor falso¹⁴. Por otra parte, si el intento es una disyunción, pasa lo contrario. En este caso, si la respuesta es afirmativa, el jugador sólo puede saber que al menos uno de los componentes de disyunción tiene valor verdadero¹⁵; pero si la respuesta es negativa puede saber que todos los componentes de la disyunción tienen valor falso¹⁶.

Esto quiere decir que si los compromisos se hacen sobre los posibles componentes de una conjunción, sólo correspondería revelar compromisos cuando la respuesta es verdadera. Como las conjunciones o disyunciones son el verdadero universo de movidas en estos casos, los compromisos deberían hacerse sobre el valor de estas posibles operaciones. El problema radica en que la cantidad de compromisos necesarios podría ser finita, pero impráctica. Esto se puede observar analizando la diferencia de complejidad entre proteger *Battleship*, y algunas de sus variantes.

Battleship es un JA donde cada jugador coloca sus cinco barcos en una cuadrícula de diez casillas de largo por diez casillas de ancho. Sus cinco barcos ocupan un total de 17 de las 100 casillas. El objetivo es hundir los barcos del jugador contrario, indicando las casillas a donde se desea disparar. Los jugadores pueden efectuar un disparo por turno a una de las 100 casillas del tablero enemigo. Si en la casilla que recibió el disparo hay un barco enemigo, el contrincante debe indicar que el disparo acertó y el barco afectado, de lo contrario indica que el disparo falló.

Para el JA *Battleship*, se tienen varias opciones de protección. El nivel mínimo de protección se logra con un compromiso sobre todo el tablero. Para el nivel intermedio de protección, se podrían considerar cinco compromisos, uno por cada barco, que se revelan cuando el barco es hundido. Otra opción intermedia es 17 compromisos, uno por cada casilla tomada por un barco, que se revelan cuando un barco recibe un disparo. El nivel máximo de protección se logra con 100 compromisos, uno por cada una de las 100 casillas del tablero, que se revelan

11 $\neg(p \wedge q) \equiv \neg p \vee \neg q$

12 $\neg(p \vee q) \equiv \neg p \wedge \neg q$

13 $p \wedge \text{Verdadero} \equiv p$

14 $p \wedge \text{Falso} \equiv \text{Falso}$

15 $p \vee \text{Verdadero} \equiv \text{Verdadero}$

16 $p \vee \text{Falso} \equiv p$

independientemente de si la casilla contenía o no un barco. Estas 100 casillas representan el universo completo de movidas.

Una primera variación sencilla de Battleship es que, por cada disparo, el contrincante sólo indique si el disparo acertó o no. Es decir, que no identifique el barco afectado al haber un acierto. Esto agrega la dificultad de que el jugador no sabe si aciertos cercanos pertenecen a un mismo barco, o a barcos distintos. En este caso, el nivel de protección mínimo y máximo se mantienen inalterados. Sin embargo, en el nivel intermedio, la opción de cinco compromisos resulta inadecuada. Esto ocurre, porque le podría revelar información indebida al adivinador, al indicarle cuáles aciertos corresponden al barco hundido, y cuáles algún un barco contiguo aún sin hundir. Por otra parte, la opción para el nivel intermedio de 17 compromisos sí es adecuada, ya que sólo revela una casilla, sin filtrar información sobre el resto del tablero.

Por otra parte, el juego *Battleship* tiene variaciones con disyunciones. “Salvo” (Ultra BoardGames, 2019) es una variación donde el objetivo y la configuración inicial del juego son los mismos que en *Battleship*. La diferencia es que no se efectúa un disparo por jugador por turno, sino una lista de disparos, uno por cada barco vivo del jugador por turno. Es decir, que al principio, con cinco barcos vivos haría cinco disparos por turno; pero más adelante cuando le hundan el primer barco, sólo serían cuatro disparos por turno, y así sucesivamente. En una variación avanzada de Salvo, los intentos no se tratan como una lista de disparos con su respectiva lista de respuestas sobre si acertaron o fallaron, sino que se tratan como una disyunción. Esto implica que se obtiene una respuesta única: o acertó al menos un disparo, o fallaron todos. En caso de algún acierto, se debe informar adicionalmente si fue hundido algún barco propio, ya que esto afecta la cantidad de disparos que puede efectuar el jugador.

La versión regular de Salvo puede ser protegida de la misma forma que *Battleship*, pero la versión avanzada requiere un esquema distinto. Para la versión regular de salvo, se pueden usar para el nivel mínimo un compromiso, para el intermedio 17 compromisos, y para el máximo 100 compromisos. La diferencia es que en cada turno se efectúa una lista de intentos, y se recibe una lista de respuestas. El caso de la variante avanzada es distinto, ya que aunque el nivel de protección mínimo se mantiene como un compromiso sobre todo el tablero, cambia sustantivamente la utilidad de los 17 compromisos sobre casillas de barco, y los 100 compromisos sobre casillas del tablero.

En la versión avanzado de Salvo, es posible que los 17 compromisos de casillas de barco ofrezcan sólo un nivel de protección mínimo (en lugar de intermedio), y los 100 compromisos de tablero ofrezcan sólo un nivel de protección intermedio (en lugar de máximo). En el caso de los 17 compromisos, un compromiso sólo se revela si al jugador que disparó le queda un sólo

barco, ya que al tener más de un barco habría una disyunción entre cuál disparo acertó. Es decir, un jugador que gane, quedándole al menos dos barcos al final de la partida, no recibiría ninguna revelación intermedia. Por otro lado, al utilizar 100 compromisos, no está garantizado que hayan revelaciones todos los turnos. Si el jugador que dispara sólo tiene un barco, hay revelación, independientemente de que acierte o falle, porque no se trata de una disyuntiva. Si le queda más de un barco, sólo hay revelaciones cuando todos los disparos son fallidos. Por lo tanto, la única forma de que un jugador reciba revelaciones todos los turnos es que falle todos sus disparos mientras tenga más de un barco. En este orden de ideas, los 100 compromisos sobre casillas del tablero pueden ser un esquema aceptable de protección intermedia.

Diseñar un nivel de protección máximo para Salvo, tiene un grado mayor de complejidad que los esquemas planteados hasta ahora. Al principio de la partida, un jugador puede disparar a cualesquiera cinco de las 100 casillas del tablero. Entonces se requiere un compromiso para cada una de estas posibilidades. Cuando pierda su primer barco, podrá disparar a cualesquiera cuatro de las 100 casillas del tablero. Entonces también debe generarse un compromiso para cada una de estas posibilidades, ya que no se sabe a priori cuales casillas atacará un jugador cuando tenga cinco barcos, y cuales pospondrá para cuando ya sólo le quedan cuatro vivos. El mismo razonamiento aplica para tres barcos, dos o uno. Esto resulta en un cálculo combinatorio que se puede expresar como:

$$\begin{aligned} & \binom{100}{1} + \binom{100}{2} + \binom{100}{3} + \binom{100}{4} + \binom{100}{5} \\ &= 100 + 4.950 + 161.700 + 3.921.225 + 75.287.520 \\ &= 79.375.495 \end{aligned}$$

Es decir, para garantizar que se revele un compromiso en cada turno de una partida de la variante avanzada de Salvo, se requieren más de 79 millones de compromisos. Aunque es un número contable, y finito, es cinco órdenes de magnitud más grande que 100 compromisos (uno por casilla). Queda de parte de los jugadores e implementadores decidir si desean aplicar este nivel de protección, con el correspondiente costo asociado de tiempo, cómputo, ancho de banda, y espacio; o si el nivel de protección intermedio o mínimo satisfacen sus necesidades de seguridad.

2.8 Enfoques de desarrollo de la solución

Baughman et al (2007) definen un modelo que toma en cuenta dos perspectivas distintas. Estas perspectivas son las capacidades de los jugadores y tramposos, y la arquitectura de resguardo del estado de juego y comunicaciones. Sin embargo estas dos últimas pueden ser separadas entre sí. En el Anexo A se incluyen diagramas de cada uno de estos enfoques.

2.8.1 Habilidades de los jugadores y tramposos

Para Baughman et al (2007), un jugador hace trampa si obtiene una ventaja de leer, modificar, o bloquear el estado de juego de una forma que el cliente original, normalmente no permite. Adicionalmente, supone que el jugador tiene acceso a la misma información que el cliente, descartando así la seguridad a través de la oscuridad. Además restringen su alcance a proteger la capa de aplicación TCP/IP.

2.8.2 Arquitectura de red

Para Baughman et al (2007), los juegos de red pueden utilizar arquitecturas cliente-servidor, o descentralizada (p2p). Sin embargo, trabajos posteriores hacen la distinción entre arquitecturas cliente-servidor centralizadas, y cliente-servidor descentralizadas. En una arquitectura cliente-servidor centralizada, los clientes sólo se comunican con el servidor, así que éste recibe todas las jugadas, y envía todas las actualizaciones. En una arquitectura cliente-servidor descentralizada, el servidor coordina ciertos aspectos del juego, pero los clientes se comunican directamente. En una arquitectura completamente descentralizada, los clientes se comunican directamente las jugadas y las actualizaciones.

2.8.3 Resguardo del estado de juego

Baughman, Liberatore & Levine (2007) definen estado de juego (EJ) como la información necesaria para describir el juego en un instante determinado. Las jugadas ocasionan cambios que deben reflejarse en subsiguientes estados de juego. Resguardar el estado de juego significa mantener su consistencia, aplicar las jugadas, y restringir la información accesible a los jugadores. Para lograr estos objetivos se puede tener el EJ duplicado, único, o dividido. Duplicar el EJ significa que cada jugador tenga su propia copia de éste, sincronice con los demás jugadores los cambios por cada jugada, y se autolimita a sólo revisar la porción del EJ que le corresponde. Mantener una copia única del EJ, implica que un tercero de confianza lo actualice según las jugadas de cada jugador, y les envíe a cada jugador sólo la porción del EJ que les corresponda. Esta estructura es utilizada frecuentemente en las arquitecturas cliente-servidor sean éstas centralizadas o descentralizadas. Finalmente, dividir el EJ quiere decir que cada jugador resguarda su propia porción del mismo, y la sincroniza de forma segura con las porciones de los demás jugadores, de forma que todas las porciones sean consistentes entre sí, sin que exista una copia consolidada del EJ.

2.9 Selección de enfoque de solución y justificación

Las características que debe tener el protocolo son que sea resistente a trampas y que también sea apto para redes descentralizadas. Ninguna de las arquitecturas cliente-servidor califica

porque requieren al menos un servidor. Por lo tanto, la arquitectura de red debe ser completamente descentralizada. En cuanto al estado de juego, si cada cliente conserva un duplicado, no se puede hacer cumplir que sólo accedan a la porción que le corresponde, por lo cual habría una vulnerabilidad a trampas. Por otra parte, si se mantiene una única copia del juego, se pueden prevenir trampas, pero este enfoque es incompatible con una arquitectura completamente descentralizada. En este sentido, el estado de juego debe estar dividido entre los clientes. De esta forma se obtiene que la solución debe ser completamente descentralizada y con el estado de juego dividido. En la Tabla 2.3 Matriz de cruce de arquitecturas de red y resguardo del estado de juego, se puede observar el resumen del cruce de arquitecturas de red y los enfoques de resguardo del estado de juego.

Tabla 2.3: Matriz de cruce de arquitecturas de red y resguardo del estado de juego

| | | Estado de Juego | | |
|--------------|-----------------------------------------|-----------------------------------------|--------------------------------------|-----------------------------------------|
| | | Duplicado | Único | Dividido |
| Arquitectura | Cliente-Servidor Centralizado | Centralizado Vulnerable a Trampas | Centralizado Resistente a trampas | Centralizado Resistente a trampas |
| | Cliente-Servidor Descentralizado | Centralizado Vulnerable a Trampas | Centralizado Resistente a trampas | Centralizado Resistente a trampas |
| | Completamente Descentralizado | Descentralizado Vulnerable a Trampas | Incompatibles | Descentralizado Resistente a trampas |

Fuente: Elaboración Propia, 2018.

3 Diseño del protocolo

Este capítulo contiene los requisitos que debe cumplir el protocolo, así como la especificación del protocolo, con la información necesaria para hacer implementaciones interoperables. La especificación se divide en tres secciones: mensajes, subprotocolos, y cifrado de las comunicaciones. El protocolo incluye los tres niveles de seguridad mencionados en el capítulo anterior: mínimo, medio, y máximo. A grandes rasgos, las partidas cumplen con las mismas fases, independientemente del nivel de seguridad en uso, aunque cada nivel implica pequeñas variaciones en los mensajes y subprotocolos. En la sección relativa a los mensajes se indica los tipos de mensaje que utiliza el protocolo, y se definen los campos que contiene. La secuencia que deben seguir los mensajes durante una partida están indicadas en la sección de subprotocolos. Cada fase de la partida corresponde a un subprotocolo distinto, y también se incluye un subprotocolo para el manejo de errores. Esta modularización disminuye la complejidad del protocolo general, lo cual facilita su análisis e implementación. En la última sección se justifica la decisión de no requerir el uso de cifrado en las comunicaciones.

3.1 Requisitos

El protocolo debe cumplir con ciertos requisitos.

3.1.1 Funcionales

- El Seleccionador debe poder escoger un Secreto, recibir intentos del Adivinador, y dar respuestas.
- El Adivinador debe poder hacer intentos de adivinar el Secreto, y recibir respuestas a sus intentos.
- Los usuarios deben poder escoger o negociar el nivel de protección que quieren utilizar para la partida.
- Los usuarios deben poder jugar en red.

3.1.2 No funcionales

3.1.2.1 Interoperabilidad

- Debe preferir el uso de estándares y tecnologías ya establecidas, al desarrollo de soluciones a la medida, si aquellas pueden satisfacer la necesidad.

3.1.2.2 *Portabilidad*

- El protocolo debe ser implementable en diversas plataformas. No debe limitar, de forma implícita o explícita, las implementaciones a plataformas específicas como un determinado tipo de *hardware*, sistema operativo, bibliotecas de *software*, etc.

3.1.2.3 *Eficiencia*

- El protocolo debe permitir que los implementadores tomen prioridades distintas de eficiencia (uso de memoria, uso de cómputo, fortaleza del algoritmo de cifrado, etc).

3.1.2.4 *Arquitectura*

- El protocolo debe ser descentralizado. Los únicos actores necesarios para el desarrollo de una partida deben ser los jugadores. En este sentido, no se debe requerir la participación de terceros que acrediten o custodien la información de la partida, como las jugadas, el Secreto, la identidad de los jugadores, o la autoría de los mensajes.

3.1.2.5 *Seguridad*

- Debe dar herramientas a las implementaciones para lidiar con clientes maliciosos y atacantes externos.
- Debe ser al menos igual de seguro que el protocolo actual para JA.
- Todos los mensajes de los jugadores deben ir firmados.
- Debe incluir los tres niveles de seguridad definidos en el capítulo anterior.
- El Adivinador debe poder validar que se mantuvo la integridad del Secreto.
- El protocolo no debe entregar información al Adivinador que facilite adivinar el Secreto.

3.1.2.6 *Licenciamiento*

- Las tecnologías que el protocolo incorpore deben estar disponible libremente. Es decir, deben estar disponibles bajo licencias libres, que no limiten el uso comercial.

3.2 Estructura de los Mensajes

Para la firma electrónica se consideró utilizar certificados X.509 (Cooper, Santesson, Farrell, Boeyen, Housley & Polk; 2008) y claves PGP (Callas, Donnerhacke, Finney, Shaw & Thayer; 2007). El requisito de que el protocolo sea descentralizado, hace descartar de plano algunas opciones robustas como requerir Firma Electrónica Avanzada (Ley N.º 19.799, 2002), ya que es un sistema basado en la confianza en un tercero, en este caso autoridades certificadoras. El

mismo razonamiento aplica a requerir certificados X.509 que sean emitidos por fuentes confiables. Si bien un enfoque que permita el uso de certificados X.509 autofirmados, cumple tecnológicamente con el requerimiento de descentralización, en la práctica, la tendencia actual es desconfiar por defecto en los certificados X.509 autofirmados. Por otra parte, el sistema de firmas PGP es eminentemente descentralizado. Desde la generación de claves, pasando por que cada usuario pueda crear su propia red de confianza, a la infraestructura existente en Internet para el intercambio de claves públicas de este tipo. Por esta razón se consideró que PGP es una tecnología idónea para este caso.

Para estructurar los datos dentro de los mensajes se descartó el uso de formatos binarios, o a la medida, ya que dificultaría la extensibilidad e interoperabilidad entre clientes. Esto es un factor importante porque en este trabajo se desea evitar que la seguridad dependa de lo desconocido del formato de comunicación (seguridad por oscuridad), o de la implementación particular que haga un cliente. En este sentido, se consideró como opciones JavaScript Object Notation (JSON) (Bray, 2017) y Extensible Markup Language (XML) (Bray, Paoli, Sperberg-McQueen, Maler & Yergeau, 2008). Ambas opciones son maduras y gozan de amplio soporte multiplataforma. Se decidió utilizar JSON porque es menos verboso que XML.

Una vez seleccionadas las tecnologías JSON y PGP, se buscaron referentes de implementación de JSON con firma electrónica. Lo más cercano que se consiguió fue JSON Web Signature (JWS) (Jones, Bradley & Sakimura, 2015a). JWS cuenta con varios estándares asociados como JSON Web Token (JWT) (Jones, Bradley & Sakimura, 2015b), que utilizan estructuras similares. Estos mensajes agregan una especie de sobre alrededor del contenido que se desea enviar. En el sobre se indica el algoritmo de firma utilizado, el contenido, y el valor desprendido de la firma electrónica. Desgraciadamente, no se consiguieron implementaciones que utilizaran claves PGP, por lo cual se decidió utilizar mensajes PGP en modo ASCII, que pueden ser firmados en modo regular, o firmados en claro, pero no de firma desprendida.

El cuerpo en claro del mensaje, estructurado como un objeto en formato JSON, contiene una serie de campos y sus valores. Existe una serie de campos con metadatos, que son comunes a todos los mensajes, y los campos que contienen información sobre el transcurso de la partida, que son específicos a cada mensaje. Los campos comunes a todos los mensajes son los siguientes:

- `type`: Cadena de texto. Contiene el tipo del mensaje.
- `unix_time`: Entero. Marca de tiempo en “formato Unix” o POSIX, que representa la fecha y hora de emisión del mensaje como segundos transcurridos desde la media noche del 1 de enero de 1970 UTC.

- `my_address`: Cadena de texto. Contiene el campo `address` de la clave PGP del emisor. Identifica al emisor del mensaje.
- `my_session`: Entero. El identificador de sesión del emisor.
- `your_address`: Cadena de texto. Contiene el campo `address` de la clave PGP del receptor. Identifica al receptor del mensaje. El mensaje `lets_play` no contiene este campo, pues el Seleccionador no se ha identificado todavía.
- `your_session`: Entero. El identificador de sesión del receptor. El mensaje `lets_play` no contiene este campo, ya que el Seleccionador todavía no ha informado su identificador de sesión para la nueva partida.

Una vez determinada la estructura, queda pendiente definir como las implementaciones del protocolo pueden delimitar los mensajes, es decir saber cuando se recibió el mensaje completo. Para este fin se consideraron tres estrategias. La primera, una estrategia implícita dinámica, donde las implementaciones deben leer el flujo de datos recibidos y detectar el fin del mensaje recibido en base a su sintaxis. Esta estrategia implica interpretar el fin de los mensajes PGP. La segunda estrategia considerada, es una implícita estática, donde se fija una longitud máxima de los mensajes, y todo mensaje de longitud inferior al máximo se rellena con espacios en blanco para que alcance dicha longitud. Esto significa que los mensajes breves desperdiciarían una gran cantidad de ancho de banda, y se limitaría a futuro la expresividad de nuevos tipos de mensajes. Finalmente, se consideró una estrategia explícita dinámica, donde cada mensaje es antecedido por un entero que indica su longitud. Se seleccionó esta última estrategia, ya que no exige la implementación de analizadores sintácticos para la recepción de mensajes, los mensajes breves no desperdician ancho de banda, y no limita el crecimiento futuro del protocolo.

A continuación, se especifica cada tipo de mensaje, así como sus campos específicos.

3.2.1 Mensaje *lets_play*

Lo envía el Adivinador para comenzar a jugar una partida. Todo mensaje `lets_play`, debe utilizar un identificador de sesión nuevo para iniciar una nueva partida, el cual no puede haber sido utilizado con anterioridad por el Adivinador. Contiene los siguientes campos específicos:

- `supported_lang`: Una lista de cadenas de texto de tres letras. Representa los idiomas en los cuales está dispuesto a jugar el Adivinador utilizando el estándar ISO-639-3.
- `supported_algo`: Una lista de cadenas de texto. Representa los algoritmos de hash que que el Adivinador puede utilizar.

- `supported_security`: Una lista de enteros. Representa los niveles de seguridad que el Adivinador puede aplicar. El nivel mínimo se representa con un 2, el nivel intermedio con un 3, y el nivel máximo con un 4.
- `min_lives`: Entero. Representa el mínimo de intentos con los cuales el Adivinador está dispuesto a jugar.
- `max_lives`: Entero. Representa el máximo de intentos con los cuales el Adivinador está dispuesto a jugar.

3.2.2 *Mensaje accept_play*

Lo envía el Seleccionador para indicar que acepta jugar una partida y los parámetros que esta tendrá. Si está especificado el campo Contiene los siguientes campos:

- `lang`: Una cadena de texto de tres letras. Representa un idioma en el cual tanto el Adivinador como el Seleccionador están dispuestos a jugar, utilizando el estándar ISO-639-3.
- `algo`: Una cadena de texto. Representa un algoritmo hash que tanto el Adivinador como el Seleccionador pueden utilizar.
- `security`: Un entero. Representa un nivel de seguridad que tanto el Adivinador como el Seleccionador pueden utilizar.
- `total_lives`: Un entero. Representa el número total de intentos fallidos que podrá realizar el Adivinador. Debe pertenecer al rango de intentos que especificó el Adivinador, y al rango de intentos que permite el Seleccionador.

3.2.3 *Mensaje reject_play*

Este mensaje lo envía el Seleccionador para indicar no puede aceptar la invitación a jugar una partida, por incompatibilidad de parámetros. Si es factible jugar, pero no desea hacerlo, debe enviar un mensaje quit. Contiene los siguientes campos:

- `supported_lang`: Una lista de cadenas de texto de tres letras. Representa los idiomas en los cuales está dispuesto a jugar el Seleccionador utilizando el estándar ISO-639-3.
- `supported_algo`: Una lista de cadenas de texto. Representa los algoritmos de hash que el Seleccionador puede utilizar.

- `supported_security`: Una lista de enteros. Representa los niveles de seguridad que el Seleccionador puede aplicar. El nivel mínimo se representa con un 2, el nivel intermedio con un 3, y el nivel máximo con un 4.
- `min_lives`: Entero. Representa el mínimo de intentos con los cuales el Seleccionador está dispuesto a jugar.
- `max_lives`: Entero: Representa el máximo de intentos con los cuales el Seleccionador está dispuesto a jugar.

3.2.4 Mensaje *play_ack*

Lo envía el Adivinador para acusar recibo de la aceptación de la partida de juego. No contiene ningún campo adicional.

3.2.5 Mensaje *nonce_start*

Lo envía el Seleccionador para iniciar el proceso de generación distribuida de nonces. Contiene el siguiente campo:

- `length`: Entero. Número de *nonces* que serán generados (J).

3.2.6 Mensaje *r_commit*

Lo envía tanto el Adivinador como el Seleccionador. Sirve para comprometerse con una lista de números aleatorios que luego serán utilizados para generar los *nonces*. Contiene los siguientes campos específicos:

- `length`: Entero. Número de *nonces* que serán generados (J). Coincide con la longitud de `my_random_commitment_vector`.
- `my_random_commitment_vector`: Lista de hexadecimales. Se trata del vector de compromisos de números aleatorios, es decir VRCA si el remitente es el adivinador, y VRCS si el remitente es el Seleccionador.

3.2.7 Mensaje *r_reveal*

Lo envía el Adivinador para revelar los números aleatorios con los cuales se comprometió, para lo cual envía también las *nonces* que utilizó para generar el compromiso. Contiene los siguientes campos específicos:

- `length`: Entero. Número de *nonces* que serán generados (J). Coincide con la longitud de `my_random_vector` y de `my_random_nonce_vector`.

- `my_random_vector`: Lista de Enteros. Se trata del vector de números aleatorios del Adivinador (VRA).
- `my_random_nonce_vector`: Lista de Enteros. Se trata del vector de *nonces* de números aleatorios del Adivinador (VRNA), utilizados para generar VRCA.

3.2.8 *Mensaje w_commit*

Lo envía el Seleccionador para comprometerse con el valor de la palabra secreta W. Contiene los siguientes campos específicos:

- `length`: Entero. Número de *nonces* generados (J). Coincide con la longitud de `my_word_commitment_vector`.
- `my_word_commitment_vector`: Lista de hexadecimales. Se trata del vector de compromisos sobre la palabra VWC.

3.2.9 *Mensaje w_ack*

Lo envía el Adivinador para acusar recibo de `w_commit`. No contiene campos específicos:

3.2.10 *Mensaje make_a_try*

Lo envía el Seleccionador para indicar el estado actual de la partida y solicitar un nuevo intento. Contiene los siguientes campos específicos:

- `status`: Cadena de texto. Tiene el mismo largo que la palabra secreta, muestra las letras que han sido acertadas hasta el momento, y la posición que ocupan dentro de la palabra. Las letras que no han sido acertadas todavía son reemplazadas por “_”.
- `lives_left`: Entero. Intentos fallidos que puede cometer el Adivinador antes de perder.
- `round`: Entero. Ronda actual.

3.2.11 *Mensaje new_try*

Lo envía el Adivinador para hacer un intento. Contiene los siguientes campos específicos:

- `guess`: Cadena de texto de una letra. Representa la letra que el Adivinador quiere intentar.
- `round`: Ronda a la cual aplica el intento.

3.2.12 *Mensaje reply*

Lo envía el Seleccionador para dar respuesta a un intento. Contiene los siguientes campos específicos:

- **result:** Booleano. Resultado del intento del Adivinador.
- **guess:** Cadena de texto de una letra. Representa la letra que el Adivinador intentó.
- **round:** Ronda a la cual aplica el intento.
- **status:** Cadena de texto. Tiene el mismo largo que la palabra secreta, muestra las letras que han sido acertadas hasta el momento, y la posición que ocupan dentro de la palabra. Las letras que no han sido acertadas todavía son reemplazadas por “_”.

En los casos cuando aplica una revelación parcial, incluye los siguientes campos adicionales:

- **partial_length:** Entero. Número de revelaciones que se efectúan en esta respuesta. Coincide con la longitud de **partial_random_vector**, **partial_nonce_vector**, y **partial_index_list**
- **partial_random_vector:** Lista de enteros. Sección de VRS a revelar.
- **partial_random_nonce_vector:** Lista de enteros. Sección de VRNS correspondiente a las revelaciones.
- **partial_index_list:** Lista de enteros. Lista de los índices que ocupa cada elemento de las secciones de VRS y VRNS dentro del vector completo correspondiente, necesaria para seleccionar los valores de VRCS apropiados para la validación.

3.2.13 *Mensaje r_ack*

Lo envía el Adivinador para acusar recibo de una respuesta. Contiene el siguiente campos específicos:

- **round:** ronda de la respuesta a la cual se está haciendo acuse de recibo.

Si en la respuesta hubo una revelación parcial, incluye el siguiente campo adicional:

- **validation:** Booleano. Indica que la validación de la revelación parcial fue exitosa. En caso de que la validación sea fallida, corresponde un mensaje **validation_error**.

3.2.14 *Mensaje game_ended*

Lo envía el Seleccionador. Indica que la partida culminó, así como su resultado. Contiene los siguientes campos específicos:

- **result:** Booleano. Resultado de la partida que indica si el Adivinador ganó.
- **secret:** Cadena de texto. Contiene la palabra secreta en claro.
- **length:** Entero. Número de *nonces* que fueron generados (J). Coincide con la longitud de `my_random_vector` y de `my_random_nonce_vector`.
- **my_random_vector:** Lista de Enteros. Se trata del vector de números aleatorios del Seleccionador (VRS).
- **my_random_nonce_vector:** Lista de Enteros. Se trata del vector de *nonces* de números aleatorios del Seleccionador (VRNS), utilizados para generar VRCS.

3.2.15 *Mensaje result_validation*

Lo envía el Adivinador para indicar el resultado de la validación de la partida. Contiene los siguientes campos específicos:

- **result:** Booleano. Indica que la validación de la revelación final fue exitosa. En caso de que la validación sea fallida, corresponde un mensaje `validation_error`.

3.2.16 *Mensaje result_ack*

Lo envía el Seleccionador para indicar el resultado de validación de la partida. No contiene campos específicos.

3.2.17 *Mensaje validation_error*

Lo puede enviar el Adivinador o el Seleccionador, para indicar que una revelación no es consistente con el compromiso previo, o cualquier otra inconsistencia entre los mensajes recibidos, o con el estado de juego. Contiene los siguientes campos específicos:

- **conflicting_messages:** Lista de mensajes. Contiene los mensajes firmados que son inconsistentes entre sí.

3.2.18 *Mensaje unexpected_message*

Lo puede enviar el Adivinador o el Seleccionador, para indicar que el mensaje que se recibió no puede ser reconciliado con el estado actual del juego. Contiene los siguientes campos específicos:

- **last_sent_message:** Mensaje firmado. Último mensaje enviado por el emisor.
- **last_received_message:** Mensaje firmado. Último mensaje recibido por el emisor.

- `expected_messages`: Lista de cadenas de texto. Contiene los tipos mensajes que está esperando el emisor como respuesta a su último mensaje.

3.2.19 *Mensaje quit*

Lo puede enviar el Adivinador o el Seleccionador en cualquier momento de la partida. No contiene campos específicos.

3.3 **Secuencia de los Mensajes**

En esta sección se explican las secuencias de mensajes que deben producirse durante la partida. Cada una de estas secuencias podría verse como un subprotocolo o miniprotocolo. Durante el desarrollo de una partida, se tiene inicialmente la Negociación de los parámetros de la partida, luego la Generación de compromisos sobre la palabra, seguido del Bucle de juego, y finalmente el Cierre de la partida. Cualquiera de los dos jugadores puede iniciar durante la partida el subprotocolo de Abandono de la partida, para indicar que no desea continuar. Finalmente se tienen dos secuencias de mensajes que afectan de forma transversal: Validación de mensajes y Manejo de errores.

3.3.1 *Negociación de los parámetros de la partida*

El objetivo de este protocolo es determinar los parámetros que regirán la partida a desarrollar. Se compone del siguiente intercambio mensajes:

1. La aplicación del Adivinador envía un mensaje `lets_play`. En este mensaje indica los idiomas que en los cuales está dispuesto a jugar, los algoritmos de funciones *hash* que tiene a su disposición, el rango de intentos que desea utilizar, y los niveles de seguridad que puede aplicar. Este mensaje no contiene identificadores de sesión, ni identificador de usuario del destinatario, pero sí contiene el identificador de usuario del remitente, es decir, del Adivinador y su número de sesión.
2. La aplicación del Seleccionador evalúa los parámetros recibidos en el mensaje `lets_play`. Si el Seleccionador ha visto antes una partida donde el Adivinador use ese número de sesión, y se trata de una partida en curso entre ambos jugadores, debe interpretarlo como un mensaje repetido, y no hacer nada. Si se trata de una partida finalizada, o en curso con otro jugador, debe rechazar ese identificador de sesión enviando un mensaje `validation_error` que incluya el mensaje recibido y los mensajes con los cuales entra en conflicto. Si el Seleccionador no ha visto antes una partida del Adivinador donde use ese número de sesión, y se solapan los parámetros de ambas

aplicaciones, entonces la partida es factible. Es decir, si tienen en común al menos un idioma, un algoritmo de *hash*, una cantidad de intentos, y un nivel de seguridad.

- a. En caso afirmativo, envía un mensaje “accept_play”. Dentro de este mensaje debe indicar el valor que escogió para cada parámetro. Cada uno de estos valores debe pertenecer a la respectiva intersección. La aplicación es libre de escoger cualquiera de los valores que de la intersección, pero es recomendable que seleccione la función hash más robusta disponible, y el nivel de seguridad más alto que esté disponible. El Seleccionador, como remitente, especifica en el mensaje su identificador de usuario, y un identificador de sesión del remitente que no haya utilizado con anterioridad. Además incluye el identificador de usuario y de sesión especificados por el Adivinador en su mensaje lets_play.
 - b. En caso negativo, envía un mensaje “reject_play”. Dentro de este mensaje debe indicar los idiomas, algoritmos *hash*, cantidad de intentos, y niveles de seguridad que hubiese aceptado. El Seleccionador especifica su identificador de usuario como identificador de remitente, y el identificador de usuario y de sesión que el Adivinador especificó en su mensaje lets_play. Este mensaje no lleva identificador de sesión del remitente.
3. La aplicación del Adivinador revisa el mensaje recibido.
- a. Si se aceptó la partida, y el Adivinador conoce de alguna otra partida del Seleccionador que utilice identificador de sesión enviado, debe remitir un mensaje validation_error incluya el mensaje recibido y aquellos mensajes con los cuales entre en conflicto. En caso contrario, debe evaluar los parámetros:
 - i. Si son válidos, envía un mensaje “play_ack” y prosigue a la siguiente fase. El Adivinador incluye el identificador de usuario y el identificador de sesión especificados por el Seleccionador en su mensaje accept_play. En este punto se perfecciona el acuerdo de jugar una partida, la cual está plenamente identificada.
 - ii. Si son inválidos, envía “validation_error”, incluyendo los parámetros esperados y los parámetros recibidos.
 - b. Si se rechazó la partida, envía un mensaje “quit”, y cierra la conexión.

Si se aceptó la partida, ambos jugadores saben el idioma que utilizará la palabra secreta, el nivel seguridad a aplicar a la partida, el número de intentos máximo, y la función de *hash* a

utilizar. Adicionalmente, hasta donde saben ambos jugadores, la partida cuenta con dos identificadores únicos de sesión.

3.3.2 Generación de compromisos sobre la palabra

El objetivo de este protocolo es que el Adivinador reciba ciertas garantías sobre la palabra secreta, de forma que el Seleccionador no pueda cambiarla. Estas garantías se dan en forma de compromisos. El nivel de seguridad de la partida seleccionado anteriormente determina la cantidad de los compromisos y cómo se calculan.

La notación que se utilizará en esta sección es:

- Φ un idioma
- Γ alfabeto utilizado por Φ
- α una letra de Γ
- H una función de *hash* criptográfico
- W una palabra secreta en el idioma Φ compuesta por L letras de Γ
- $\text{longitud}(W) = L$
- $\text{letra}(W, i) = i\text{-ésima letra de } W$.
- $\text{posiciones}(W, \alpha) = \{i : 0 \leq i \leq L : \text{letra}(W, i) = \alpha\}$

A fin de generalizar el algoritmo, se puede considerar que para una palabra secreta W , se calcula un vector de palabra VW de longitud J . Y si se genera un vector de J *nonces* para la palabra VWN , se puede calcular un vector de J compromisos sobre la palabra VWC .

Si se instancia el protocolo propuesto por Chambers et al (2005), se tiene que $J = 1$, ya que dicho protocolo utiliza un sólo compromiso. VW puede ser definido como un singletón que contiene la palabra, VWN como un singletón que contiene un nonce que escoge unilateralmente el Seleccionador, y VWC como un singletón con un compromiso. Es decir:

- $VW = \{W\};$
- $VWN = \{\text{nonce}\}$
- $VWC_i = H(VWN_i, VW_i)$

En la sección 2.7.7, se plantea que este esquema es teóricamente vulnerable a ataques de colisión sobre H . La raíz de este problema es que el Seleccionador puede escoger arbitrariamente el valor del nonce a utilizar en el compromiso de la palabra. Por esta razón, en

este trabajo se propone un protocolo de generación distribuida de nonces, basado en el trabajo de Pittman & GauthierDickey (2013), quienes a su vez se basan en Blum (1983).

El valor de VW y J dependen del nivel de seguridad elegido para la partida:

- Para el nivel mínimo, $VW = \{W\}$, y por lo tanto $J = 1$. Se establece sólo un compromiso, sobre la palabra en su totalidad.
- Para el nivel intermedio, VW contiene cada letra de W, es decir $VW_i = \text{letra}(W, i)$, la i -ésima letra de W. Por lo tanto, J es igual a la longitud W. Se establece un compromiso por la letra en cada posición de la palabra W. Si W tiene letras repetidas, se generan compromisos independientes por cada instancia de la letra.
- Para el nivel máximo, VW contiene la lista de posiciones que ocupa cada letra del alfabeto Γ en W. Es decir $VW_i = \text{posiciones}(W, \Gamma_i)$. Por lo tanto, J es igual a la cantidad de letras del alfabeto Γ . Este valor depende del idioma seleccionado, en el caso del inglés es 26, y del español es 27 (incluyendo la ñ). Se establece un compromiso sobre cuáles posiciones ocupa cada letra dentro de la palabra.

En la Tabla 3.1 Valor de las variables según el nivel de protección, se pueden observar un ejemplo de los valores que toman las variables definidas anteriormente. Este ejemplo se realizó utilizando la palabra “sera”.

Tabla 3.1: Valor de las variables según el nivel de protección

| Variables | Nivel Mínimo | Nivel Intermedio | Nivel Máximo |
|----------------------------------------|--------------------------------------|--------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Idioma (Φ) | Inglés | Inglés | Inglés |
| Alfabeto (Γ) | {“a”, “b”, “c”, ... , “x”, “y”, “z”} | {“a”, “b”, “c”, ... , “x”, “y”, “z”} | {“a”, “b”, “c”, ... , “x”, “y”, “z”} |
| Palabra (W) | “sera” | “sera” | “sera” |
| Longitud (L) | 4 | 4 | 4 |
| Número de Nonces (J) | 1 | 4 | 26 |
| Vector de Palabra (VW) | {“sera”} | {“s”, “e”, “r”, “a”} | { {3}, \emptyset , \emptyset , \emptyset , {1}, \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , {2}, {0}, \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset , \emptyset } |
| Vector de Nonces de Palabra (VWN) | {VWN ₀ } | {VWN ₁ , VWN ₂ , VWN ₃ , VWN ₄ } | {VWN ₀ , ..., VWN ₂₅ } |
| Vector de Compromisos de Palabra (VWC) | {H(VWN ₀ , “sera”)} | {H(VWN ₀ , “s”), H(VWN ₁ , “e”), H(VWN ₂ , “r”), H(VWN ₃ , “a”)} | {H(VWN ₀ , {3}), H(VWN ₁ , \emptyset), ..., H(VWN ₄ , {1}), ..., H(VWN ₁₇ , {2}), ..., H(VWN ₁₈ , {0}), ..., H(VWN ₂₅ , \emptyset) } |

Fuente: Elaboración Propia, 2018.

La secuencia de mensajes intercambiados es la siguiente:

1. El Seleccionador escoge una palabra secreta W , en el idioma acordado. Envía un mensaje "nonce_start" indicando el valor de J , es decir el número de compromisos a generar.
2. El Adivinador genera tres vectores. Primero genera un vector de J números aleatorios VRA y un vector de J nonces $VRNA$. Seguidamente genera un vector de J compromisos sobre $VRCA$, donde $VRCA_i = H(VRNA_i, VRA_i)$, utilizando el algoritmo *hash* acordado. Envía un mensaje "r_commit" con $VRCA$.
3. El Seleccionador genera tres vectores, de forma similar al adivinador. Primero genera un vector de J números aleatorios VRS y un vector de J nonces $VRNS$. Seguidamente genera un vector de J compromisos sobre $VRCS$, donde $VRCS_i = H(VRNS_i, VRS_i)$. Envía un mensaje "r_commit" con $VRCS$.
4. El Adivinador revela su compromiso, para ello envía un mensaje "r_reveal" con VRA y $VRNA$.
5. El Seleccionador valida que $VRCA_i = H(VRNA_i, VRA_i)$.
 - a. En caso afirmativo, el Seleccionador calcula un vector de con los J *nonces* de palabra VWN , donde $VWN_i = XOR(VRA_i, VRS_i)$. Luego calcula un vector con los J compromisos de palabra VWC , donde $VWC_i = H(VWN_i, VW_i)$. Envía un mensaje "w_commit" e incluye VWC .
 - b. En caso negativo, envía un mensaje "validation_error", indicando los valores conflictivos de VRA , $VRNA$, y $VRCA$. Seguidamente, cierra la conexión.
6. El Adivinador recibe "w_commit" y envía como acuse de recibo un mensaje "w_ack".

Al final de la ejecución de este protocolo, ambos jugadores conocen J , $VCRA$, $VCRS$, VRA , $VRNA$, y VWC . Sólo el Seleccionador conoce W , VW , VRS , $VRNS$, y VWN .

En la Figura 3.1 Diagrama de secuencia de mensajes para la generación de compromisos sobre la palabra, se puede apreciar la secuencia de acciones de este subprotocolo, y la evolución del estado de juego de cada jugador.

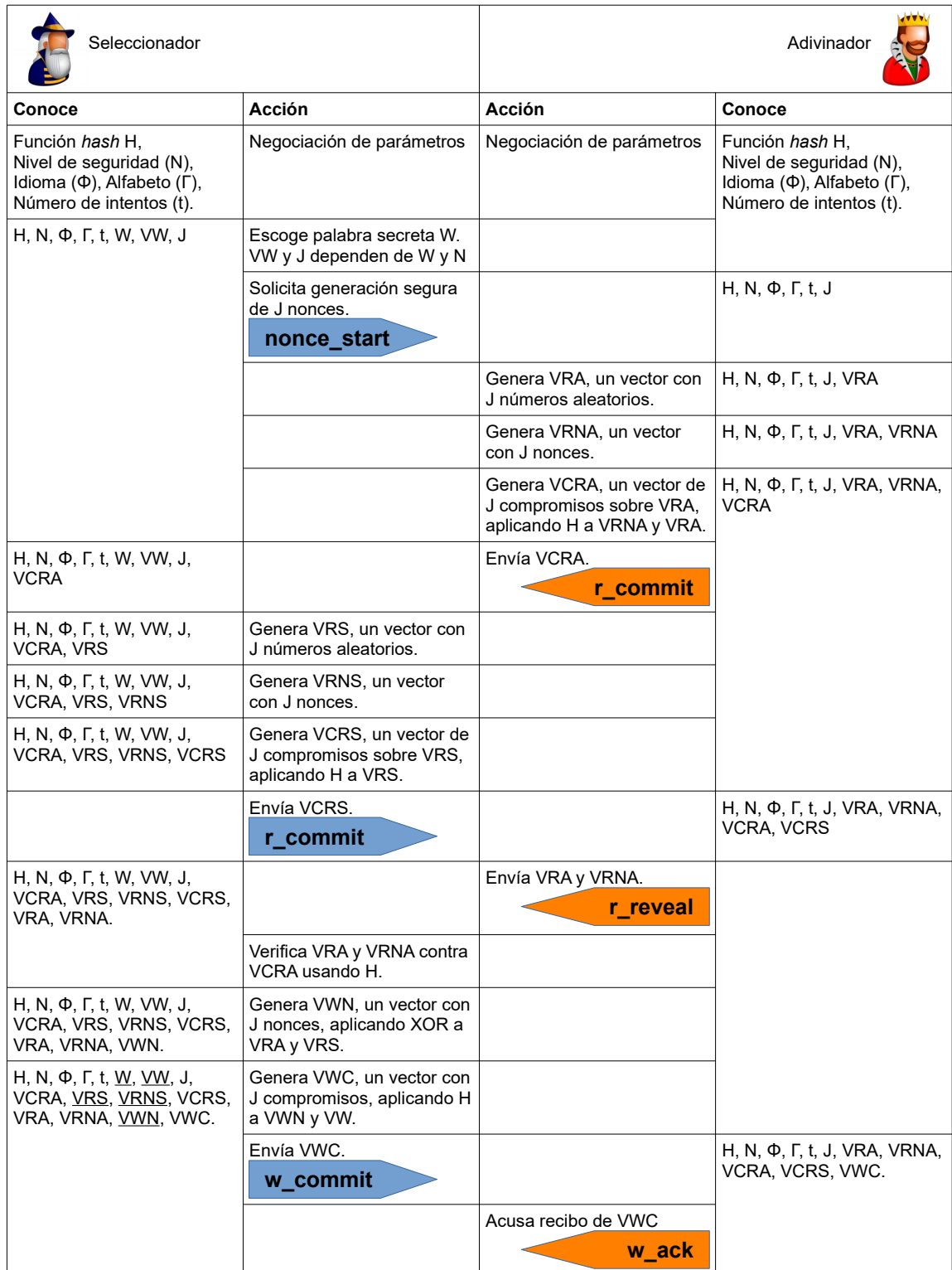


Figura 3.1: Diagrama de secuencia de mensajes para la generación de compromisos sobre la palabra

Fuente: Elaboración Propia, 2018.

3.3.3 Bucle de juego

Este protocolo abarca el desarrollo de las jugadas, toda vez que los protocolos anteriores ya fijaron los parámetros del juego y las formalidades, viene el desarrollo de las jugadas. La secuencia de mensajes utilizada es:

1. El Seleccionador verifica que el Adivinador no ha adivinado la palabra secreta *W* y sigue teniendo intentos.
 - a. En caso afirmativo, envía un mensaje “make_a_try” indicando lo que lleva adivinado de la palabra, el número de ronda, y cuántos intentos quedan.
 - b. En caso negativo, pasa a la siguiente fase.
2. El Adivinador valida que los valores incluidos en “make_a_try” son consistentes con el estado de juego que maneja.
 - a. En caso afirmativo, envía un mensaje “new_try” indicando la letra que desea intentar.
 - b. En caso negativo, envía un mensaje “validation_error” indicando los mensajes que son inconsistentes con el estado presentado. Seguidamente, cierra la conexión.
3. El Seleccionador aplica el intento que indica el mensaje “new_try”. Luego:
 - a. Si el nivel de protección es mínimo, envía un mensaje “reply” indicando si el intento fue exitoso o no.
 - b. Si el nivel de protección es intermedio y el intento es exitoso, indica el resultado y revela los valores de VRS y VRNS que corresponden a las posiciones acertadas. Si el intento es fallido, sólo indica el resultado, sin hacer revelaciones.
 - c. Si el nivel de protección es máximo, indica el resultado y revela el valor de VRS y VRNS que corresponde a la letra intentada.
4. El Adivinador evalúa el mensaje “reply”.
 - a. Si el nivel de seguridad es mínimo, envía un mensaje “r_ack”.
 - b. Si el nivel de seguridad es intermedio y:
 - i. El intento es fallido, envía un mensaje “r_ack”.
 - ii. El intento es exitoso, valida que la sección de VRS y VRNS recibida sea consistente con VRCS, y que la sección de VRS junto con VRA sean consistentes con VWC.

1. Si la validación es exitosa, envía un mensaje "r_ack".
 2. Si la validación es fallida, envía un mensaje "validation_error", incluyendo los valores de VRCS y VWC inconsistentes con la información recibida. Seguidamente, cierra la conexión.
- c. Si el nivel de seguridad es máximo valida que la sección de VRS y VRNS recibida sea consistente con VRCS, y que la sección de VRS junto con VRA sean consistentes con VWC.
1. Si la validación es exitosa, envía un mensaje "r_ack".
 2. Si la validación es fallida, envía un mensaje "validation_error", incluyendo los valores de VRCS y VWC inconsistentes con la información recibida. Seguidamente, cierra la conexión.
5. El Seleccionador recibe el mensaje "r_ack", actualiza el número de ronda, el estado de la palabra, y los intentos restantes. Ir a 1.

En la Figura 3.2 Diagrama de secuencia de mensajes para el bucle de juego, se muestran las acciones que toman los jugadores y su efecto en el estado de juego. Las variables generadas previamente y que son utilizadas en este subprotocolo son: Función hash (H), Nivel de seguridad (N), Idioma (Φ), Alfabeto (Γ), Número de intentos (t), Palabra (W), Vector de Palabra (VW), Número de Nonces (J), Vector de Compromiso de Números Aleatorios del Adivinador (VCRA), Vector de Números Aleatorios del Seleccionador (VRS), Vector de Nonces para Números Aleatorios del Seleccionador (VRNS), Vector de Compromisos sobre Números Aleatorios del Seleccionador (VCRS), Vector de Números Aleatorios del Adivinador (VRA), Vector de Nonces para Números Aleatorios del Adivinador (VRNA), Vector de Nonces para la Palabra (VWN), Vector de Compromisos sobre la Palabra (VWC).

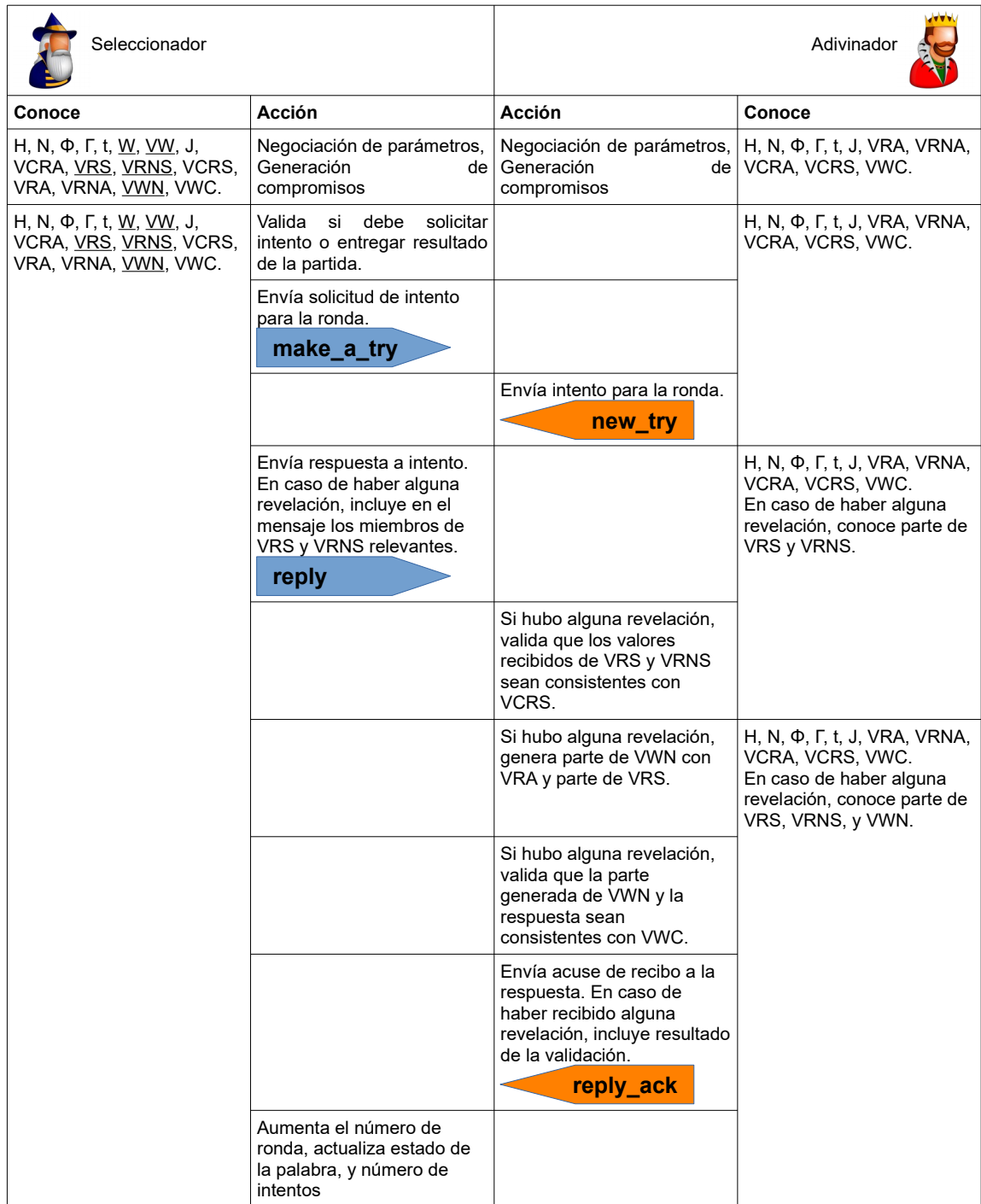


Figura 3.2: Diagrama de secuencia de mensajes para el bucle de juego

Fuente: Elaboración Propia, 2018.

3.3.4 Cierre de la partida

Este protocolo comprende las acciones posteriores al bucle de juego. En esta etapa se revelan los compromisos sobre la palabra para validar que la partida se desarrolló de forma justa. La secuencia de mensajes utilizada es:

1. El Seleccionador envía un mensaje “game_ended”, en el resultado indica si la palabra fue adivinada antes de que agotaran los intentos, e incluye VRS y VRNS.
2. El Adivinador valida que VRS y VRNS sean consistentes con VRCS, y que VRS junto con VRA sean consistentes con VWC.
 - a. En caso afirmativo, envía un mensaje “result_validation”.
 - b. En caso negativo, envía un mensaje “validation_error”, incluyendo los valores de VRCS y VWC inconsistentes con la información recibida. Seguidamente, cierra la conexión.

3.3.5 Abandono de la partida

En cualquier momento, cualquiera de los dos jugadores puede indicar que no desea proseguir con la partida. Para ello, basta con que envíe un mensaje “quit” y cierre la conexión. Se interpreta el envío de un mensaje quit como una rendición frente al oponente. En caso de abandono tácito de la partida, cuando uno de los jugadores deja de enviar jugadas, pero no abandona explícitamente la partida, queda pendiente la definición de un protocolo que permita la adjudicación de la partida sea de forma provisional o permanente.

3.3.6 Validación de mensajes

Cada jugador tiene la responsabilidad de validar los mensajes recibidos antes de actualizar el estado de juego, realizar alguna acción, o enviar algún mensaje. Los criterios para considerar que un mensaje recibido es válido son:

- Tiene una firma válida: Sin esto no puede establecerse su autenticidad.
- La firma clave utilizada es confiable: La clave utilizada para firmar el mensaje no ha sido revocada, y cumple con los requisitos de confiabilidad del receptor.
- Está bien formado: Usa el formato adecuado (JSON), tiene todos los campos necesarios, los campos usan el tipo correcto.
- Lo firmó el remitente: El identificador del usuario que firma el mensaje debe coincidir con el usuario identificado en el mensaje como remitente.

- Está dirigido al receptor: El identificador de usuario del receptor coincide con el identificador de usuario identificado en el mensaje como destinatario.
- Pertenece a la partida: Los identificadores de sesión de emisor y destinatario identifican una partida válida.
- Encaja con el estado de la partida: El tipo de mensaje es una respuesta aceptable al último mensaje enviado. Los valores de los campos son consistentes con los parámetros de juego negociados. Si se trata del bucle de juego, el número de ronda está indicado y es correcto.

Si no se cumplen estos criterios, se debe enviar un mensaje “validation_error” incluyendo el mensaje recibido.

3.3.7 Manejo de errores

A continuación la secuencia que deben seguir los jugadores para manejar errores y recuperarse:

1. Si un jugador A detecta que ha pasado un tiempo prudencial y no ha recibido respuesta del jugador B a su último mensaje, puede reintentar enviándolo de nuevo a B. Esto permite que la partida se reanude en caso de que el último mensaje de A haya sido interceptado o no se haya podido entregar a B.
2. Si un jugador B recibe un mensaje de A que no responde su último mensaje enviado, debe compararlo con su registro de mensajes anteriores:
 - a. Si el mensaje fue enviado antes por A, y es el último mensaje que B recibió de A, debe interpretarse como una solicitud de reenvío, por lo cual B debe repetir el último mensaje que envió a A. Se recomienda reenviar el mensaje último mensaje sin ninguna alteración, ni siquiera en los campos generales, para facilitar la detección de mensajes repetidos.
 - b. Si el mensaje fue enviado antes por A, pero B recibió mensajes posteriores de A, debe interpretarse como una solicitud de reenvío caduca, o un mensaje tardío, o un intento de ataque de repetición. B debe ignorar dicho mensaje.
 - c. Si el mensaje no fue enviado por A anteriormente, pero se trata de un contenido distinto al enviado por un mensaje anterior, debe tratarse como un intento de repudio. En ese caso B debe enviar a A un mensaje de “validation_error” incluyendo ambos mensajes. Nótese que un cambio en unix_time, no debe ser considerado como un intento de repudio, sino de reenvío.

- d. Si el mensaje no fue enviado por A anteriormente, no responde al último mensaje de B, y es de tipo "validation_error", B debe verificar si los mensajes referenciados dentro de "validation_error" cuentan con una firma válida de B, y si son incompatibles entre sí.
 - i. Si no están firmados por B, B debe enviar un mensaje "validation_error" incluyendo el mensaje de A.
 - ii. Si están firmados por B y son incompatibles entre sí, B debe enviar quit, ya que la partida no puede recuperarse.
 - iii. Si están firmados por B pero son compatibles entre sí, B debe enviar un mensaje "validation_error" que incluya el mensaje de A y ambos mensajes de B.
- e. Si el mensaje no fue enviado por A anteriormente, y no responde al último mensaje de B y no es de tipo "unexpected_message", entonces B debe enviar un mensaje "unexpected_message" a A indicando el último mensaje enviado por B que no sea unexpected_message, el mensaje recibido de A, y los tipos de mensajes que está esperando B de A.
- f. Si el mensaje es de tipo "unexpected_message", B debe validar que los mensajes referenciados, hayan sido enviados anteriormente.
 - i. Si B no envió el mensaje que A indica como recibido, pero este está firmado válidamente por B, B debe enviar quit, ya que la partida no puede recuperarse.
 - ii. Si B no envió el mensaje que A indica como recibido, pero no está firmado válidamente por B, B debe enviar un validation_error con el unexpected_message de A.
 - iii. Si B envió el mensaje que A indica como recibido, y el mensaje que A indica como enviado responde el último mensaje de B, tomarlo como respuesta y proseguir la partida.
 - iv. Si B envió el mensaje que A indica como recibido, y el mensaje que A indica como enviado es unexpected_message, enviar un validation_error con el unexpected_message externo.
 - v. Si B envió el mensaje que A indica como recibido, y el mensaje que A indica como enviado no es unexpected_message, procesar este mensaje para determinar la respuesta (ir a 2).

3.4 Cifrado de la comunicación

En los requisitos establecidos para este protocolo sólo se consideró importante proteger la confidencialidad del Secreto durante la partida, más no así la confidencialidad de las jugadas durante la partida. Es decir, no se considera que el justo desenvolvimiento de la partida se vea afectado por la posibilidad de terceros de monitorear el avance de la partida y los mensajes que intercambien los jugadores. En este sentido no requiere ningún mecanismo de cifrado para las comunicaciones, ni el cifrado PGP de los mensajes con la clave pública del receptor, ni el establecimiento de un canal de comunicación cifrado utilizando tecnologías como túneles VPN, túneles TLS, o algún mecanismo similar.

4 Desarrollo del Prototipo

Esta sección se trata sobre el desarrollo de una prototipo que hace uso del protocolo. El objetivo de este ejercicio es demostrar que el protocolo puede ser utilizado para jugar El Ahorcado, y obtener datos sobre su funcionamiento.

El desarrollo se hizo utilizando el lenguaje de programación Python, versión 3. Se utilizaron módulos por defecto para implementar los *sockets* de red, para formatear la información utilizando JSON, para generar números aleatorios, y para aplicar las funciones *hash*. Se seleccionó la función de hash SHA256 (National Institute of Standards and Technology, 2015). Para la firma electrónica PGP, se utilizó el módulo GPG para Python 3. El desarrollo se realizó en dos computadores, uno con Ubuntu 14.04 LTS¹⁷ y otro con Parrot Linux¹⁸, ambas distribuciones pertenecen a la familia de distribuciones derivadas de Debian. Todos las bibliotecas y aplicaciones necesarias se obtuvieron del repositorio de software de cada aplicación.

La idea inicial era implementar un prototipo que pudiese ser utilizado por humanos para jugar El Ahorcado. Sin embargo, durante el desarrollo de este trabajo se hizo más atractiva la idea de desarrollar unos bots que pudiesen jugar partidas de El Ahorcado de forma automatizada. Las razones que impulsaron esta decisión son que con los bots se pueden establecer estrategias de juego predecibles, que los resultados entre partidas son más fácilmente comparables, que se elimina el error humano como factor, y que es posible ejecutar una mayor cantidad de repeticiones.

Se implementaron tres aplicaciones conceptualmente distintas, aunque comparten diversas rutinas y clases. La primera es una aplicación para el Seleccionador, la cual escoge una palabra secreta, recibe intentos, y envía respuestas. La segunda corresponde al Adivinador, la cual hace intentos para adivinar la palabra secreta, y recibe respuestas. La tercera es un proxy que sirve de intermediario entre estas dos aplicaciones. Este proxy permite capturar el tráfico de red de la partida, y medir el número de mensajes intercambiados, la cantidad de datos transmitidos, y el tiempo de la partida.

Para evitar un anidamiento profundo de condicionales, se optó por un patrón de diseño por estados. Esto significa que las aplicaciones almacenan su estado actual, y las posibles transiciones que pueden hacerse desde ese estado.

¹⁷ <http://releases.ubuntu.com/14.04/>

¹⁸ <https://www.parrotsec.org>

Como en el protocolo propuesto el Adivinador es quien toma la iniciativa, se decidió que el Seleccionador fuese un servidor que espera conexiones, y el Adivinador un cliente que envía invitaciones a jugar. La estrategia de implementación utilizada para el Seleccionador y el Adivinador fue incremental. A continuación se detalla más información sobre las funcionalidades implementadas.

4.1 Funcionalidad a implementar

Los elementos del protocolo se pueden dividir en dos grupos. Por un lado, están los elementos externos, que son necesarios para que una aplicación envíe mensajes válidos utilizando el protocolo. Por otro lado, están los elementos internos, que son aquellos necesarios para que una aplicación valide que los mensajes que está recibiendo son válidos. Se consideran elementos externos del protocolo la estructura de los mensajes, y los subprotocolos de negociación de parámetros, generación de compromisos, bucle de juego, cierre de partida, y abandono de partida. Los elementos internos del protocolo son la validación de mensajes, y el manejo de errores.

La implementación de los elementos externos permite demostrar la funcionalidad del protocolo con aplicaciones correctas, mientras que implementar los elementos internos permitiría hacer pruebas prácticas de seguridad con aplicaciones maliciosas. En este trabajo no está previsto hacer pruebas prácticas de seguridad, por lo cual sólo es indispensable la implementación de los elementos externos.

El desarrollo se dividió en fases incrementales. Con esto se persiguió mantener un base de código fuente manejable, facilitar la identificación y corrección de los errores, y evitar la duplicación de código.

4.1.1 Primera fase

Se partió con la implementación de un cliente y un servidor simples (SimpleClient y SimpleServer). Para el proxy se utilizó una aplicación en Python disponible libremente en Internet¹⁹. Estas clases sólo tiene el código suficiente para establecer comunicaciones de red, por lo cual las implementaciones de del protocolo heredan de ellas estas funciones, y determinan el contenido de los mensajes así como su flujo.

Luego se implementó una versión básica del juego, con las clases SimpleChooserServer para el Seleccionador, y SimpleGuesserClient para el Adivinador. Además se definió la clase auxiliar Chooser para que lleve el control de los intentos recibidos, cuántas letras se han adivinado de la palabra, y cuántos intentos le quedan al Adivinador. En esta etapa SimpleChooserServer

19 <http://voorloopnul.com/blog/a-python-proxy-in-less-than-100-lines-of-code/>

requiere que se introduzca manualmente la palabra, y SimpleGuesserClient requiere que se introduzcan manualmente los intentos. Para registrar el estado actual de la aplicación se usa la clase auxiliar CurrentState.

SimpleChooserServer y SimpleGuesserClient no utilizan ninguna medida anti trampas. Es decir, el Seleccionador no hace ningún compromiso sobre el Secreto, y los mensajes no van firmados. Por esta razón no utiliza el módulo hashlib, ni el módulo gpg. Esta versión emplea un subconjunto propio del protocolo propuesto en este trabajo, que incluye el bucle de juego y el cierre de la partida. Estas clases proveen la base de la gestión del estado de juego, incluyendo el envío de intentos y respuestas.

4.1.2 Segunda fase

Se añade a SimpleChooserServer y SimpleGuesserClient la posibilidad de jugar automatizadamente. Para ello Chooser utiliza un archivo de texto que contenga una lista de palabras, una palabra por línea. La primera operación que realiza es excluir todas aquellas palabras que contengan caracteres distintos de letras del normales del abecedario, como apóstrofes, o letras con diéresis, tildes, ligaduras, etc. En este sentido, se seleccionó como fuente el diccionario de palabras en inglés estadounidense disponible en el sistema operativo.

En el caso de SimpleGuesserClient, se crea una clase auxiliar Guesser. Esta clase lleva el control de la información que se conoce de la palabra, y realiza intentos automatizados. Carga una lista de palabras de forma similar a Chooser, y utiliza expresiones regulares para seleccionar las palabras que coinciden con la información conocida hasta el momento. Luego verifica cuál de las letras que restan por intentar está presente en más palabras, y ése es el intento que envía SimpleGuesserClient.

4.1.3 Tercera fase

Se implementa un subconjunto propio del protocolo propuesto en este trabajo, que permite utilizar el protocolo propuesto por Chambers et al (2005). Éste incluye un mensaje de compromiso sobre la palabra antes del bucle de juego, y otro de revelación del compromiso luego del bucle de juego. Para ello se crean las clases ChooserServer_Level1 y GuesserClient_Level1. Estas clases heredan a SimpleChooserServer y SimpleGuesserClient. Para la gestión del compromiso de palabra, se crean las clases auxiliares SimpleCommitment para ChooserServer_Level1, y SimpleRevelation para GuesserClient_Level1, las cuales utilizan la función de *hash* seleccionada.

Se implementa el nivel de seguridad mínimo propuesto en este trabajo. Se crean las clases ChooserServer_Level2 y GuesserClient_Level2. Estas clases heredan de

ChooserServer_Level1 y GuesserClient_Level1, respectivamente. Implementan la lógica de generación distribuida de nonces especificada en el protocolo. Para la gestión del compromiso distribuido, se crean las clases auxiliares DistributedCommitment y DistributedRevelation, las cuales no heredan de SimpleCommitment y SimpleRevelation aunque tienen una interfaz similar. Se utiliza la función de hash seleccionada.

Para la lógica de negociación de parámetros de la partida, se crean las clases auxiliares SupportedParams y GameParams. La aplicación tiene configurado los parámetros de juego que puede utilizar en SupportedParams, y en GameParams almacena los parámetros que se negociaron para una determinada partida. Esta lógica se implementa en SimpleChooserServer y SimpleGuesserClient, de forma que todas las clases que heredan de ellas puedan hacer uso de esta funcionalidad. Así las tres versiones de juego pueden reconocer si su contraparte es compatible.

4.1.4 Cuarta fase

Se implementa el nivel de seguridad intermedio propuesto en este trabajo. Se crean las clases ChooserServer_Level3 y GuesserClient_Level3. Estas clases heredan de ChooserServer_Level2 y GuesserClient_Level2, respectivamente. Se añade a DistributedCommitment y DistributedRevelation la lógica necesaria para gestionar revelaciones intermedias. Se agregan las modificaciones necesarias a los mensajes para comunicar las revelaciones intermedias.

4.1.5 Quinta fase

Se implementa el nivel de seguridad máximo propuesto en este trabajo. Se crean las clases ChooserServer_Level4 y GuesserClient_Level4. Estas clases heredan de ChooserServer_Level3 y GuesserClient_Level3, respectivamente. Se hacen ajustes para que las revelaciones intermedias se hagan en todas las rondas. Se implementa la firma de mensajes con claves PGP, y se traspa esta funcionalidad a las clases ChooserServer_Level2, GuesserClient_Level2, ChooserServer_Level3 y GuesserClient_Level3.

Se hacen ajustes al proxy utilizado y a las clases desarrolladas, a fin de alistarlas para las pruebas del capítulo de verificación y validación.

4.2 Plan de trabajo

En líneas generales, la planificación prevé implementar una fase a la semana, partiendo en noviembre. En la Tabla 4.1 Cronograma de las fases de implementación del prototipo, se puede observar la calendarización completa de estas actividades.

Tabla 4.1: Cronograma de las fases de implementación del prototipo

| | 5 noviembre 9 noviembre | 12 noviembre 16 noviembre | 19 noviembre 23 noviembre | 26 noviembre 30 noviembre | 3 diciembre 7 diciembre |
|--------|----------------------------|------------------------------|------------------------------|------------------------------|----------------------------|
| Fase 1 | | | | | |
| Fase 2 | | | | | |
| Fase 3 | | | | | |
| Fase 4 | | | | | |
| Fase 5 | | | | | |

Fuente: Elaboración Propia, 2018.

4.3 Arquitectura del prototipo

En esta sección se ofrece información que permite entender de mejor manera el funcionamiento del prototipo. En la Figura 4.1 Diagrama de clases resumido se muestra la relación que existen entre las clases implementadas.

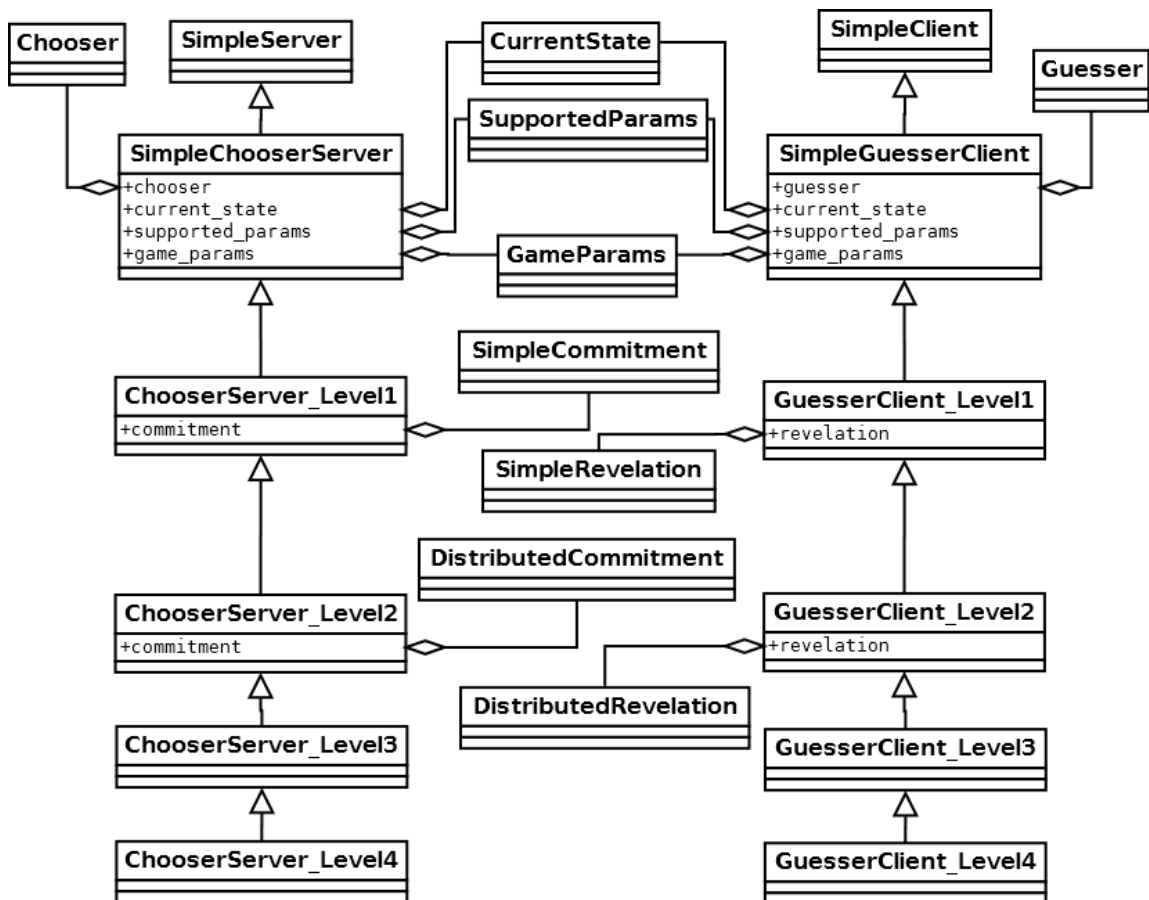


Figura 4.1: Diagrama de clases resumido

Fuente: Elaboración Propia, 2018.

4.4 Resumen

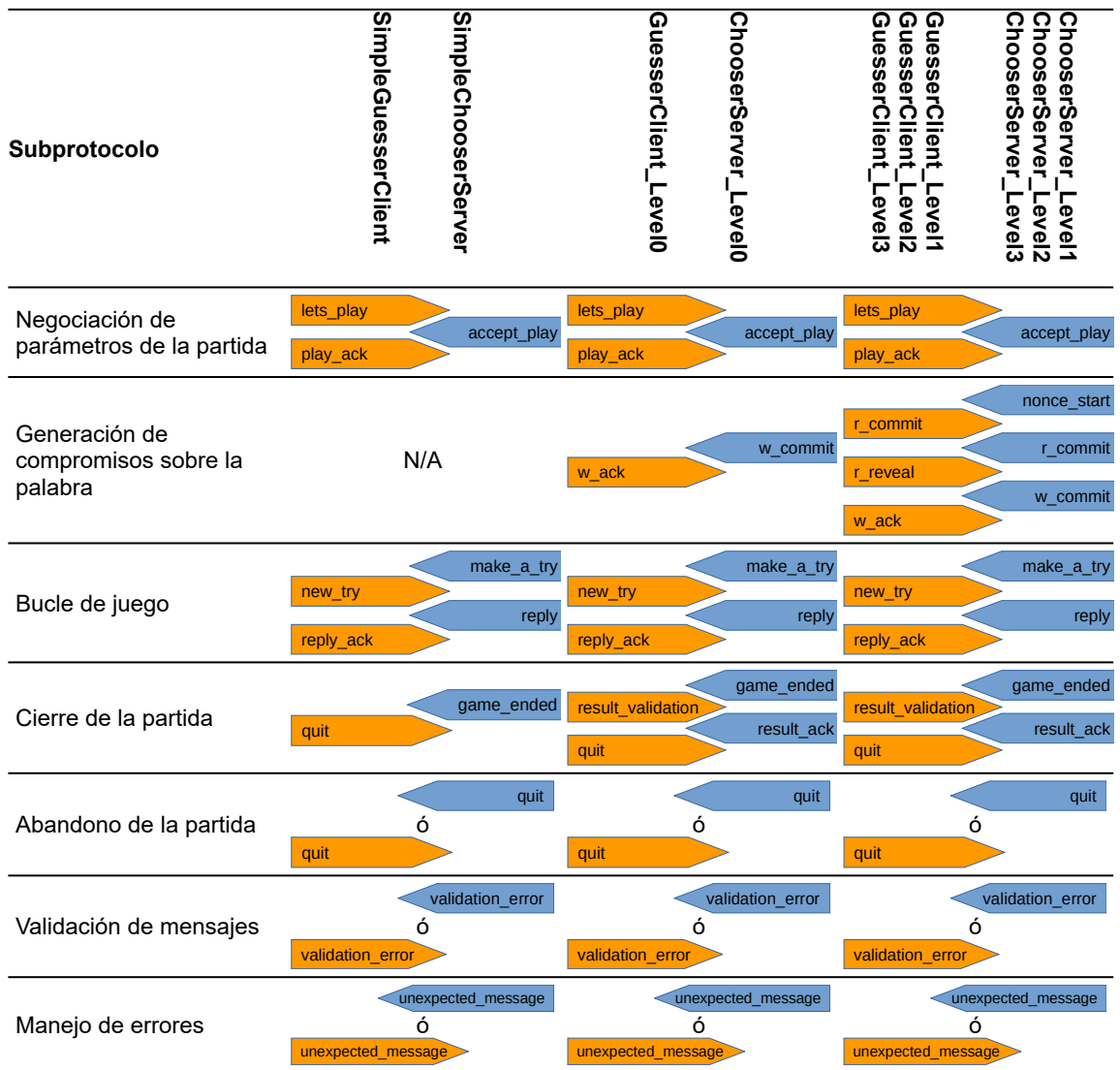
Se obtuvieron cinco implementaciones del juego El Ahorcado, cada una con su Adivinador y su Seleccionador. Una implementación sin seguridad, otra que utiliza una adaptación del protocolo de Chambers et al (2005) y las tres restantes utilizan cada uno de los niveles de seguridad del protocolo propuesto en este trabajo. Las aplicaciones de cada implementación sólo son compatibles con su contraparte, es decir, no se puede desarrollar una partida entre un Adivinador y un Seleccionador de implementaciones distintas. En la Tabla 4.2 Mensajes utilizados por cada implementación, se puede observar cuáles mensajes son utilizados por cada implementación del juego, y en la Tabla 4.3 Secuencia de mensajes utilizada por cada implementación, se puede observar la secuencia que siguen estos mensajes.

Tabla 4.2: Mensajes utilizados por cada implementación

| Tipo de Mensaje | SimpleChooserServer SimpleGuesserClient | ChooserServer_Level0 GuesserClient_Level0 | ChooserServer_Level1 GuesserClient_Level1 | ChooserServer_Level2 GuesserClient_Level2 | ChooserServer_Level3 GuesserClient_Level3 |
|--------------------|--------------------------------------------|----------------------------------------------|----------------------------------------------|----------------------------------------------|----------------------------------------------|
| lets_play | ✓ | ✓ | ✓ | ✓ | ✓ |
| accept_play | ✓ | ✓ | ✓ | ✓ | ✓ |
| reject_play | ✓ | ✓ | ✓ | ✓ | ✓ |
| play_ack | ✓ | ✓ | ✓ | ✓ | ✓ |
| nonce_start | | | ✓ | ✓ | ✓ |
| r_commit | | | ✓ | ✓ | ✓ |
| r_reveal | | | ✓ | ✓ | ✓ |
| w_commit | | ✓ | ✓ | ✓ | ✓ |
| w_ack | | ✓ | ✓ | ✓ | ✓ |
| make_a_try | ✓ | ✓ | ✓ | ✓ | ✓ |
| new_try | ✓ | ✓ | ✓ | ✓ | ✓ |
| reply | ✓ | ✓ | ✓ | ✓ | ✓ |
| r_ack | ✓ | ✓ | ✓ | ✓ | ✓ |
| game_ended | ✓ | ✓ | ✓ | ✓ | ✓ |
| result_validation | | ✓ | ✓ | ✓ | ✓ |
| result_ack | | ✓ | ✓ | ✓ | ✓ |
| quit | ✓ | ✓ | ✓ | ✓ | ✓ |
| validation_error | ✓ | ✓ | ✓ | ✓ | ✓ |
| unexpected_message | ✓ | ✓ | ✓ | ✓ | ✓ |

Fuente: Elaboración Propia, 2018.

Tabla 4.3: Secuencia de mensajes utilizada por cada implementación



Fuente: Elaboración Propia, 2018.

5 Verificación y validación

5.1 Evaluación teórica de seguridad

Para la evaluación teórica de la seguridad del protocolo propuesto,

En esta sección, se analizan los controles que usa el protocolo, es decir, las medidas que toma para mantener la seguridad de la partida. Para ello, se contrastan estos controles contra las amenazas, vulnerabilidades, y buenas prácticas relevantes. En la sección 2.7.4 se listaron las amenazas contempladas en este trabajo. Luego, en la sección 2.7.5 se listaron las vulnerabilidades conocidas. Finalmente, las buenas prácticas a considerar son los once principios para el diseño de protocolos criptográficos (Abadi & Needham, 1995) enunciados en la sección 2.3. Los controles en cuestión son:

- C1. Los mensajes están firmados.
- C2. Los mensajes contienen el identificador del remitente.
- C3. Los mensajes están asociados a una partida específica.
- C4. Los mensajes del bucle de juego están asociados a una ronda específica.
- C5. Los *nonces* se generan de forma descentralizada.
- C6. Los mensajes contiene una marca de tiempo.
- C7. Están definidas las condiciones de validez de los mensajes.
- C8. Están definidas las acciones para el manejo de errores.
- C9. Los parámetros de juego se negocian de forma explícita.
- C10. La gestión de claves PGP es externa al protocolo.
- C11. La comunicación va sin cifrar.

En la Tabla 5.1 Alineamiento de los controles utilizados con los principios de diseño para protocolos criptográficos, se puede observar un cruce entre los controles empleados y las buenas prácticas.

Tabla 5.1: Alineamiento de los controles utilizados con los principios de diseño para protocolos criptográficos

| | | Principios para el diseño de protocolos criptográficos | | | | | | | | | | |
|---------------------|-----|--------------------------------------------------------|----|----|----|----|----|----|----|----|-----|-----|
| | | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | P11 |
| Controles aplicados | C1 | | | | ✓ | ✓ | | | | | | |
| | C2 | ✓ | | | | | | | | | | |
| | C3 | ✓ | | ✓ | | | | ✓ | | | | |
| | C4 | ✓ | | | | | | ✓ | | | | |
| | C5 | | | | ✓ | | ✓ | | | ✓ | | |
| | C6 | | | | | | | | ✓ | | | |
| | C7 | | ✓ | | | | | | | | | |
| | C8 | | ✓ | | | | | | | | | |
| | C9 | | | | | | | | | | ✓ | |
| | C10 | | | | | | | | | | | ✓ |
| | C11 | | | | ✓ | | | | | | | ✓ |

Fuente: Elaboración Propia, 2018.

En la Tabla 5.2 Amenazas cubiertas por los controles aplicados, se puede observar cuáles amenazas son cubiertas por los controles especificados. Tal como se indicó al discutir A1. El Seleccionador pierde el secreto, esta amenaza no puede ser mitigada a nivel de protocolo, ya que no existe un tercero de confianza que resguarde el Secreto en caso de pérdida.

Tabla 5.2: Amenazas cubiertas por los controles aplicados

| | | Amenazas | | | | | | | | | |
|---------------------|-----|----------|----|----|----|----|----|----|----|----|-----|
| | | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 |
| Controles aplicados | C1 | | | | | ✓ | ✓ | ✓ | ✓ | | |
| | C2 | | | | | ✓ | ✓ | ✓ | ✓ | | |
| | C3 | | ✓ | ✓ | | | | | | | |
| | C4 | | ✓ | ✓ | | | | | | | |
| | C5 | | | | ✓ | | | | | ✓ | |
| | C6 | | ✓ | ✓ | | | | | | | |
| | C7 | | ✓ | ✓ | | | | | | | |
| | C8 | | ✓ | ✓ | | | | | | | ✓ |
| | C9 | | | | | | | | | | |
| | C10 | | | | | | | | | | |
| | C11 | | | | | | | | | | |

Fuente: Elaboración Propia, 2018.

En la Tabla 5.3 Vulnerabilidades remediadas/resueltas por los controles, se observan las vulnerabilidades conocidas y cuáles controles las subsanan.

Tabla 5.3: Vulnerabilidades remediadas/resueltas por los controles

| | | Vulnerabilidades | | | | | | | |
|----------------------------|------------|------------------|----|----|----|----|----|----|----|
| | | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 |
| Controles aplicados | C1 | | | | | | | ✓ | |
| | C2 | | | | | | | | ✓ |
| | C3 | | | | | | | | ✓ |
| | C4 | | | | | | | | |
| | C5 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| | C6 | | | | | | | | ✓ |
| | C7 | | | | | | | | |
| | C8 | | | | | | | | |
| | C9 | | | | | | | | |
| | C10 | | | | | | | | |
| | C11 | | | | | | | | |

Fuente: Elaboración Propia, 2018.

5.1.1 C1. Los mensajes están firmados

La firma electrónica garantiza la integridad del mensaje recibido, autentica al emisor, y provee el servicio de no repudio. Con esto, la partida queda protegida de A5, A6, A7, A8. Una de las principales vulnerabilidades detectadas en el protocolo de Chambers et al (2005) es que no especifica la necesidad de utilizar firmas electrónicas para proteger la autenticidad e integridad de los mensajes. Sin esta medida, los demás controles que se agreguen a los mensajes tienen una efectividad limitada, ya que el un atacante puede falsificar mensajes arbitrariamente. Éste es el uso que se declara para la firma electrónica, en concordancia con el cuarto principio de diseño de protocolos criptográficos, el cual indica que debe se debe saber para qué se está haciendo uso de la criptografía en el protocolo. Además, de acuerdo al quinto principio de diseño de protocolos criptográficos, como el remitente firma el el mensaje sin que sea cifrado previamente, puede deducirse que conoce el conoce el texto claro del mismo.

5.1.2 C2. Los mensajes contienen el identificador del remitente

Todos los mensajes del protocolo identifican el remitente dentro del cuerpo del mensaje. Esto refuerza C1, ya que evita vulnerabilidades a ataques MITM²⁰ donde un tercero modifique un

²⁰ MITM: Ataques de interceptación. Del inglés: *Man in the Middle* ("hombre en el medio").

mensaje, lo firma con su propia clave, y el receptor lo considera válidamente firmado. De esta forma fortalece la protección contra A5, A6, A7, y A8. Aunque el receptor podría utilizar otra información de contexto para determinar cuál firma es válida para los mensajes en una partida, esto no sería consistente con el primer principio de diseño de protocolos criptográficos, que indica que los mensajes deben contener todo el contexto necesario para ser comprendidos.

5.1.3 C3. Los mensajes están asociados a una partida específica.

En este protocolo no existe una autoridad central que asigne un identificador único a cada partida. En consecuencia, esta responsabilidad recae en los jugadores. No basta con utilizar un identificador de sesión que asigne sólo uno de los jugadores, para identificar una partida. Incluso si el jugador responsable de asignar el número de sesión lo hace de forma robusta, digamos que utilizando el número n para su n -ésima partida, otros jugadores podrían estar utilizando la misma estrategia. Esto resulta en que haya varias partidas identificadas con el número n , y no puedan ser distinguidas entre sí. El séptimo principio de diseño de protocolos criptográficos, indica habría que proteger el valor de identificador de sesión para que no sea vulnerable a alguien lo simule y luego repita la respuesta. En este sentido, se puede agregar un segundo número de sesión, escogido por el otro jugador, o complementar este número con un espacio de nombre.

Agregar un espacio de nombre al identificador de la partida, quiere decir que en lugar de interpretarse un identificador n como “la partida n ”, se interpretaría como la n -ésima partida de un jugador A , o “la partida n de A ”. Pero en este caso, el jugador B depende de A para que una partida entre ellos cuente con un identificador único. Si A falla en su responsabilidad, varias de sus partidas pueden tener el mismo n . Esto resultaría en que una partida entre A y B se pueda confundir con una partida entre A y C , si ambas se pueden identificar como “la partida n de A ”.

Usar dos números de sesión significa que uno de los jugadores encoge n y su contraparte escoge m . A primera vista, pareciera menos probable que existan dos partidas con los mismos identificadores de sesión si se usa una combinación de n y m , a que existan dos partidas con el mismo identificador si sólo se usa n . Sin embargo, en un contexto de asignación del número de sesión correlativo, podrían haber numerosas partidas entre novatos que usen combinaciones de números bajos, como $(1, 1)$, $(1, 2)$, $(2, 2)$. El uso de identificadores de sesión aleatorios, o no correlativos, disminuiría la probabilidad de que dos partidas tengan el mismo par de identificadores, pero no garantizaría que no ocurra. Ciertamente, se podría utilizar cierto contexto externo para determinar a qué partida pertenece realmente un mensaje, como la dirección IP de origen o destino, o el firmante del mensaje. Pero entonces no se estaría

cumpliendo con el primer principio de diseño de protocolos criptográficos, que indica que los mensajes deben contener todo el contexto necesario para ser comprendidos.

Finalmente, combinando ambas estrategias para identificar correctamente una partida, cada jugador debe ser responsable de asignar un número único a la partida, y estos números deben acotarse dentro de un espacio de nombres. De esta forma se puede determinar que se trata de “la partida n de A” y “la partida m de B”. En este caso B no depende de A para mantener la unicidad de la partida, ya que si A repite n en una partida con C, y C usase el mismo número m, se podría distinguir entre la partida (n de A, m de B) y la partida (n de A, m de C). En consecuencia un jugador que escoge bien su sesión puede estar seguro de que no habrá confusión, así su contraparte no sea igual de responsable.

El tercer principio de diseño de protocolos criptográficos establece que es prudente mencionar explícitamente en el mensaje a los participantes, si su identidad es importante para el significado del mensaje. En este sentido, una vez negociados los parámetros iniciales, el protocolo aplica este control agregando al cuerpo del mensaje la dirección del emisor, la dirección del remitente, y los números de sesión escogidos por cada uno de ellos.

5.1.4 C4. Los mensajes del bucle de juego están asociados a una ronda específica

Las partidas contienen diversas rondas de intentos y respuestas, las cuales usan mensajes del mismo tipo. Estos mensajes pueden confundirse si se reenvían otros mensajes de la misma partida, o de otras partidas. No basta sólo con identificar la partida, ya que igual se podría reenviar un intento o una respuesta previa de la misma partida. Tampoco es suficiente identificar la ronda solamente, porque se podrían confundir los mensajes de partidas distintas, pero que correspondan a la misma ronda.

En este sentido, el protocolo es cónsono con el primer y séptimo de los principios de diseño de protocolos criptográficos. El primer principio reza que los mensajes deben contener todo el contexto necesario para ser comprendidos. Por otro lado, del séptimo principio se puede determinar que se debe proteger el valor de la ronda para que no sea vulnerable a alguien lo use en otra partida y luego repita la respuesta. El protocolo especifica que todos los mensajes involucrados en el transcurso de una ronda (`make_a_try`, `new_try`, `reply`, y `r_ack`) incluyen el número de ronda como campo. Y el control C3 garantiza que se identifique efectivamente la partida en todos los mensajes del bucle de juego.

5.1.5 C5. Los nonces se generan de forma descentralizada.

La mayor dificultad de jugar partidas de JA en forma descentralizada y resistente a trampas, es balancear las necesidades de integridad y confidencialidad del Secreto. Cuando el compromiso es nulo o ambiguo (V1, V6), el Seleccionador tiene más oportunidades de afectar la integridad del Secreto (A4). Sin embargo, algunas medidas que se toman para remediar esta situación (V2, V3, V4, V5), terminan afectando la confidencialidad del Secreto (A9).

El sexto principio de diseño de protocolos criptográficos recomienda tener cuidado con las propiedades que se suponen sobre las *nonces*. Mientras que el noveno principio indica que el uso reciente de una clave, no la hace más confiable. Si se presume el uso de una función de *hash* robusta y el uso de un nonce de suficiente longitud, el protocolo propuesto por Chambers et al (2005) evita V1, sin caer en A9 debido a V2, V3, V4, o V5. Sin embargo, se considera que no otorga al Adivinador suficientes garantías sobre la aleatoriedad y frescura del nonce utilizado para el compromiso sobre la palabra. Por esta razón se plantea que existe una exposición residual a A4 a través de V6.

El protocolo propuesto en este trabajo toma dos medidas respecto a V6. Por un lado, el algoritmo *hash* es acordado entre los jugadores, por lo cual la partida se desarrolla con un algoritmo que ambos consideran seguro. Por otro lado, el nonce para el compromiso de la palabra no es escogido unilateralmente por el Seleccionador, sino que se utiliza un algoritmo seguro de generación distribuida de nonces.

Se deben tener en cuenta las operaciones criptográficas necesarias para la generación distribuida y segura de los *nonces*, en acuerdo con el cuarto principio de diseño de protocolos criptográficos, el cual dice que debe saberse para qué se está haciendo uso de la criptografía en el protocolo. En este sentido, por cada nonce en el vector VWN, se necesita:

- Generar cuatro números aleatorios seguros, que corresponden a los vectores VRS, VRA, VRNS, y VRNA. Cada una de las partes genera un número que saben que no está trucado en su contra.
- Aplicar dos veces la función hash, una para obtener el compromiso en VRCA y otra en VRCS. Impide que alguna de las partes cambie su número aleatorio para forzar un determinado valor de VWN.
- Ejecutar XOR para obtener un valor de VWN. Asegura a ambas partes que el nonce en VWN es imparcial, pues es al menos tan aleatorio como el número que cada uno entregó.

- Ejecutar una vez la función hash, sobre un valor de VWN concatenado con uno de VW, para obtener un compromiso en VWC. Garantiza al Adivinador la integridad del Secreto, y al Seleccionador su confidencialidad.

De esta forma, los *nonces* utilizado para los compromisos sobre la palabra se calculan a partir de números aleatorio escogido por el Seleccionador y otro escogido por el Adivinador. En la etapa inicial el Adivinador sólo conoce sus número aleatorio, pero no conoce los del Seleccionador, aunque tiene compromisos sobre sus valores. Esto quiere decir que el Seleccionador puede estar seguro de que el Adivinador no conoce el valor de los *nonces* para los compromisos de palabra, con lo cual no hay exposición a A9 por V2, V3, V4, ni V5. En cambio, el Adivinador puede estar seguro que los *nonce* para los compromisos de palabra son al menos tan aleatorios y tan fresco como los valores que proporcionó, por lo cual no hay exposición a A4 a través de V1, y se elimina la exposición residual que existía a través de V6.

5.1.6 C6. Los mensajes contiene una marca de tiempo.

El protocolo prevé que todos los mensajes lleven la marca de tiempo de su fecha y hora de emisión. En la versión actual del protocolo, este campo es utilizado para que el orden cronológico de los mensajes ayude a desambiguar su orden lógico.

No se establecieron controles de frescura de los mensajes, lo cual es consistente con el octavo principio de diseño de protocolos criptográficos. Esto se debe a que este principio indica que el establecimiento de controles de frescura implica que el sistema se hace dependiente del buen mantenimiento de la fecha y hora en los equipos participantes, y la medición de la diferencia de hora entre sus relojes. El establecimiento de controles de frescura queda abierto para futuras versiones del protocolo, que incorporen funcionalidades de gestión de fecha hora, así como de gestión de deriva en estos valores.

5.1.7 C7. Están definidas las condiciones de validez de los mensajes.

El segundo principio de diseño de protocolos criptográficos indica que deben estar definidas claramente las condiciones de validez de los mensajes. En este sentido, uno de los subprotocolos establece claramente los pasos que deben seguir las implementaciones del protocolo para validar si un mensaje es válido o no, y el mensaje de error que deben enviar en caso de que falle la validación.

5.1.8 C8. Están definidas las acciones para el manejo de errores

Existe un subprotocolo que define las acciones que deben tomar las implementaciones, y los mensajes que deben enviar en caso de que se presenten errores en el flujo de mensajes

durante la partida. Esta especie de validación de los mensajes, en contexto, es cónsona con el segundo principio de diseño de protocolos criptográficos, que llama a definir claramente las condiciones de validez de los mensajes.

5.1.9 C9. Los parámetros de juego se negocian de forma explícita

De acuerdo al décimo principio de diseño de protocolos criptográficos, debe ser posible determinar la codificación del mensaje, y si ésta depende de la versión del protocolo, entonces debe ser posible saber la versión del protocolo. Esto se logra en el protocolo presentado este trabajo, ya que la secuencia inicial de negociación de parámetros de juego permite que dos clientes establezcan todos los parámetros que utilizarán durante el transcurso de la partida. Así, permite negociar el nivel de seguridad, la función hash a utilizar, y el idioma de juego. Versiones posteriores del protocolo pueden incorporar parámetros adicionales.

5.1.10 C10. La gestión de claves PGP es externa al protocolo.

De acuerdo al undécimo principio de diseño de protocolos criptográficos, el diseñador del protocolo debe explicitar las relaciones de confianza que afectan el protocolo. En este sentido, en la sección de diseño se establece que la correcta gestión de las claves PGP propias, así como las claves públicas de terceros, y la gestión de la red de confianza son responsabilidad de los usuarios.

5.1.11 C11. La comunicación va sin cifrar.

Los requisitos de seguridad de las partidas de JA levantados son: se debe proteger la integridad del Secreto, la confidencialidad del Secreto, y la autenticidad de las jugadas. En consecuencia, en la sección de diseño se indica explícitamente que el protocolo propuesto en este trabajo no requiere ni que se cifren los mensajes, ni que sean transmitidos a través de un canal cifrado. Esta declaración cumple con el cuarto y el undécimo principio de diseño de protocolos criptográficos. El cuarto principio indica que las operaciones criptográficas son costosas, y que por lo tanto se debe saber para qué se están usando. En consecuencia, se está evitando un uso innecesario a la luz de los requisitos. Por otro lado, el undécimo principio requiere que el diseñador de protocolo haga explícitas las relaciones de confianza que afectan al protocolo. En este caso, se dejó explícito que el diseño del protocolo parte de la suposición de que no es necesario proteger la confidencialidad de las jugadas.

5.2 Experimento con el prototipo

Se realizó un experimento controlado con el prototipo que se presentó en el capítulo anterior. El objetivo de este experimento es comprobar que se puede jugar El Ahorcado con una

implementación del protocolo propuesto en este trabajo. Además se desea medir el desempeño de esta solución.

Los elementos a evaluar son los tres niveles de seguridad del protocolo propuesto: mínimo intermedio, y máximo. Como medición de control se cuenta con la implementación que se hizo del protocolo delineado por Chambers et al (2005), y la implementación que se hizo del juego sin ninguna medida de seguridad.

5.2.1 Preparación

Para el experimento se decidió tomar una muestra aleatoria de 100 palabras en inglés estadounidense. En este sentido, se utilizó como fuente el diccionario del sistema con palabras en inglés estadounidense. Este listado se filtró utilizando la herramienta grep para expresiones regulares extendidas, quedando seleccionadas sólo las entradas que contuviesen exclusivamente una de las 26 letras del alfabeto anglosajón. Luego este listado se reordenó de forma aleatoria. Finalmente se seleccionaron las primeras 100 palabras resultantes.²¹ Esta lista se guardó en un nuevo archivo, y puede ser consultada en el Anexo C: Listado de palabras de prueba.

Se hicieron modificaciones de “modo de prueba” a los bots. Para evitar que el tiempo de procesamiento del diccionario del sistema se refleje dentro del tiempo de la partida, se hicieron ajustes para que Adivinador cargase su lista palabras antes de enviar el primer mensaje. En el caso del bot Seleccionador, se adaptó para que utilizase la lista de 100 palabras seleccionadas, y las escoja las palabras secuencialmente, en lugar de aleatoriamente. Esto permite que los resultados entre corridas de experimento sean directamente comparables. Es decir todas las corridas del experimento comienzan por la misma palabra, así que se puede ver el comportamiento relacionado con esa palabra en cada nivel de seguridad. Y luego el Adivinador usa un algoritmo predecible para seleccionar sus intentos. Por lo cual no varía el contenido sustantivo de las partidas, sino el relacionado con las medidas anti trampas.

Además, se configuró el proxy desarrollado para que interceptase las comunicaciones entre los bot Seleccionador y bot Adivinador. Así este proxy puede mantener un registro de los mensajes, cuando fueron enviados, cuando inició la partida, y cuándo culminó. A medida que llegan los mensajes, se van guardando en un archivo de texto que es único para dicha partida. Al finalizar la partida se guardan las métricas recabadas en un archivo de texto que es común a una corrida del experimento.

²¹ `cat /usr/share/dict/american-english | egrep "^[abcdefghijklmnopqrstuvwxyz]+$" > clean.txt`
`cat clean.txt | sort -R | head -n100 > test.txt`

Las tres aplicaciones utilizadas para el experimento se ejecutan en un mismo equipo y se conectan por el loopback de localhost. El bot Seleccionador escucha por el puerto 8089, el proxy escucha en el puerto 9090. Cuando el Adivinador se conecta al proxy, éste inicia una comunicación con el Seleccionador, y de allí comienza la partida.

5.2.2 Variables de contexto que intervienen

Condiciones experimentales:

- Computador portátil marca Lenovo, modelo G40. Cuenta con un procesador Intel(R) Celeron(R) CPU N2840 @ 2.16GHz, y de RAM tiene 4GiB SODIMM DDR3 Synchronous 1333.
- Muestra de palabras: el Seleccionador utiliza en cada corrida del experimento la misma muestra de 100 palabras, y en el mismo orden.
- Diccionario del Adivinador: El Adivinador utiliza en todas las partidas la misma lista de palabras (63.248)
- La función hash, en los niveles de seguridad que la utilizan, es SHA256.
- Las claves PGP, en los niveles de seguridad que la utilizan, son siempre las mismas. Una para el Seleccionador, otra para el Adivinador. Utilizan RSA-3072.

Variables independientes:

- Nivel de seguridad utilizado para jugar El Ahorcado (ninguno, Chambers et al (2005), mínimo, intermedio, máximo)

Variables dependientes:

- Número de mensajes intercambiados.
- Número de bytes transmitidos.
- Tiempo de duración de la partida.

5.2.3 Ejecución

Se ejecutaron cinco corridas del experimento. Cada una de estas rondas se corresponde a una implementación distinta para jugar el Ahorcado. Se pueden ver los detalles de cada corrida en la Tabla 5.4 Resumen de nivel de seguridad y clases a utilizar por corrida del experimento.

Tabla 5.4: Resumen de nivel de seguridad y clases a utilizar por corrida del experimento

| Corrida | Nivel de Seguridad | Clase Seleccionador | Clase Adivinador |
|---------|-----------------------|----------------------|----------------------|
| 0 | Ninguno | SimpleChooserServer | SimpleGuesserClient |
| 1 | Chambers et al (2005) | ChooserServer_Level1 | GuesserClient_Level1 |
| 2 | Mínimo | ChooserServer_Level2 | GuesserClient_Level2 |
| 3 | Intermedio | ChooserServer_Level3 | GuesserClient_Level3 |
| 4 | Máximo | ChooserServer_Level4 | GuesserClient_Level4 |

Fuente: Elaboración Propia, 2018.

La corrida 0 y 1 sirven como grupo de control, ya que permiten medir qué diferencias resultan de aplicar los controles definidos el capítulo 3. Las corridas 2, 3, y 4 utilizan los tres niveles de seguridad definidos en este trabajo. En el Anexo B se incluyen el contenido JSON de los mensajes de las partidas correspondientes a la palabra “sera”, correspondiente a cada una de las cinco corridas. Allí se pueden observar las diferencias entre los mensajes enviados, y los campos utilizados.

5.2.4 Resultados

En esta sección se presentan los resultados de las cinco corridas del experimento. Los datos están agrupados por variable dependiente para facilitar la comparación entre sí de los niveles de seguridad.

5.2.4.1 Mensajes intercambiados

En la Tabla 5.5 Resumen de mensajes intercambiados, se presenta la totalización de los datos recabados de todas las corridas, incluyendo ciertos valores estadísticos relevantes.

Tabla 5.5: Resumen de mensajes intercambiados

| Corrida | # Partidas | Total Mensajes | Mínimo mensajes | Máximo mensajes | Mediana mensajes |
|---------|------------|----------------|-----------------|-----------------|------------------|
| 0 | 100 | 4.164 | 21 | 61 | 41 |
| 1 | 100 | 4.564 | 25 | 65 | 45 |
| 2 | 100 | 4.964 | 29 | 69 | 49 |
| 3 | 100 | 4.964 | 29 | 69 | 49 |
| 4 | 100 | 4.964 | 29 | 69 | 49 |

Fuente: Elaboración Propia, 2018.

Las principales observaciones que se pueden hacer acerca de la tabla son:

- Hay cuatro mensajes adicionales por partida entre la corrida 0 y la corrida 1
- Hay cuatro mensajes adicionales por partida entre la corrida 1 y la corrida 2.
- Las corridas 2, 3, y 4 utilizan la misma cantidad de mensajes.

En la Figura 5.1 Detalle de mensajes intercambiados, se puede observar que la distribución de estos mensajes adicionales es uniforme. Esto quiere decir que el aumento en la cantidad de los mensajes afecta transversalmente a todas las palabras. Esta diferencia en la cantidad de mensajes, se puede atribuir entonces al uso de mensajes de control. Los bots de corrida 0 no utilizan ningún tipo de seguridad, sino sólo los mensajes funcionalmente necesarios para el desarrollo de una partida. En el caso de la corrida 1, los bots que utilizan el protocolo de Chambers et al (2005) utilizan cuatro mensajes de control: `w_commit` para comprometerse con la palabra, `w_ack` como acuse de recibo a `w_commit`, `result_validation` que expresa que el compromiso se cumplió satisfactoriamente, y `result_ack` que acusa el recibo de `result_validation`. Por otro parte, el protocolo propuesto en este trabajo, en sus tres niveles, además de los cuatro mensajes de control para el compromiso sobre la palabra, requiere otros cuatro mensajes de control adicionales para la generación distribuida de nonces. Los mensajes adicionales que utiliza son: `nonce_start`, dos mensajes `r_commit`, y un mensaje `r_reveal`.

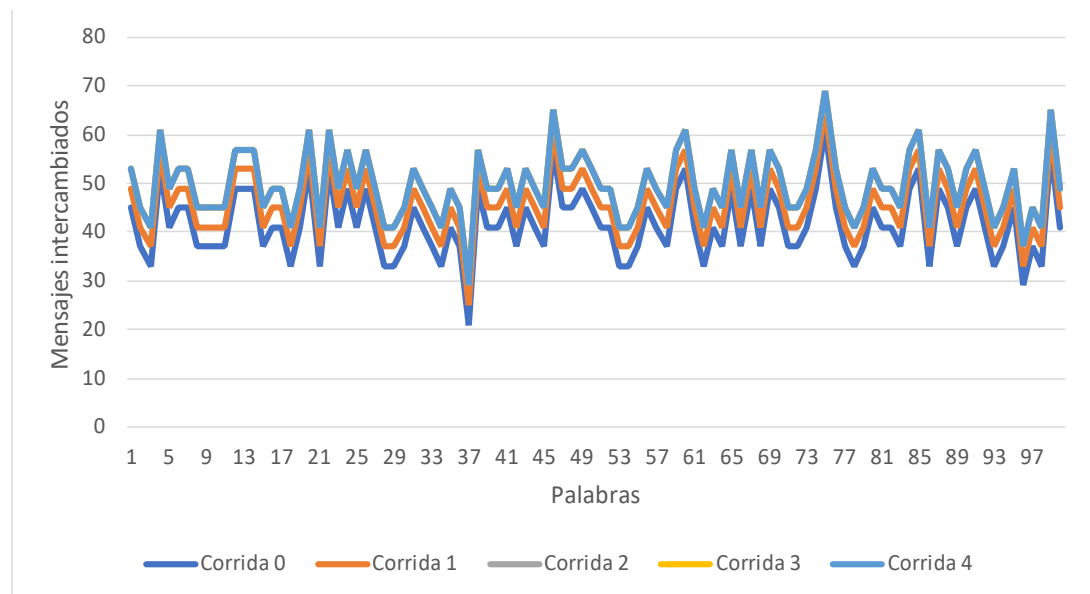


Figura 5.1: Detalle de mensajes intercambiados

Fuente: Elaboración Propia, 2018.

5.2.4.2 Bytes transmitidos

En la Tabla 5.6 Resumen de bytes transmitidos, se presenta la totalización de los datos recabados de todas las corridas, incluyendo ciertos valores estadísticos relevantes.

Tabla 5.6: Resumen de bytes transmitidos

| Corrida | # Partidas | Total bytes | Mínimo bytes | Máximo bytes | Mediana bytes |
|---------|------------|-------------|--------------|--------------|---------------|
| 0 | 100 | 219.196 | 1.113 | 3.096 | 2.186 |
| 1 | 100 | 247.685 | 1.398 | 3.381 | 2.471 |
| 2 | 100 | 4.514.534 | 26.428 | 62.631 | 44.594 |
| 3 | 100 | 4.777.530 | 27.629 | 64.546 | 47.617,5 |
| 4 | 100 | 5.203.123 | 32.618 | 70.153 | 51.436,5 |

Fuente: Elaboración Propia, 2018.

Las principales observaciones que se pueden hacer sobre estos resultados son:

- A medida que aumenta el nivel de seguridad, aumenta la cantidad de bytes enviados.
- Las corridas 2, 3, y 4 usan una orden de magnitud más de datos que las corridas 0 y 1.
- Al comparar la corrida 2 con la 3, y la 3 con la 4, se observa que la diferencia absoluta de bytes es comparable al de una partida completa de las corridas 0 y 1.

En la Figura 5.2 Distribución por partida de los bytes transmitidos, se utiliza una escala logarítmica para que sean comparables los totales de bytes transmitidos para una misma palabra. Se observa que el aumento y disminución de bytes transmitidos por palabra, es consistente entre los niveles de seguridad.

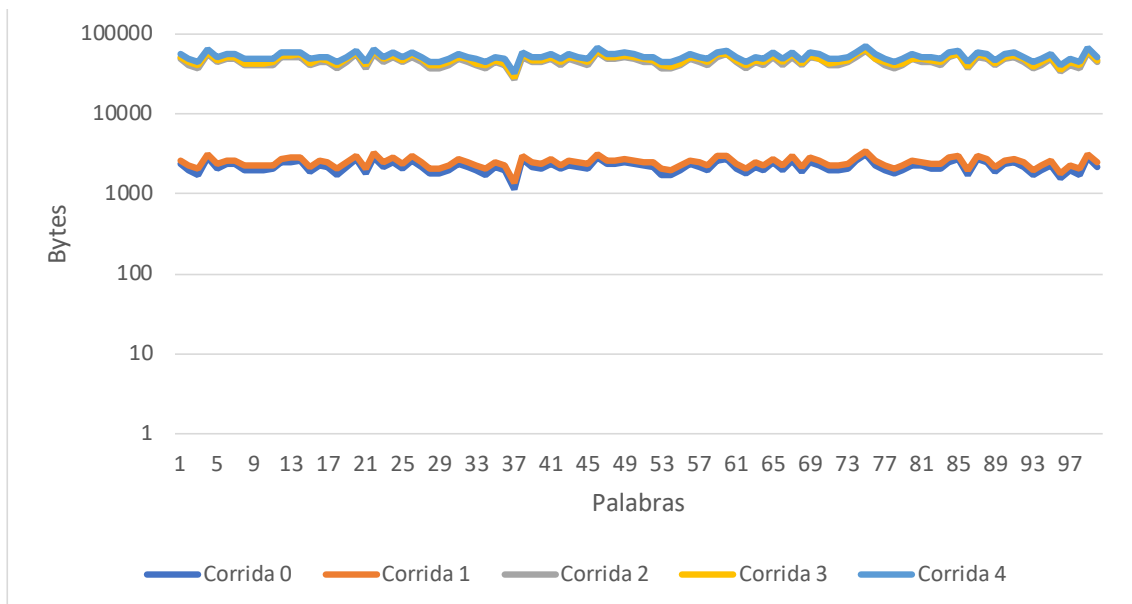


Figura 5.2: Distribución por partida de los bytes transmitidos

Fuente: Elaboración Propia, 2018.

Sin embargo, se debe tener en cuenta el aumento en el número de mensajes de control detectado previamente. En este sentido conviene calcular la densidad de bytes por mensaje. De esta forma se puede determinar en cuánta medida se están transmitiendo más bytes porque se transmiten más mensajes, y en cuánto se debe a que los mensajes sean más largos. En la Tabla 5.7 Resumen de tamaño de mensaje (bytes/mensaje), se puede observar la cantidad de bytes por mensaje.

Tabla 5.7: Resumen de tamaño de mensaje (bytes/mensaje)

| Corrida | # Partidas | Tamaño Promedio | Mínima tamaño | Máxima tamaño | Mediana tamaño |
|---------|------------|-----------------|---------------|---------------|----------------|
| 0 | 100 | 52,64 | 50,22 | 55,36 | 52,93 |
| 1 | 100 | 54,27 | 51,81 | 57,08 | 54,55 |
| 2 | 100 | 909,45 | 906,49 | 911,87 | 909,57 |
| 3 | 100 | 962,44 | 927,51 | 1.003,49 | 965,66 |
| 4 | 100 | 1.048,17 | 1.016,71 | 1.124,76 | 1.049,72 |

Fuente: Elaboración Propia, 2018.

En la anterior se puede observar el tamaño de los mensajes aumenta con el nivel de seguridad. Este aumento conviene analizarlo por tramos. Esto se debe a que el tramo del grupo de control tiene valores similares entre sí, el tramo del protocolo propuesto también tiene valores similares entre sí, pero la diferencia es grande entre ambos grupos.

En cuanto al primer grupo, el aumento de tamaño de mensajes en la corrida 1 respecto a la corrida 0, se debe al compromiso sobre la palabra. Ya que `w_commit` es el segundo mensaje más largo de la partida, por contener un valor hash SHA256 (el mensaje más largo es `lets_play` por la cantidad de parámetros que contiene). Adicionalmente, en la corrida 1 `game_ended` debe revelar el *nonce* utilizado para comprometerse con la palabra, por lo cual se alarga este mensaje. Los otros mensajes de control, no afectan tanto la cantidad de bytes transmitidas. Por un lado, `result_validation` está por debajo del tamaño promedio de los mensajes, y `r_ack` es un mensaje breve.

En cuanto al segundo grupo, se debe recordar que el número de mensajes por partida es el mismo. Así que la diferencia la hace el contenido de los mensajes. En este sentido, se tienen campos adicionales en los mensajes, y campos que se hacen más extensos. Los campos que se hacen más extensos, incluyen todo lo relacionado a la generación distribuida de nonces, y al compromiso sobre la palabra. En particular, estos mensajes son `r_commit`, `r_reveal`, `w_commit`, y `game_ended`. Estos mensajes en el nivel mínimo usa listas de longitud 1, mientras que en nivel intermedio utiliza listas de la longitud de la palabra, y en el nivel máximo utiliza listas de longitud 26. En algunos casos los campos son enteros (`r_reveal`, `game_ended`), pero en otros

casos se trata de valores hash SHA256 (r_commit, w_commit). Los campos adicionales se refiere aquellos necesarios para realizar las revelaciones y comprobaciones intermedias. Los mensajes afectados son reply, y r_ack. Si bien r_ack, solo debe agregar un campo breve, reply debe incluir un campo entero y tres listas de enteros.

La gran diferencia entre ambos grupos tiene que ver con dos factores, campos de contexto y firma electrónica. En el segundo grupo, todos los mensajes llevan ciertos campos de contexto, que no se incluyen en el primer grupo. Estos campos de contexto son los identificadores de los jugadores, los identificadores de sesión de juego, la marca de tiempo de emisión, y el número de ronda en los mensajes del bucle de juego. Sin embargo, su impacto es menor con el impacto de la firma PGP. Se apreciar como estos factores afectan la longitud del mensaje al comparar la Figura 5.3 Mensaje play_ack de la corrida 0 palabra 95 (execute), con la Figura 5.4 Mensaje play_ack de partida corrida 3, palabra 95 (execute). En este caso, los campos extras aportan 135 bytes, mientras que la firma PGP aporta 708 bytes. Nótese que el número que precede el mensaje, se transmite en red como un entero de 32 bits, y así fue tabulado.

```
21{"type": "play_ack"}
```

Figura 5.3: Mensaje play_ack de la corrida 0 palabra 95 (execute)

Fuente: Elaboración Propia, 2018.

```
864-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

{"type": "play_ack", "unix_time": 1561346883, "my_address":
"arthur@tesis", "my_session_id": 6371, "your_address": "merlin@tesis",
"your_session_id": 3371}
-----BEGIN PGP SIGNATURE-----

iQGzBAEBCgAdFiEEMe7yokU2rRLFiurcabxprheRJvcFAl0QQ0MACgkQabxprheR
Jvf0vgv/bN7soC3Y7M97XYedc813n87E+vMHaTa8XYwgP4LK3B9U8MwUkp6h2f5S
o2PlrvN2OVN9eEPVa/ccVlpdBcWzZTPyI+VZnlyQS6RbPhKjVlvPl4Gj2DOzBKm/
ROGECLVnq23FbLjwe6ZN2+s+xY9URLb01ixO52i6ltF+mPXfuM4HuvpOqkHW6DPd
4gDHNO+tI88nVWt6zWeIug00MYolap7VQU3VEiVRxQvB58mNPzEnXZFnnY2aC4qn
bnkNdxh9k7tD/1JDRsKPJrWQY/qVvNit4gBg23Yyk2HN8WQTUjkrgeWGDmdQI/RA
HmRJsSLnO0u2tF9ztpg+r7SLodZRmtmAIZapsaw9v0ow7B6Siq3GULPjninpjrUM
dBMggHHE6Jaf9NKT5/mgi3GA1+4btWhKrMvnWkADnSwBMmQNfLFri5gXRK4UYC+H
y/0bMY91zQvdW3s2fWV85jBEQbLFAJ0DvwWkXxis3GffIOIdd4puc9Q2wO8Wz5CD
W6bS9Cqs
=UA/Y
-----END PGP SIGNATURE-----
```

Figura 5.4: Mensaje play_ack de partida corrida 3, palabra 95 (execute)

Fuente: Elaboración Propia, 2018.

Usando como base la información anterior, y teniendo en cuenta el largo constante de las firmas PGP, se puede aproximar el impacto de las firmas PGP en el tamaño de los mensajes. Se puede observar este cálculo en la Tabla 5.8 Impacto de la firma en el tamaño promedio de los mensajes, y en la Figura 5.5 Impacto de la firma en el tamaño promedio de los mensajes.

Tabla 5.8: Impacto de la firma en el tamaño promedio de los mensajes

| Corrida | Tamaño Promedio | Bytes firma PGP | Bytes sin firmar |
|---------|-----------------|-----------------|------------------|
| 0 | 52,64 | | 52,64 |
| 1 | 54,27 | | 54,27 |
| 2 | 909,45 | 708 | 201,45 |
| 3 | 962,44 | 708 | 254,44 |
| 4 | 1.048,17 | 708 | 340,17 |

Fuente: Elaboración Propia, 2018.

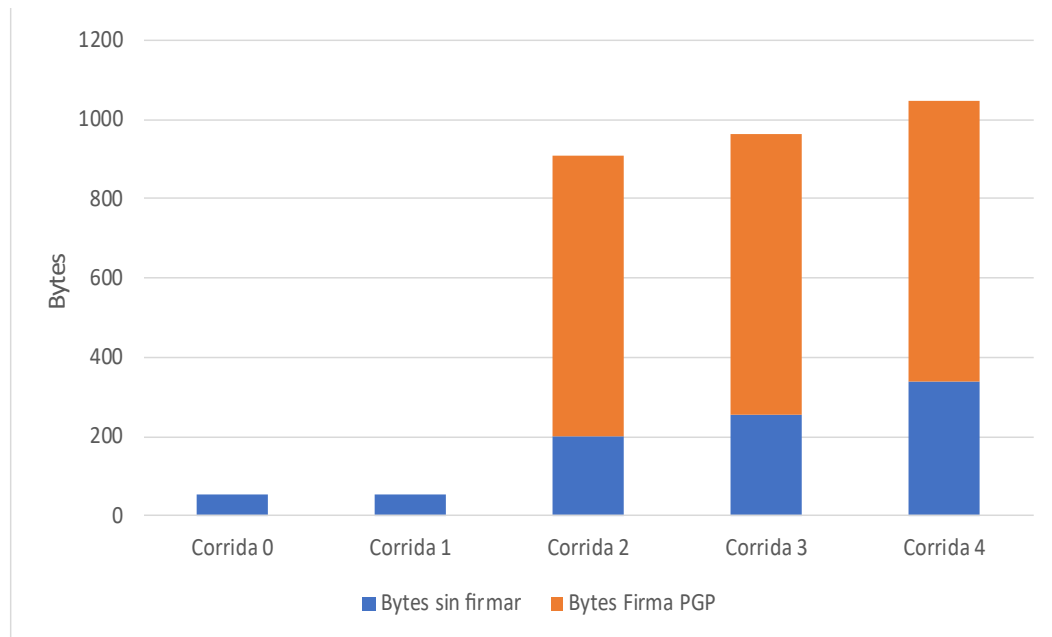


Figura 5.5: Impacto de la firma en el tamaño promedio de los mensajes

Fuente: Elaboración Propia, 2018.

La tabla anterior permite granularizar el impacto de los controles propuestos en este trabajo. Es relevante recordar que la diferencia entre el nivel mínimo y el protocolo de Chambers et al (2005), es que se añaden campos de contexto, el nonce para el compromiso de la palabra se genera de forma distribuida, y los mensajes van firmados. En particular, se observa que el impacto de la firma PGP en los bytes transmitidos es mucho mayor que el de los campos de contexto y la generación segura de *nonces*. Por otro lado, al comparar de nuevo las corridas 2,

3, y 4, se aprecia como crece el tamaño de los mensajes sin firmar, en la medida en cuanto se agregan más compromisos, y revelaciones intermedias.

5.2.4.3 *Tiempo de partida*

En la Tabla 5.9 Resumen tiempo de partida, se presenta la totalización de los datos recabados de todas las corridas, incluyendo ciertos valores estadísticos relevantes.

Tabla 5.9: Resumen tiempo de partida

| Corrida# | Partidas | Total | Tiempo Mínimo | Tiempo Máximo | Tiempo Mediana | Tiempo |
|----------|----------|----------|---------------|---------------|----------------|--------|
| 0 | 100 | 125,22 | 0,40 | 2,10 | 1,22 | |
| 1 | 100 | 125,32 | 0,41 | 2,15 | 1,22 | |
| 2 | 100 | 2.589,05 | 14,89 | 36,22 | 25,43 | |
| 3 | 100 | 2.586,21 | 15,02 | 35,97 | 25,41 | |
| 4 | 100 | 2.629,43 | 15,21 | 36,39 | 25,89 | |

Fuente: Elaboración Propia, 2018.

En la tabla anterior se puede observar nuevamente una clusterización de los resultados. Por un lado la corrida 0 y 1 tienen resultados muy similares, así como las corridas 2, 3, y 4 tienen resultados muy parecidos entre sí. Sin embargo, entre ellos hay una brecha grande. Las corridas de 2, 3, y 4 demoran unas 25 veces más tiempo en completarse que las partidas de las corridas 0 y 1. Por otro lado, resulta interesante que la cuarta corrida tenga algunos tiempos menores que la corrida 3.

En la Figura 5.6 Distribución de tiempo de partida, se utiliza una escala logarítmica para que sean comparables los tiempos de partida para una misma palabra. Se observa que el aumento y disminución del tiempo de partida es consistente entre los niveles de seguridad.

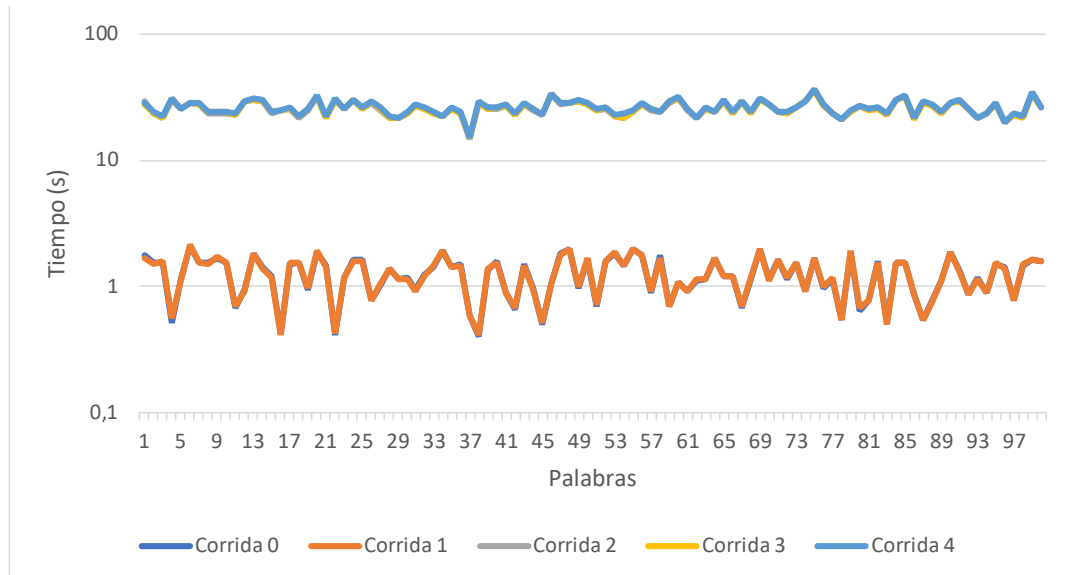


Figura 5.6: Distribución de tiempo de partida

Fuente: Elaboración Propia, 2018.

La única diferencia entre las corridas 0 y 1, son los 4 mensajes de control adicionales y el cálculo de un hash y su posterior verificación. En la Figura 5.7 Diagrama de caja tiempo de partida corridas 0 y 1, se puede observar que no existe una diferencia significativa entre los tiempos de ejecución de estas dos corridas.

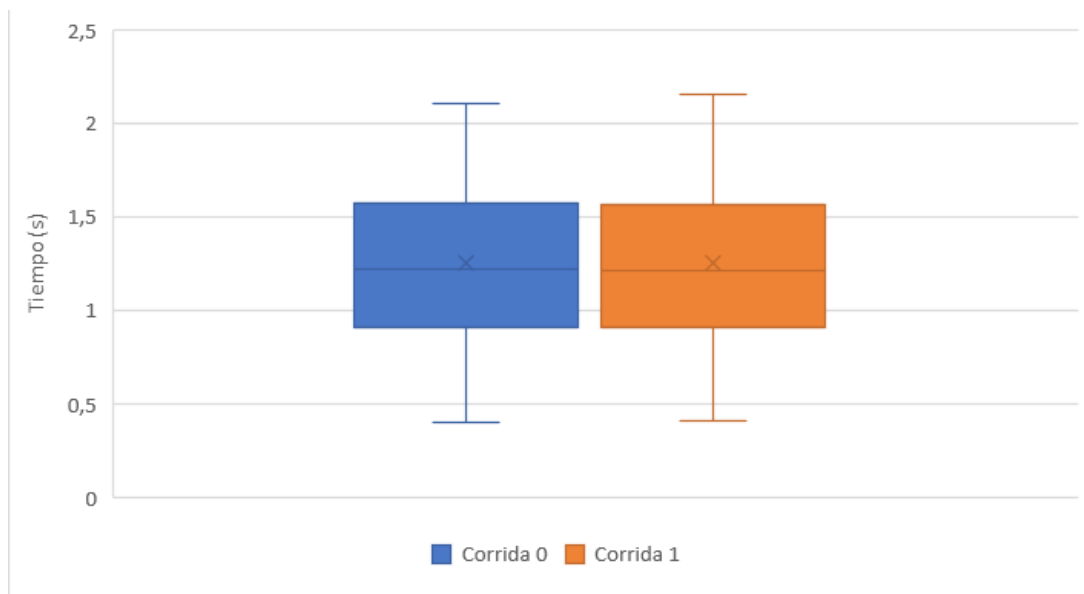


Figura 5.7: Diagrama de caja tiempo de partida corridas 0 y 1

Fuente: Elaboración Propia, 2018.

La diferencia entre las corridas 2, 3, y 4, es el número de números aleatorios generados, valores *hash* calculados, y operaciones XOR realizadas. En la Figura 5.8 Diagrama de caja de tiempo de partida corridas 2, 3, y 4, se puede observar que no existe una diferencia significativa entre los tiempos de partida de estas tres corridas.

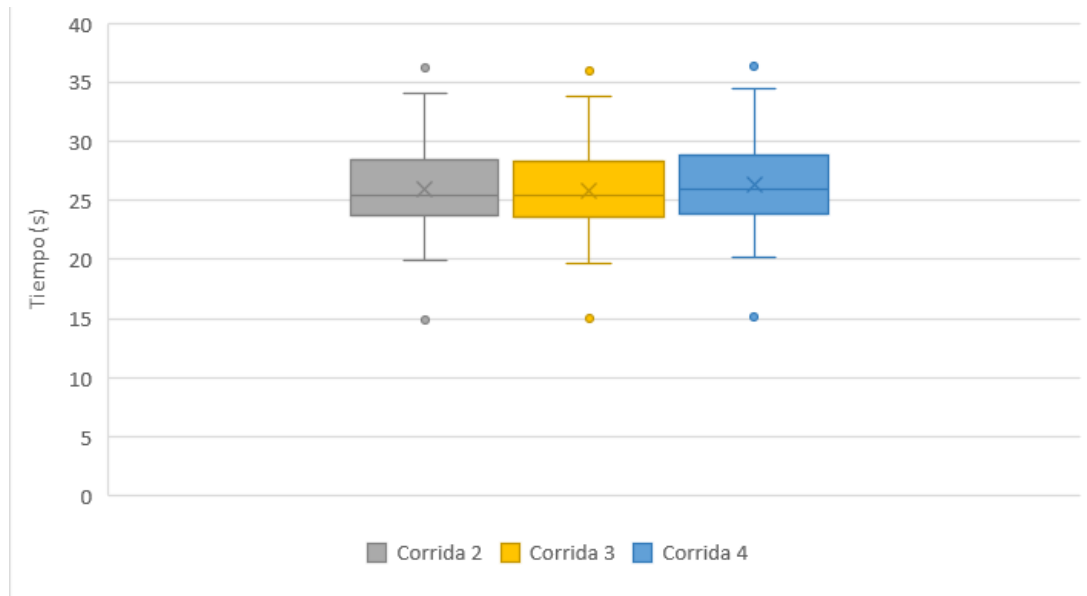


Figura 5.8: Diagrama de caja de tiempo de partida corridas 2, 3, y 4

Fuente: Elaboración Propia, 2018.

Sólo queda por explicar entonces la gran brecha en el tiempo de partida, que existe entre el grupo de control y el protocolo propuesto. Ya se observó que el impacto de calcular *hashes* y realizar operaciones XOR, no es significativo. Entonces el factor que queda por tomar en cuenta es, nuevamente, la firma electrónica. Esto explicaría que el tiempo de ejecución de la corrida 2 sea 25 veces el de la corrida 1.

6 Conclusiones

6.1 Conclusiones generales

Para prevenir las trampas en JA, no es necesario que terceros de confianza mantengan un registro centralizado o exhaustivo de los datos de los jugadores y sus actividades. Se propusieron controles que pueden ser aplicados por las aplicaciones de los jugadores, y que aseguran el justo desarrollo de las partidas. Se realizó una evaluación teórica de la seguridad de estos controles, y se implementaron en un prototipo, el cual fue utilizado para probar el desempeño del protocolo.

Este trabajo extiende el estado previo del arte, al especificar en detalle un protocolo para JA. Existen trabajos que detallan protocolos para otros tipos de juegos, y trabajos que plantean un protocolo para JA, pero sólo a grandes rasgos.

Adicionalmente, este trabajo realiza análisis de las vulnerabilidades y amenazas relevantes para los JA (ver sección 2.7). En particular, detecta algunas debilidades en protocolos previos, y siguiendo las mejores prácticas de diseño de protocolos criptográficos, propone controles que resuelven estas vulnerabilidades.

El nivel mínimo de seguridad provee una capacidad de validación similar a las implementaciones tangibles de los JA, donde el Secreto se suele revelar al final. El nivel intermedio de seguridad provee una capacidad superior de validación a las que están disponibles en los JA tangibles, y en el estado previo del arte para JA. El nivel máximo de seguridad provee una capacidad de validar todas las jugadas de una partida de JA. Sin embargo, se plantea como limitación que requiere que las posibles jugadas del adivinador puedan ser enumeradas, y sean finitas. Esto se logra adaptando a los JA un protocolo ideado originalmente para el lanzamiento de monedas de forma descentralizada, y luego utilizado en los JCC.

El elemento más crítico para optimizar el protocolo es la firma electrónica. En la etapa de evaluación (ver sección 5.2), las variables más impactadas por el nivel de seguridad fueron bytes transmitidos y el tiempo de la partida. Y en ambos casos la firma PGP tuvo la mayor parte de la responsabilidad.

Además de la implementación descentralizada y segura de juegos de adivinanza, como El Ahorcado, *Battleship*, o ¿Adivina Quién?, el protocolo desarrollado en este trabajo tiene otros usos. En particular se puede utilizar para la implementación segura y auditable de apuestas, o de concursos, tales como los programas de concursos de la TV que usan El Ahorcado.

6.2 Sobre los objetivos

6.2.1 Objetivo General

OG. Diseñar un protocolo resistente a trampas descentralizado para juegos de adivinanza en redes.

El protocolo propuesto en este trabajo es descentralizado, ya que no depende de autoridades centrales ni terceros de confianza. Adicionalmente, añade medidas de resistencia a trampas más robustas que las propuestas anteriormente, las cuales fueron contrastadas contra una lista de amenazas, vulnerabilidades, y buenas prácticas. Finalmente, a través de la implementación, se demostró que permite jugar partidas de El Ahorcado en red.

6.2.2 Objetivos Específicos

OE1. Sintetizar las dificultades específicas asociadas llevar a cabo juegos de adivinanzas de forma descentralizada.

En el capítulo 1 se explicaron los elementos de los JA: el Secreto, el Seleccionador, y el Adivinador. En la sección 2.7 se listaron las vulnerabilidades y amenazas relevantes para los JA. De la misma forma, se detallaron las necesidades de integridad, confidencialidad, y autenticación de estos juegos.

OE2. Definir los requisitos que debe cumplir un protocolo para que sea resistente a trampas y descentralizado.

En el capítulo 1 se listaron las características con las cuales debe contar una solución, y luego en el capítulo 3 se listaron los requisitos que debe cumplir una solución.

OE3. Diseñar y desarrollar un protocolo para juegos de adivinanza que cumpla con los requisitos definidos.

En el capítulo 3 se detalló el protocolo propuesto en este trabajo. Allí se definió el formato general que tienen los mensajes a intercambiar durante la partida, y la estructura específica de cada tipo de mensaje. Luego se definieron las secuencias de mensajes que deben intercambiarse en cada fase de la partida, incluyendo las condiciones que éstos deben cumplir. El protocolo permite realizar partidas en los tres niveles seguridad definidos.

En la sección 5.1 se muestra que los controles que se incorporaron al protocolo cumple con los requisitos definidos. Por una parte, desde una perspectiva general, el protocolo se ajusta a las mejores prácticas de diseño de protocolos criptográficos. Por otro lado, a un nivel más

específico a los JA, se observa que el protocolo controla las amenazas previstas, tomando en cuenta los posibles ataques, y las vulnerabilidades conocidas.

OE4. Demostrar la funcionalidad del protocolo mediante la implementación de un prototipo de juego de adivinanza.

En el capítulo 4 se entrega información sobre los bots que se implementaron para jugar El Ahorcado de forma automatizada. Un bot cumple el rol de Seleccionador y otro bot el rol de Adivinador. Se hizo una versión de los bots para cada nivel de seguridad definido, por lo cual se demostró la funcionalidad del protocolo en los tres niveles de seguridad.

OE5. Evaluar el desempeño del protocolo mediante la repetición de experimentos controlados con el prototipo.

En el capítulo 5 se muestran un resumen los resultados de la prueba de ejecución de las partidas de el Ahorcado con los bots implementados. En este sentido se pudo observar el impacto que tuvieron los controles implementados en el desempeño de una aplicación para jugar El Ahorcado.

6.3 Trabajos futuros

- Realizar una evaluación práctica de seguridad del protocolo. Para ello se deben implementar los protocolos de validación y manejo de errores en el prototipo. Luego, se debe definir una batería de ataques, y aplicarla al prototipo para validar su resistencia.
- Cambiar el formato externo de los mensajes, de mensajes PGP firmados, a una estructura similar a JWS, donde se especifique en un sobre el contenido del mensaje y una firma desprendida. Esto permitiría separar la validación del mensaje de su lectura, y el uso de esquemas de firma distintos.
- Explorar alternativas a de firmado electrónico que sean más livianas.
- Extender el protocolo para permitir que quien inicia la partida puede tomar tanto el rol de Adivinador como de Seleccionador.
- Adaptar el protocolo a otros JA como *Battleship* o “Adivina quién?”.
- Diseñar un mecanismo para llevar de forma descentralizada un registro con los resultados de las partidas, y que permita generar una tabla de clasificación de los jugadores.
- Establecer mecanismos descentralizados de descubrimiento de jugadores.

6.4 Reflexión personal

Resulta preocupante el actual proceso de centralización de Internet. Los grandes conglomerados de Internet ofrecen una amplia variedad de servicios, que les dan acceso a los datos personales de los usuarios. Así conocen quienes son sus allegados, el contenido de sus comunicaciones, su ubicación, sus intereses, e inquietudes. Esto genera una gran asimetría de información, ya que estas organizaciones conocen todo sobre sus usuarios, pero los usuarios no tienen control sobre su uso. Actualmente, esta información no sólo está siendo utilizada para manipular a las personas con fines comerciales, sino también con fines políticos.

En este sentido, vale la pena que los usuarios elaboren requisitos éticos que deben cumplir los sistemas, de forma de contrarrestar esta tendencia. En este caso, relacionados más específicamente con la privacidad y la descentralización. Para proteger la privacidad, se pueden establecer objetivos como: minimizar la recolección de datos personales, especialmente si se trata de datos sensibles; priorizar el procesamiento local de datos sobre el procesamiento en la nube; o que los datos se cifren en la nube con claves bajo dominio de los usuarios. Para promover la descentralización, se puede establecer como objetivo la portabilidad de datos de los usuarios, el uso de estándares establecidos, o el uso de sistemas bajo licencias libres.

Glosario

- Acuerdo bizantino: estrategia para llegar a acuerdos cuando las comunicaciones no son confiables, y pueden haber participantes maliciosos. Basado en el problema de los Generales Bizantinos.
- Amenaza: “causa potencial de un incidente no deseado, el cual puede ocasionar daño a un sistema o a una organización” (Asociación Española de Normalización y Certificación, 2014).
- Ataque: “tentativa de destruir, exponer, alterar, robar o acceder sin autorización o hacer uso no autorizado de un activo” (Asociación Española de Normalización y Certificación, 2014).
- Caja de Botín (loot box): Ítem que al ser abierto revela aleatoriamente otros ítems. En algunos casos, el juego puede revelar cierta información probabilista. Su venta ha sido señalada como una forma de apuesta (Perks, 2016). En inglés se le conoce como *loot box*.
- Conmutador: Equivalente al término en inglés *switch*.
- Encaminador: Equivalente al término en inglés *router*.
- Estado del juego: información necesaria para describir el juego en un instante determinado (Baughman et al, 2007)
- Frescura: que se generó luego del comienzo de la ejecución actual del protocolo. (Ling & Chen, 2012). Equivalente al término en inglés *freshness*. Se contrapone a la ranciedad o *staleness*.
- *Hash*: es una función que toma una entrada de largo variable y devuelve un valor pseudo-aleatorio de largo fijo (Burnett & Paine, 2001).
- Juego de adivinanza: aquellos cuyo objetivo es adivinar un Secreto. Equivalente al término en inglés *guessing game*.
- Juego de cartas coleccionables: Equivalente al término en inglés *Trading Card Games*, o *Collectible Card Games*.
- Juego de disparos en primera persona (*FPS*): juego de disparos donde el punto de vista del jugador es a través de los ojos de su avatar. Equivalente al término en inglés *First Person Shooter*.

- Juego Multijugador Masivo (*MMOG*): Equivalente al término en inglés *Massively Multiplayer Online Game*.
- Marca de tiempo: Equivalente al término en inglés *Time stamp*.
- MITM: Ataques de interceptación. Del inglés: Man in the Middle (“hombre en el medio”).
- Navegación por estima: Término de origen naval. Es una técnica para aproximar la ubicación de los jugadores ante una desincronización. A partir de un punto anterior, y conociendo el vector de movimiento previo, se estima la posición actual. (Instituto Superior de Navegación Darío Fernández, 2018). Equivalente al término en inglés *dead reckoning*.
- Niebla de guerra: Equivalente al término en inglés *Fog of war*. Se refiere originalmente a la incertidumbre y falta de información experimentada por los participantes en una operación militar. En los juegos, se refiere a la incertidumbre que tienen los jugadores sobre ciertos elementos del estado de juego actual.
- Nonce:
 - Número único que se utiliza una sola vez en un protocolo, y que no debe repetirse (Mollin, 2007).
 - Número aleatorio que se envía a alguien para que lo firme y así demuestre su identidad (Burnett & Paine, 2001).
- Resistente a trampas: que previene trampas o las detecta. Equivalente al término en inglés *cheat-proof*.
- *Salt*: valor aleatorio que se concatena a una contraseña antes de aplicarle una función hash con la finalidad de evitar ataques de precomputación o de tablas arcoiris (Burnett & Paine, 2001).
- Tabla arco iris: Permite reversar funciones *hash* criptográficas. Cada registro consta de una palabras o contraseñas común, y su valor *hash* correspondiente. De esta forma, dado un valor hash, si éste se encuentra en la tabla, se puede obtener una preimagen. Equivalente al término en inglés *Rainbow table*.
- Trampa: acciones que dan una ventaja a un jugador o grupo de jugadores, e infringen las normas del juego. Equivalente al término en inglés *cheat*.
- Tramposo: jugador que ejecuta una trampa. Equivalente al término en inglés *cheater*.

- Vulnerabilidad: “debilidad de un activo o control que puede ser explotado por una o más amenazas” (Asociación Española de Normalización y Certificación, 2014).

Referencias Bibliográficas

- 99 Bitcoins (2019). *Bitcoin Mining Hardware Reviews and Comparison*. Recuperado 01 de mayo de 2019 desde: <https://99bitcoins.com/bitcoin-mining/bitcoin-mining-hardware/>
- Abadi, M., & Needham, R. (1994). *Prudent engineering practice for cryptographic protocols*. En Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on (pp. 122-136). IEEE.
- Armerding, T. (2018). *The 17 biggest data breaches of the 21st century*. Recuperado 16 de abril de 2018, desde: <https://www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html>
- Asociación Española de Normalización y Certificación (2014). *Tecnología de Información – Técnicas de seguridad – Sistema de Gestión de Seguridad de la Información (SGSI) – Visión de conjunto y vocabulario: UNE-ISO/IEC 27000*. Madrid: AENOR.
- Baughman, N. E., Liberatore, M., & Levine, B. N. (2007). *Cheat-proof payout for centralized and peer-to-peer gaming*. IEEE/ACM Transactions on Networking (ToN), 15(1), 1-13.
- Bhargavan, K., Fournet, C., & Kohlweiss, M. (2016). *mitls: Verifying protocol implementations against real-world attacks*. IEEE Security & Privacy, 14(6), 18-25.
- Bitansky, N., Canetti, R., Chiesa, A., & Tromer, E. (2012). *From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again*. En *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (pp. 326-349). ACM.
- Blum, M. (1983). *Coin flipping by telephone a protocol for solving impossible problems*. ACM SIGACT News, 15(1), 23-27.
- Bond, A. (2018). *With Entrepreneurs Freaking Out After Facebook's Drastic Newsfeed Change, Here Are 4 Things You Can Do Right Now to Survive*. Recuperado 01 de mayo de 2019 desde: <https://www.entrepreneur.com/article/307505>
- Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E. & Yergeau, F. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C. Recuperado 1 de diciembre de 2018, desde: <https://www.w3.org/TR/REC-xml/>
- Bray, T. (editor). (2017). *The JavaScript Object Notation (JSON) Data Interchange Format*. Internet Engineering Task Force. RFC 8259. Recuperado 1 de diciembre de 2018, desde: <https://tools.ietf.org/html/rfc8259>

- Brooker, K. (2018, agosto). *"I WAS DEVASTATED": TIM BERNERS-LEE, THE MAN WHO CREATED THE WORLD WIDE WEB, HAS SOME REGRETS*. Vanity Fair. Recuperado 12 de julio de 2018, desde <https://www.vanityfair.com/news/2018/07/the-man-who-created-the-world-wide-web-has-some-regrets>
- Burnett, S. & Paine, S. (2001). RSA Security's Official Guide to Cryptography. Doi: 10.1036/0072192259
- Callas, J., Donnerhake, L., Finney, H., Shaw, D. & Thayer, R. (2007). *OpenPGP Message Format*. Internet Engineering Task Force. RFC 4880. Recuperado 1 de diciembre de 2018, desde <https://tools.ietf.org/html/rfc4880>
- Chappell, W. & Tsiulcas, A. (2018). *YouTube, Apple and Facebook Ban Infowars, Which Decries 'Mega Purge'*. Recuperado 01 de mayo de 2019, desde <https://www.npr.org/2018/08/06/636030043/youtube-apple-and-facebook-ban-infowars-which-decries-mega-purge>
- Chambers, C., Feng, W. C., Feng, W. C., & Saha, D. (2005). *Mitigating information exposure to cheaters in real-time strategy games*. En Proceedings of the international workshop on Network and operating systems support for digital audio and video (pp. 7-12). ACM.
- Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. & Polk, W. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Internet Engineering Task Force. RFC 5280. Recuperado 1 de diciembre de 2018, desde: <https://tools.ietf.org/html/rfc5280>
- Cronin, E., Filstrup, B. & Jamin, S. (2003). *Cheat-proofing dead reckoned multiplayer games*. En Proceedings of the 2nd International Conference on Application and Development of Computer Games (ADCOG '03), Hong Kong, January 2003.
- Damgård I. (1999) *Commitment Schemes and Zero-Knowledge Protocols*. En su: *Lectures on Data Security*. EEF School 1998. Lecture Notes in Computer Science, vol 1561. Springer, Berlin, Heidelberg. Doi: 10.1007/3-540-48969-X_3
- El Mercurio (2018). *Industria de videojuegos facturará US\$ 220 millones en 2018 en el país*. Recuperado 1 de mayo de 2019, desde: <http://www.economiaynegocios.cl/noticias/noticias.asp?id=523308>
- Entertainment Software Association (2016). *ESSENTIAL FACTS ABOUT THE COMPUTER AND VIDEO GAME INDUSTRY*. Recuperado 16 de abril de 2018, desde: https://www.isfe.eu/sites/isfe.eu/files/attachments/esa_ef_2016.pdf

- Fan, L., Trinder, P., & Taylor, H. (2010). *Design issues for peer-to-peer massively multiplayer online games*. International Journal of Advanced Media and Communication, 4(2), 108-125.
- Gartenberg, C. (2017). *China's new law forces Dota, League of Legends, and other games to reveal odds of scoring good loot*. Recuperado 16 de abril de 2018, desde: <https://www.theverge.com/2017/5/2/15517962/china-new-law-dota-league-of-legends-odds-loot-box-random>
- Geigner, T. (2016a). *The Future Is Now: Cheating In Online Games Leads To Arrests In Japan*. Recuperado 16 de abril de 2018, desde: <https://www.techdirt.com/articles/20140625/08443327682/future-is-now-cheating-online-games-leads-to-arrests-japan.shtml>
- Geigner, T. (2016b). *South Korea To Tackle Video Game Cheating By Criminalizing Breaking A Game's ToS*. Recuperado 16 de abril de 2018, desde: <https://www.techdirt.com/articles/20161205/06575436190/south-korea-to-tackle-video-game-cheating-criminalizing-breaking-games-tos.shtml>
- Griffin, A. (2018). *Facebook's latest data scandal is just the beginning – and not even the worst of it, warn privacy experts*. Recuperado 16 de abril de 2018, desde: <https://www.independent.co.uk/life-style/gadgets-and-tech/news/facebook-cambridge-analytica-latest-data-breach-privacy-explained-a8265601.html>
- Hasbro Inc. (2015a). *Clásico juego Adivina Quién?*. Recuperado 16 de abril de 2018, desde: <https://www.hasbro.com/es-mx/product/guess-who-classic-game:7DEC61D9-5056-9047-F55F-FC686C17D23F>
- Hasbro Inc. (2015b). *BATTLESHIP JUEGO DE VIAJE*. Recuperado 16 de abril de 2018, desde: <https://www.hasbro.com/es-mx/product/battleship-grab-and-go-game:0FB0F372-5056-9047-F58E-006516715764>
- Hickey, A. (2017). *It's all just fun and games...until it's not*. Recuperado 16 de abril de 2018, desde: <https://www.csoonline.com/article/3224508/network-security/the-three-ss-of-online-gaming.html>
- Instituto Superior de Navegación Darío Fernández (2018). *Curso de Timonel: navegación por estima (clase 46)*. Recuperado 12 de junio de 2018, desde: <http://www.isndf.com.ar/navegacion-por-estima/>

- Johannesson, P., & Perjons, E. (2014). *An introduction to design science*. Springer. Doi: 10.1007/978-3-319-10632-8
- Jones, M., Bradley, J. & Sakimura, N. (2015a). *JSON Web Signature (JWS)*. Internet Engineering Task Force. RFC 7515. Recuperado 1 de diciembre de 2018, desde: <https://tools.ietf.org/html/rfc7515>
- Jones, M., Bradley, J. & Sakimura, N. (2015b). *JSON Web Token (JWT)*. Internet Engineering Task Force. RFC 7519. Recuperado 1 de diciembre de 2018, desde: <https://tools.ietf.org/html/rfc7519>
- Katz, J. & Lindell, Y. (2015). *Hash Functions and Applications*. En *Introduction to Modern Cryptography*, 2nd edn., (pp. 153 – 192). Boca Raton, FL: CRC Press.
- Lee, H., Kozłowski, E., Lenker, S., & Jamin, S. (2003). *Multiplayer game cheating prevention with pipelined lockstep protocol*. En *Entertainment Computing* (pp. 31-39). Springer, Boston, MA.
- Ley N.º 19.628 (1999, 28 de agosto) sobre Protección de la Vida Privada. Recuperado 22 de julio de 2018, desde: <https://www.leychile.cl/Navegar?idNorma=141599>
- Ley N.º 19.799 (2002, 25 de marzo) sobre Documentos Electrónicos, Firma Electrónica y servicios de certificación de dicha firma. Recuperado 1 de diciembre de 2018, desde: <https://www.leychile.cl/Navegar?idNorma=196640>
- Ling, D., & Chen, K. (2012). *Background of Cryptographic Protocols*. En: *Cryptographic Protocol: Security Analysis Based on Trusted Freshness*. Springer, Berlin, Heidelberg. Doi: 10.1007/978-3-642-24073-7_2
- Litpak, A. (2019). *Facebook says that it removed 1.5 million videos of the New Zealand mass shooting*. Recuperado 01 de mayo de 2019 desde: <https://www.theverge.com/2019/3/17/18269453/facebook-new-zealand-attack-removed-1-5-million-videos-content-moderation>
- Macbeth, G., Razumiejczyk, E., Crivello, M. del C., Bolzán, C., Pereyra Girardi, C. I., & Campitelli, G. (2014). *Mental Models for the Negation of Conjunctions and Disjunctions*. *Europe's Journal of Psychology*, 10(1), 135–149. Doi:10.5964/ejop.v10i1.696
- Martineau, P. (2018). *Tumblr's Porn Ban Reveals Who Controls What We See Online*. Recuperado 01 de mayo de 2019 desde: <https://www.wired.com/story/tumblrs-porn-ban-reveals-controls-we-see-online/>

- McCormick, R. (2014). *Hack leaks hundreds of nude celebrity photos*. Recuperado 01 de mayo de 2019 desde: <https://www.theverge.com/2014/9/1/6092089/nude-celebrity-hack>
- McConnell, S. (1996). *Rapid Development: Taming Wild Software Schedules*. Microsoft Press.
- Mollin, R. (2007). *An Introduction To Cryptography* (2da ed.). Boca Ratón: Chapman & Hall/CRC.
- Montes, C. (2019). *¿Cómo se visualiza el mercado nacional de videojuegos para el 2019?*. Recuperado 1 de mayo de 2019, desde: <https://www.latercera.com/que-pasa/noticia/se-visualiza-mercado-nacional-videojuegos-2019/524029/>
- N. V., P. (2019). *Ethereum [ETH] ProgPow: ASIC resistance is a myth and a fallacy, says developer*. Recuperado 01 de mayo de 2019 desde: <https://ambcrypto.com/ethereum-eth-progpow-asic-resistance-is-a-myth-and-a-fallacy-says-developer/>
- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Recuperado 01 de mayo de 2019 desde: <https://bitcoin.org/bitcoin.pdf>
- National Institute of Standards and Technology (2015). *Secure Hash Standard (SHS)*. FIPS PUB 180-4. Doi: 10.6028/NIST.FIPS.180-4
- Neumann, C., Prigent, N., Varvello, M., & Suh, K. (2007). *Challenges in peer-to-peer gaming*. ACM SIGCOMM Computer Communication Review, 37(1), 79-82.
- Perks, M. (2016). *Limited Edition Loot Boxes: Problematic Gambling and Monetization*. Recuperado 12 de Junio de 2018, desde: <https://medium.com/the-cube/limited-edition-loot-boxes-problematic-gambling-and-monetization-756819f2c54f>
- Pittman, D., & GauthierDickey, C. (2013). *Match+ Guardian: a secure peer-to-peer trading card game protocol*. Multimedia systems, 19(3), 303-314.
- Prather, J., Nix, R., & Jessup, R. (2017). *Trust management for cheating detection in distributed massively multiplayer online games*. En Network and Systems Support for Games (NetGames), 2017 15th Annual Workshop on (pp. 1-3). IEEE.
- Publimetro (2018). *El despegue mundial de los videojuegos "made in Chile"*. Recuperado 1 de mayo de 2019, desde: <https://www.publimetro.cl/cl/noticias/2018/09/03/la-industria-del-videojuego-made-in-chile.html>
- Quintero, B. (2018). *La ascendente industria chilena de videojuegos*. Recuperado 1 de mayo de 2019, desde: <https://www.pauta.cl/negocios/la-ascendente-industria-chilena-de-videojuegos>

- Rice, S. L. (2014). *Secure Map Generation for Multiplayer, Turn-Based Strategy Games* (Tesis Doctoral, University of Denver). Recuperado 09 de mayo de 2018 desde: <https://digitalcommons.du.edu/cgi/viewcontent.cgi?article=1547&context=etd>
- Silva, M. V. M., & Simplicio, M. A. (2017). *A secure protocol for exchanging cards in P2P trading card games based on transferable e-cash*. IEEE Latin America Transactions, 15(7), 1286-1294.
- Stevens, M. (2007). *On Collisions for MD5* (Tesis de Magister, Eindhoven University of Technology). Recuperado 20 de abril de 2019 desde: [https://www.win.tue.nl/hashclash/On Collisions for MD5 – M.M.J. Stevens.pdf](https://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf)
- Stevens, M., Lenstra, A. & de Weger, B. (2007). *Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*. Recuperado 20 de abril de 2019 desde: [https://www.win.tue.nl/hashclash/On Collisions for MD5 – M.M.J. Stevens.pdf](https://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf)
- Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D. A., & De Weger, B. (2009, Agosto). *Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate*. En *Annual International Cryptology Conference* (pp. 55-69). Springer, Berlin, Heidelberg. Doi: 10.1007/978-3-642-03356-8_4
- Stoughton, A., Johnson, A., Beller, S., Chadha, K., Chen, D., Foner, K., & Zhivich, M. (2014, July). *You sank my battleship!: A case study in secure programming*. En *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security* (p. 2). ACM.
- Sweigart, A. (2015). *Inventa tus propios juegos de computadora con Python*. (Trad. A. Carella, A. Pernin & F. Palm) (3a ed.). (Original en inglés, 2008). Recuperado 22 de mayo de 2018 desde: https://inventwithpython.com/es/InventarConPython_3a_es.pdf
- Tabora, V. (2018). *The Evolution of the Internet, From Decentralized to Centralized*. Recuperado 01 de mayo de 2019 desde: <https://hackernoon.com/the-evolution-of-the-internet-from-decentralized-to-centralized-3e2fa65898f5>
- Techkle (2018). *Top 10 Gaming Laptops for Mining in 2018*. Recuperado 01 de mayo de 2019 desde: <https://techkle.com/top-gaming-laptops-mining/>
- Ultra BoardGames (2019). *Battleship Salvo Game Rules*. Recuperado 01 de mayo de 2019 desde : <https://www.ultraboardgames.com/battleship/salvo-rules.php>
- Unión Internacional de Telecomunicaciones (1995). *TECNOLOGÍA DE LA INFORMACIÓN – INTERCONEXIÓN DE SISTEMAS ABIERTOS – MODELO DE REFERENCIA BÁSICO: EL MODELO BÁSICO: UIT-T X.200 ISO/IEC 7498-1*.

- Unión Internacional de Telecomunicaciones (2003). *UIT-T X.805 Arquitectura de seguridad para sistemas de comunicaciones extremo a extremo - SERIE X: REDES DE DATOS Y COMUNICACIÓN ENTRE SISTEMAS ABIERTOS – Seguridad*.
- van Saberhagen, N. (2013). *CryptoNote v 2.0*. Recuperado 01 de mayo de 2019 desde: <https://cryptonote.org/whitepaper.pdf>
- Wilmoth, J. (2018). *Manufacturer Holds Cryptonight ASIC Firesale after Monero Hard Forks*. Recuperado 01 de mayo de 2019 desde: <https://ca.finance.yahoo.com/news/manufacturer-holds-cryptonight-asic-firesale-200840436.html>
- Wizkid (1997). THE GAME OF TWENTY QUESTIONS. Recuperado 01 de mayo de 2019 desde: http://barelybad.com/20_questions.htm
- Wohlin, C. (2014). *Guidelines for snowballing in systematic literature studies and a replication in software engineering*. In Proceedings of the 18th international conference on evaluation and assessment in software engineering (p. 38). ACM.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.
- Yahyavi, A., & Kemme, B. (2013). *Peer-to-peer architectures for massively multiplayer online games: A survey*. ACM Computing Surveys (CSUR), 46(1), 9.
- Yan, J., & Randell, B. (2009). *An investigation of cheating in online games*. IEEE Security & Privacy, 7(3).
- Yan, J. (2003). *Security design in online games*. En Computer Security Applications Conference, 2003. Proceedings. 19th Annual (pp. 286-295). IEEE.

Anexo A: Diagramas de estrategias antitrampas

En la Figura B.1 Estado de juego duplicado y arquitectura descentralizada, se puede observar el esquema de un juego de red donde dos clientes se conectan directamente sin que se aplique ninguna medida antitrampa. En este caso se confía en que el adversario no ha alterado el cliente ni su entorno.

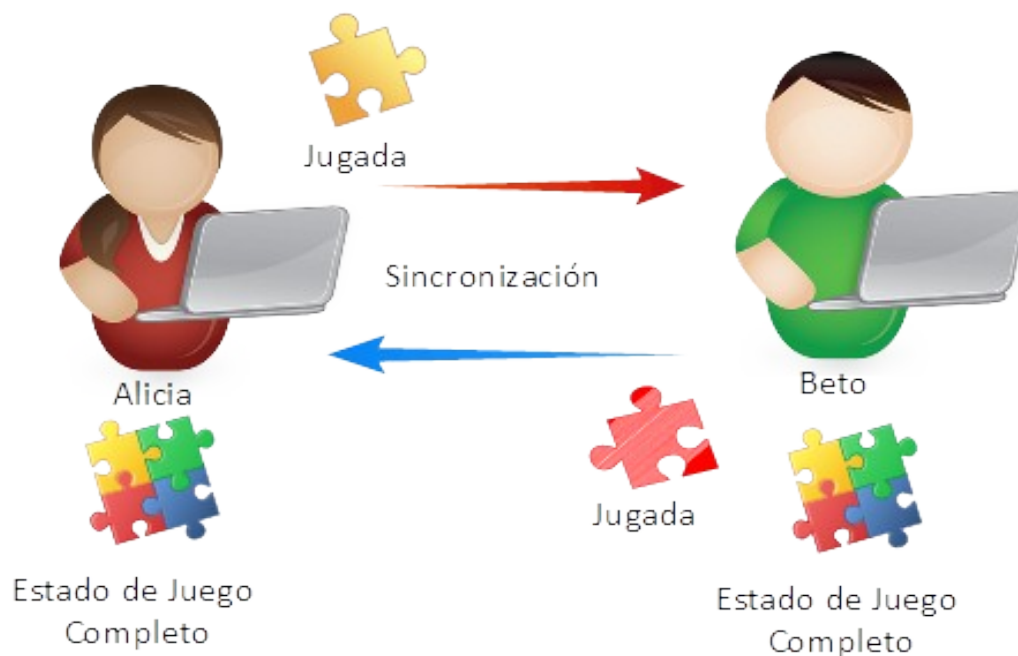


Figura A.1: Estado de juego duplicado y arquitectura descentralizada.

Fuente: Elaboración Propia, 2018.

En la Figura B.2 Estado de Juego duplicado y ofuscado con arquitectura descentralizada, se puede observar la aplicación de medidas en el cliente para evitar que los jugadores lean o modifiquen indebidamente el Estado de Juego. Acá se confía en que las medidas aplicadas son suficientemente robustas para mantener a raya al adversario.

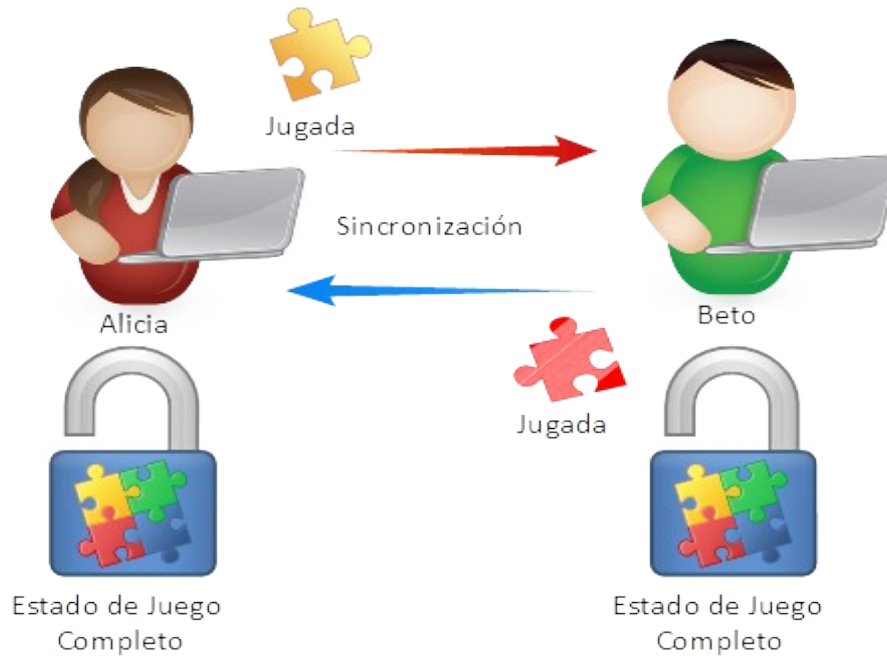


Figura A.2: Estado de Juego duplicado y ofuscado con arquitectura descentralizada.

Fuente: Elaboración Propia, 2018.

En la Figura B.3 Estado de Juego único y arquitectura cliente-servidor centralizada, se mueve la confianza a un servidor, que sirve como tercero de confianza. Éste se encarga de resguardar una única copia del Estado de Juego, y de aplicar las jugadas. Como todo el tráfico pasa por el servidor, los jugadores sólo tienen acceso a la información que el servidor les entregue, y sólo pueden hacer las jugadas que éste autorice. Ambos jugadores confían en la imparcialidad y seguridad del servidor.

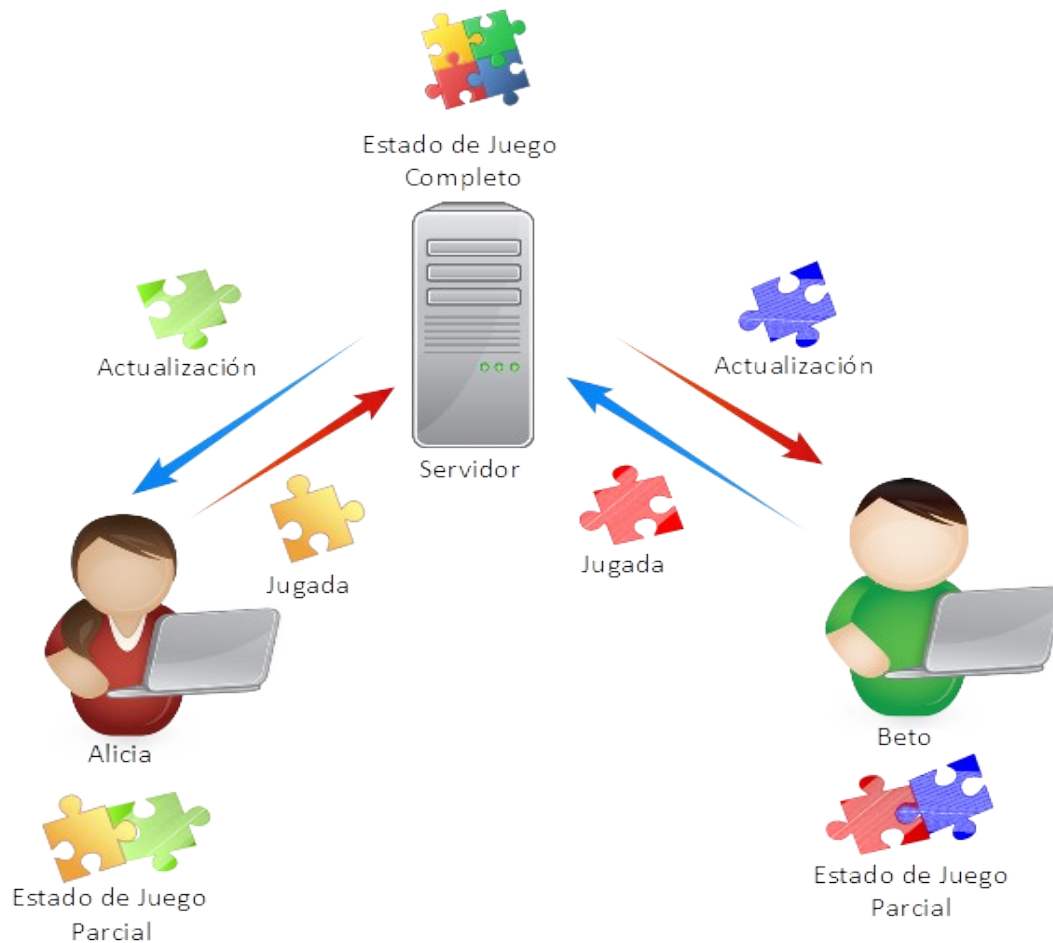


Figura A.3: Estado de Juego único y arquitectura cliente-servidor centralizada.

Fuente: Elaboración Propia, 2018.

En la Figura B.4 Estado de Juego único y arquitectura cliente-servidor descentralizada, los clientes se envían las jugadas directamente entre sí. Esto tiene como beneficio que se reduce el uso de ancho de banda del servidor. Sin embargo, el servidor debe seguir recibiendo las jugadas de los jugadores, para mantener actualizado el Estado de Juego, que sigue a cargo de resguardar. Para hacer efectivo este resguardo, el servidor envía a los clientes la mensajes de control o coordinación a fin de corregir cualquier inconsistencia que se presente en el juego.

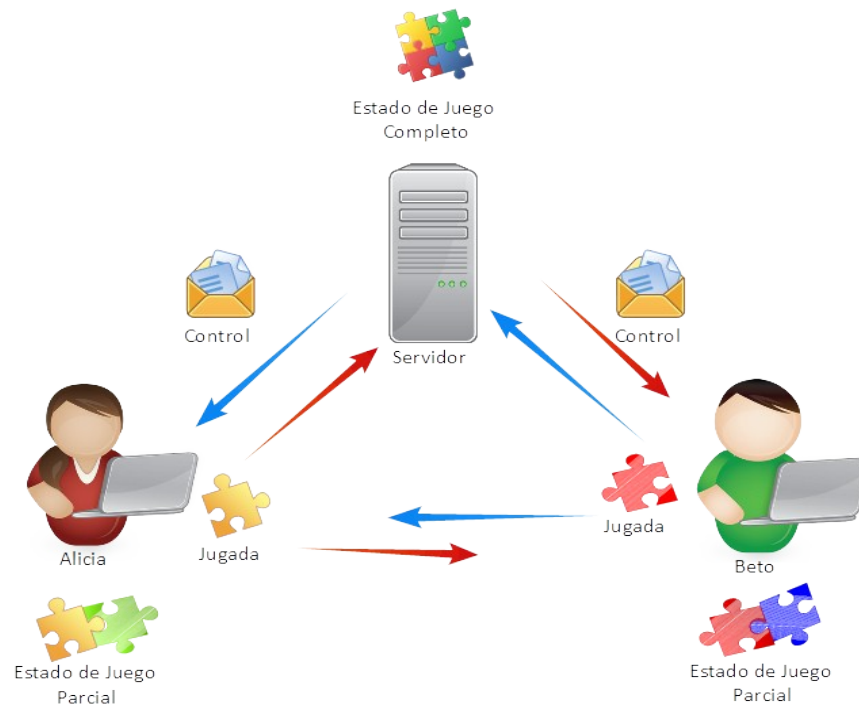


Figura A.4: Estado de Juego único y arquitectura cliente-servidor descentralizada.

Fuente: Elaboración Propia, 2018.

En la Figura B.5 Estado de Juego dividido y arquitectura completamente descentralizada, cada cliente mantiene una parte del Estado de Juego, sin que exista una copia consolidada de los fragmentos. Sobre la base de sus jugadas, cada cliente determina las actualizaciones a enviar a los demás jugadores. Deben usarse mecanismos de control que eviten jugadas ilegales, que los jugadores retrasen su jugada para contar con mayor información que sus pares, que se pierda la consistencia entre los fragmentos del Estado de Juego.

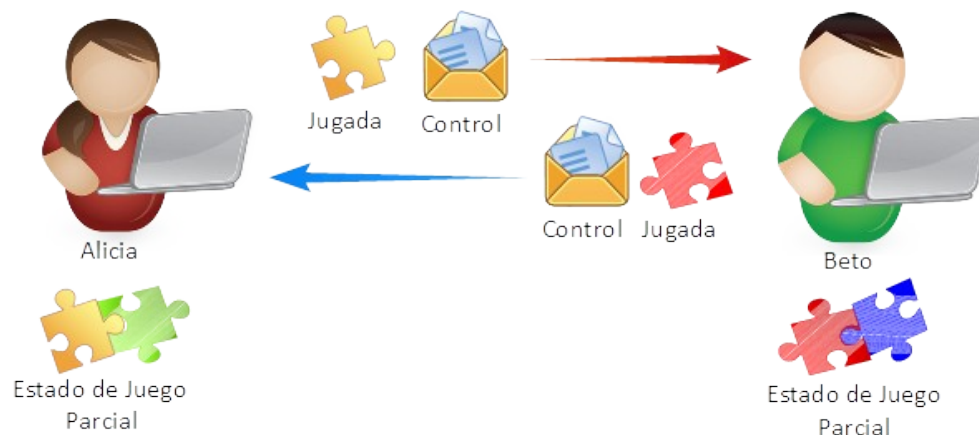


Figura A.5: Estado de Juego dividido y arquitectura completamente descentralizada.

Fuente: Elaboración Propia, 2018.

Anexo B: Mensajes de partidas

A continuación se presenta un extracto de los mensajes intercambiados durante la fase de evaluación del prototipo. Se seleccionó la palabra “execute”, ya que corresponde a una partida breve, y que contiene intentos acertados y fallidos. Sólo se muestra el contenido JSON de los mensajes, es decir, se omite el prefijo numérico con el largo del mensaje, y la firma PGP: En la Tabla B.1 Mensajes de la partida "execute" (95), corrida 0. Esta es la corrida sin ninguna medida de seguridad, con la menor cantidad de mensajes, para un total de 29.

Tabla B.1: Mensajes de la partida "execute" (95), corrida 0

| Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 0 | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| <code>{"type": "lets_play", "min_lives": 5, "max_lives": 10, "supported_lang": ["eng"], "supported_security": [0], "supported_algo": ["none"]}</code> | |
| <code>{"type": "accept_play", "total_lives": 10, "lang": "eng", "security": 0, "algo": "none"}</code> | <code>{"type": "play_ack"}</code> |
| <code>{"type": "make_a_try", "status": "_____", "lives_left": 10}</code> | <code>{"type": "new_try", "guess": "e"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_e___e", "guess": "e"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_e___e", "lives_left": 10}</code> | <code>{"type": "new_try", "guess": "t"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_e__te", "guess": "t"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_e__te", "lives_left": 10}</code> | <code>{"type": "new_try", "guess": "a"}</code> |
| <code>{"type": "reply", "result": false, "status": "e_e__te", "guess": "a"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_e__te", "lives_left": 9}</code> | <code>{"type": "new_try", "guess": "c"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_ec_te", "guess": "c"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_ec_te", "lives_left": 9}</code> | <code>{"type": "new_try", "guess": "u"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_ecute", "guess": "u"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_ecute", "lives_left": 9}</code> | <code>{"type": "new_try", "guess": "x"}</code> |
| <code>{"type": "reply", "result": true, "status": "execute", "guess": "x"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "game_ended", "secret": "execute", "result": true}</code> | <code>{"type": "quit"}</code> |

Fuente: Elaboración Propia, 2018.

En la Tabla B.2 Mensajes de la partida "execute" (95), corrida 1, se pueden observar los mensajes de la partida en el siguiente nivel de seguridad, la implementación de Chambers et al (2005). El número de mensajes sube de 29 a 33, ya que incluye los mensajes de control w_commit, w_ack, result_validation, y result_ack.

Tabla B.2: Mensajes de la partida "execute" (95), corrida 1

| Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 1 | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <code>{"type": "lets_play", "min_lives": 5, "max_lives": 10, "supported_lang": ["eng"], "supported_security": [1], "supported_algo": ["sha256"]}</code> | |
| <code>{"type": "accept_play", "total_lives": 10, "lang": "eng", "security": 1, "algo": "sha256"}</code> | <code>{"type": "play_ack"}</code> |
| <code>{"type": "w_commit", "length": 1, "my_word_commitment_vector": ["522c07fd6152f49e04a0002e799d49e63490a023af81ce86ed5e6062cf48bbfc"]}</code> | <code>{"type": "w_ack"}</code> |
| <code>{"type": "make_a_try", "status": "_____", "lives_left": 10}</code> | <code>{"type": "new_try", "guess": "e"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_e___e", "guess": "e"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_e___e", "lives_left": 10}</code> | <code>{"type": "new_try", "guess": "t"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_e___te", "guess": "t"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_e___te", "lives_left": 10}</code> | <code>{"type": "new_try", "guess": "a"}</code> |
| <code>{"type": "reply", "result": false, "status": "e_e___te", "guess": "a"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_e___te", "lives_left": 9}</code> | <code>{"type": "new_try", "guess": "c"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_ec_te", "guess": "c"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_ec_te", "lives_left": 9}</code> | <code>{"type": "new_try", "guess": "u"}</code> |
| <code>{"type": "reply", "result": true, "status": "e_ecute", "guess": "u"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "make_a_try", "status": "e_ecute", "lives_left": 9}</code> | <code>{"type": "new_try", "guess": "x"}</code> |
| <code>{"type": "reply", "result": true, "status": "execute", "guess": "x"}</code> | <code>{"type": "r_ack"}</code> |
| <code>{"type": "game_ended", "secret": "execute", "result": true, "length": 1, "my_word_nonce_vector": [8557]}</code> | <code>{"type": "result_validation", "result": true}</code> |
| <code>{"type": "result_ack"}</code> | <code>{"type": "quit"}</code> |

Fuente: Elaboración Propia, 2018.

En la Tabla B.3 Mensajes de la partida "execute" (95), corrida 2; se pueden observar los mensajes de la partida en el nivel de seguridad mínimo del protocolo propuesto en este trabajo. En este nivel se incluyen los mensajes de control de la corrida anterior: w_commit, w_ack, result_validation, y result_ack. Adicionalmente, se incluyen los mensajes necesarios para la generación distribuida de *nonces*: nonce_start, dos mensajes r_commit, y r_reveal. Así, el número de mensajes sube de 33 a 37. Por otro lado, se puede observar que los mensajes son más largos, ya que incluyen campos de contexto.

Tabla B.3: Mensajes de la partida "execute" (95), corrida 2

| Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 2 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>{"type": "lets_play", "unix_time": 1561344176, "my_address": "arthur@tesis", "my_session_id": 6697, "min_lives": 5, "max_lives": 10, "supported_lang": ["eng"], "supported_security": [2], "supported_algo":</code> |

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 2

```
["sha256"]
{"type": "accept_play", "unix_time": 1561344177, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "total_lives": 10, "lang": "eng", "security": 2,
"algo": "sha256"}
{"type": "play_ack", "unix_time": 1561344178, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939}
{"type": "nonce_start", "unix_time": 1561344178, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "length": 1}
{"type": "r_commit", "unix_time": 1561344179, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "length": 1, "my_random_commitment_vector":
["84cfe11b4ce03609842cbe56549ae64c01f23a52b3d213921595e68a65d5625e"]}
{"type": "r_commit", "unix_time": 1561344179, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "length": 1, "my_random_commitment_vector":
["be076481247fc7c1c3c2e3facd52ad5ce842a10005424bda00baed802581cb8e"]}
{"type": "r_reveal", "unix_time": 1561344180, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "length": 1, "my_random_vector": [5080],
"my_random_nonce_vector": [9969]}
{"type": "w_commit", "unix_time": 1561344180, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "length": 1, "my_word_commitment_vector":
["99ed9fa6f2ea13c59f104d1f56f7c4fba773486d9f457357e07ccd02dabf1907"]}
{"type": "w_ack", "unix_time": 1561344181, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939}
{"type": "make_a_try", "unix_time": 1561344181, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 1, "status": "_____", "lives_left": 10}
{"type": "new_try", "unix_time": 1561344183, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 1, "guess": "e"}
{"type": "reply", "unix_time": 1561344183, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 1, "result": true, "status": "e_e__e",
"guess": "e"}
{"type": "r_ack", "unix_time": 1561344184, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 1}
{"type": "make_a_try", "unix_time": 1561344184, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 2, "status": "e_e__e", "lives_left": 10}
{"type": "new_try", "unix_time": 1561344185, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 2, "guess": "t"}
{"type": "reply", "unix_time": 1561344185, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 2, "result": true, "status": "e_e__te",
"guess": "t"}
{"type": "r_ack", "unix_time": 1561344186, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 2}
{"type": "make_a_try", "unix_time": 1561344186, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 3, "status": "e_e__te", "lives_left": 10}
{"type": "new_try", "unix_time": 1561344187, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 3, "guess": "a"}
{"type": "reply", "unix_time": 1561344187, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 3, "result": false, "status": "e_e__te",
"guess": "a"}
{"type": "r_ack", "unix_time": 1561344188, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 3}
{"type": "make_a_try", "unix_time": 1561344188, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 4, "status": "e_e__te", "lives_left": 9}
{"type": "new_try", "unix_time": 1561344189, "my_address": "arthur@tesis", "my_session_id": 6697,
"your_address": "merlin@tesis", "your_session_id": 1939, "round": 4, "guess": "c"}
{"type": "reply", "unix_time": 1561344189, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 4, "result": true, "status": "e_ec__te",
"guess": "c"}
{"type": "r_ack", "unix_time": 1561344190, "my_address": "arthur@tesis", "my_session_id": 6697,
```

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 2

```
        "your_address": "merlin@tesis", "your_session_id": 1939, "round": 4}
{"type": "make_a_try", "unix_time": 1561344190, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 5, "status": "e_ec_te", "lives_left": 9}
    {"type": "new_try", "unix_time": 1561344191, "my_address": "arthur@tesis", "my_session_id": 6697,
        "your_address": "merlin@tesis", "your_session_id": 1939, "round": 5, "guess": "u"}
{"type": "reply", "unix_time": 1561344191, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 5, "result": true, "status": "e_ecute",
"guess": "u"}
    {"type": "r_ack", "unix_time": 1561344192, "my_address": "arthur@tesis", "my_session_id": 6697,
        "your_address": "merlin@tesis", "your_session_id": 1939, "round": 5}
{"type": "make_a_try", "unix_time": 1561344192, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 6, "status": "e_ecute", "lives_left": 9}
    {"type": "new_try", "unix_time": 1561344193, "my_address": "arthur@tesis", "my_session_id": 6697,
        "your_address": "merlin@tesis", "your_session_id": 1939, "round": 6, "guess": "x"}
{"type": "reply", "unix_time": 1561344193, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "round": 6, "result": true, "status": "execute",
"guess": "x"}
    {"type": "r_ack", "unix_time": 1561344194, "my_address": "arthur@tesis", "my_session_id": 6697,
        "your_address": "merlin@tesis", "your_session_id": 1939, "round": 6}
{"type": "game_ended", "unix_time": 1561344194, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697, "secret": "execute", "result": true, "length": 1,
"my_random_vector": [6292], "my_random_nonce_vector": [3483]}
    {"type": "result_validation", "unix_time": 1561344195, "my_address": "arthur@tesis", "my_session_id":
        6697, "your_address": "merlin@tesis", "your_session_id": 1939, "result": true}
{"type": "result_ack", "unix_time": 1561344195, "my_address": "merlin@tesis", "my_session_id": 1939,
"your_address": "arthur@tesis", "your_session_id": 6697}
    {"type": "quit", "unix_time": 1561344196, "my_address": "arthur@tesis", "my_session_id": 6697,
        "your_address": "merlin@tesis", "your_session_id": 1939}
```

Fuente: Elaboración Propia, 2018.

En la Tabla B.4 Mensajes de la partida "execute" (95), corrida 3; se pueden observar los mensajes de la partida en el nivel de seguridad intermedio del protocolo propuesto en este trabajo. En este nivel se incluyen los mensajes de control de la corrida. Así, el número de mensajes se mantiene en 37. De la misma forma, incluye los mismos campos de contexto que el nivel anterior. Sin embargo, en el nivel mínimo se generó un solo nonce, mientras que en esta corrida se generaron siete. Por otro lado, el intento de la tercera ronda es fallido, por lo cual no tiene revelación, mientras que los cinco intentos acertados sí cuentan con revelaciones.

Tabla B.4: Mensajes de la partida "execute" (95), corrida 3

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 3

```
    {"type": "lets_play", "unix_time": 1561346882, "my_address": "arthur@tesis", "my_session_id": 6371,
"min_lives": 5, "max_lives": 10, "supported_lang": ["eng"], "supported_security": [3], "supported_algo":
    ["sha256"]}
{"type": "accept_play", "unix_time": 1561346883, "my_address": "merlin@tesis", "my_session_id": 3371,
"your_address": "arthur@tesis", "your_session_id": 6371, "total_lives": 10, "lang": "eng", "security": 3,
"algo": "sha256"}
    {"type": "play_ack", "unix_time": 1561346883, "my_address": "arthur@tesis", "my_session_id": 6371,
        "your_address": "merlin@tesis", "your_session_id": 3371}
{"type": "nonce_start", "unix_time": 1561346884, "my_address": "merlin@tesis", "my_session_id": 3371,
"your_address": "arthur@tesis", "your_session_id": 6371, "length": 7}
```

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 3

```
{
  "type": "r_commit", "unix_time": 1561346884, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "length": 7, "my_random_commitment_vector":
    ["f85e0548cdee2656b21502ddd7707c07ffcba32857e4bb3884eb39d52dfdd90d",
     "a5ca512e56cedd686702f0c3b851da4089d25defb4e8bc152a2e163113608eab",
     "e666a46991fb23dedeff2f947aa4a6a1633a190378212e84213d9fa973c3d703",
     "799bc53965cdcb9a1034dc7ac40efc0aeb4b5eebabbb83de0e45a35d3c644776",
     "142942ebae3f4d0aa4ab5e13a9382789d718bfad420fbc45a23a5dcc30189dfa",
     "9a6e5049436c35c6df9c7d6bcdcf7350485ac6042989c11bb4e7a763609dc416",
     "e5735255893c507313111c4dd2451a04e178354d8610150be958ca2575f632c3"]
},
{
  "type": "r_commit", "unix_time": 1561346885, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "length": 7, "my_random_commitment_vector":
    ["02eaa66f82ac26867ac9896a9c167de9253d95e44765aabd4a54563c4e611dbb",
     "e44657eba0c646f5cbc87053d478dfef1128216275566cd925bbd9cac3ad6721",
     "8837b26748dd136747ba72da320e132c9701bf3532d8cef266fbb17cb9e10625",
     "940d2e13ff3433a286ec8a9233c9a6df338dc8076c218c333482775034cf40ea",
     "39ac4787458824cd2d667ec92c91392b413d189c813291a009f12d2c6b13fc1",
     "fa239b3816da8cd7563b7c081ef41914d4da17703ef596a1c78f43ff0174519f",
     "f440d6087e5a50c4aef861dbe0f0dd7752e43e8d6fb7b0a6393f704d359bba12"]
},
{
  "type": "r_reveal", "unix_time": 1561346885, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "length": 7, "my_random_vector": [5373, 4344,
    4526, 1287, 1827, 8388, 6796], "my_random_nonce_vector": [8575, 5438, 7732, 2023, 5560, 3154, 551]
},
{
  "type": "w_commit", "unix_time": 1561346886, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "length": 7, "my_word_commitment_vector":
    ["8f40d5a9ed7860cac45db4d5d504cff4e496f8069aebd8af147a6af54fa8f878",
     "cf48c833361610e1fdbee673077d1b029cde6fc49fe2b419d8bf1847fc66a71f",
     "a28a8f6fe56f2ce717eabcc31e9ac8c527ca99a857a939edb0aa9692f84b5d09",
     "e67a13081c285caf18015199c36129d77077324cc6a9e4ba2c21bee956b72eaa",
     "06633e4de249baf2e3818916380cd37f7d0e951b668d66a5f5d05845e7526531",
     "c1f1d7ce4fa64fd3c00199cc7b7600b354e209878c3e002f66702ace559b175b",
     "88c7b6888dac3e9c483a614d9e2e718425ddb3c0bda306b7bce929e01ae56403"]
},
{
  "type": "w_ack", "unix_time": 1561346886, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371
},
{
  "type": "make_a_try", "unix_time": 1561346887, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 1, "status": "_____", "lives_left": 10
},
{
  "type": "new_try", "unix_time": 1561346889, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 1, "guess": "e"
},
{
  "type": "reply", "unix_time": 1561346889, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 1, "result": true, "status": "e_e__e",
  "guess": "e", "partial_length": 3, "partial_indices": [0, 2, 6], "partial_random_vector": [3135, 6099, 2501],
  "partial_nonce_vector": [8855, 544, 3275]
},
{
  "type": "r_ack", "unix_time": 1561346890, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 1, "validation": true
},
{
  "type": "make_a_try", "unix_time": 1561346890, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 2, "status": "e_e__e", "lives_left": 10
},
{
  "type": "new_try", "unix_time": 1561346891, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 2, "guess": "t"
},
{
  "type": "reply", "unix_time": 1561346891, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 2, "result": true, "status": "e_e__te",
  "guess": "t", "partial_length": 1, "partial_indices": [5], "partial_random_vector": [4661],
  "partial_nonce_vector": [2122]
},
{
  "type": "r_ack", "unix_time": 1561346892, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 2, "validation": true
},
{
  "type": "make_a_try", "unix_time": 1561346892, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 3, "status": "e_e__te", "lives_left": 10
},
{
  "type": "new_try", "unix_time": 1561346893, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 3, "guess": "a"
},
{
  "type": "reply", "unix_time": 1561346893, "my_address": "merlin@tesis", "my_session_id": 3371,

```

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 3

```
"your_address": "arthur@tesis", "your_session_id": 6371, "round": 3, "result": false, "status": "e_e_te",
"guess": "a"}
{"type": "r_ack", "unix_time": 1561346894, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 3}
{"type": "make_a_try", "unix_time": 1561346894, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 4, "status": "e_e_te", "lives_left": 9}
{"type": "new_try", "unix_time": 1561346895, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 4, "guess": "c"}
{"type": "reply", "unix_time": 1561346895, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 4, "result": true, "status": "e_ec_te",
  "guess": "c", "partial_length": 1, "partial_indices": [3], "partial_random_vector": [6821],
  "partial_nonce_vector": [8305]}
{"type": "r_ack", "unix_time": 1561346896, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 4, "validation": true}
{"type": "make_a_try", "unix_time": 1561346896, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 5, "status": "e_ec_te", "lives_left": 9}
{"type": "new_try", "unix_time": 1561346897, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 5, "guess": "u"}
{"type": "reply", "unix_time": 1561346897, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 5, "result": true, "status": "e_ecute",
  "guess": "u", "partial_length": 1, "partial_indices": [4], "partial_random_vector": [1862],
  "partial_nonce_vector": [2661]}
{"type": "r_ack", "unix_time": 1561346898, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 5, "validation": true}
{"type": "make_a_try", "unix_time": 1561346898, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 6, "status": "e_ecute", "lives_left": 9}
{"type": "new_try", "unix_time": 1561346899, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 6, "guess": "x"}
{"type": "reply", "unix_time": 1561346899, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "round": 6, "result": true, "status": "execute",
  "guess": "x", "partial_length": 1, "partial_indices": [1], "partial_random_vector": [2418],
  "partial_nonce_vector": [90]}
{"type": "r_ack", "unix_time": 1561346900, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371, "round": 6, "validation": true}
{"type": "game_ended", "unix_time": 1561346900, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371, "secret": "execute", "result": true, "length": 7,
  "my_random_vector": [3135, 2418, 6099, 6821, 1862, 4661, 2501], "my_random_nonce_vector": [8855,
  90, 544, 8305, 2661, 2122, 3275]}
{"type": "result_validation", "unix_time": 1561346901, "my_address": "arthur@tesis", "my_session_id":
  6371, "your_address": "merlin@tesis", "your_session_id": 3371, "result": true}
{"type": "result_ack", "unix_time": 1561346901, "my_address": "merlin@tesis", "my_session_id": 3371,
  "your_address": "arthur@tesis", "your_session_id": 6371}
{"type": "quit", "unix_time": 1561346902, "my_address": "arthur@tesis", "my_session_id": 6371,
  "your_address": "merlin@tesis", "your_session_id": 3371}
```

Fuente: Elaboración Propia, 2018.

En la Tabla B.4 Mensajes de la partida "execute" (95), corrida 4; se pueden observar los mensajes de la partida en el nivel de seguridad máximo del protocolo propuesto en este trabajo. En este nivel se incluyen los mensajes de control de la corrida. Así, el número de mensajes se mantiene en 37. De la misma forma, incluye los mismos campos de contexto que el nivel anterior. Sin embargo, esta corrida se generaron 26 nonces, en contraposición a un nonce en el nivel mínimo, y siete nonces en el nivel intermedio. Por otro lado, los seis intentos tienen

revelaciones, tanto los exitosos como el intento fallido, porque en el nivel máximo todos los intentos reciben una revelación.

Tabla B.5: Mensajes de la partida "execute" (95), corrida 4

| Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 4 | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <pre>{ "type": "lets_play", "unix_time": 1561350001, "my_address": "arthur@tesis", "my_session_id": 6198, "min_lives": 5, "max_lives": 10, "supported_lang": ["eng"], "supported_security": [4], "supported_algo": ["sha256"] }</pre> | |
| <pre>{ "type": "accept_play", "unix_time": 1561350002, "my_address": "merlin@tesis", "my_session_id": 104, "your_address": "arthur@tesis", "your_session_id": 6198, "total_lives": 10, "lang": "eng", "security": 4, "algo": "sha256" }</pre> | |
| <pre>{ "type": "play_ack", "unix_time": 1561350003, "my_address": "arthur@tesis", "my_session_id": 6198, "your_address": "merlin@tesis", "your_session_id": 104 }</pre> | |
| <pre>{ "type": "nonce_start", "unix_time": 1561350003, "my_address": "merlin@tesis", "my_session_id": 104, "your_address": "arthur@tesis", "your_session_id": 6198, "length": 26 }</pre> | |
| <pre>{ "type": "r_commit", "unix_time": 1561350004, "my_address": "arthur@tesis", "my_session_id": 6198, "your_address": "merlin@tesis", "your_session_id": 104, "length": 26, "my_random_commitment_vector": ["933a2597d86c8e3cebec2b7d88609ddcfc4d55628a511792a597a856fc80ee", "5ff69d0b842d0ec4d99fb9ebb992e5b7dfa7c6c60768ac545dd8a99c5fc0d40f", "ffddf5e77e34f033444a20c4b950e10872916a0554b8ccfc69183362c178f8e", "7b1e6c09331a0e3c1ff6564597251464fa1d4e9db3fa70e75acd087b75c68ff1", "ba3034389d17096112316b051f75f2e365e5fd93b6dd5fef17ebe310b4d35560", "ed75ab1c1c765acee92b1af8da0f6ec9a2c4c1029b5756a6cda0cda4a3687cf6", "8deb3ba56c3a9b1414f661684a91bfdce794f013f2fc801e2d7397b51ca35e07", "4da9cf9c63b2e87d638cf47a34da78e0523c7ea62f847083f0cf2a99af289311", "0174dd71233775b0eb8ac25b6cbf35d9e64dd31099d5815a743ee1d2992cb414", "6b97ea66e4a3d1dbe5235349e4240f068b1ff786d3460f2531f9e96a17933476", "7011103c8ca5a1648c7a814a7642c5e6e056c54f171bf99074b7529c2058423e", "b84214e93457d085fd9f77c6cb9da05a18fab2afda42ce20dc0fd725bd5f1364", "a3085491575e23d27748609943ae9af0506c9a63d894153a7ca11795b561c875", "488e22c5a9bcae94cc5cab0bbaf5fa38b158e989cd6382f31b1ffbd5f23d1c8e", "9cdda4b32ce06426ad6f97b37527b21fe27673ba888272a51a023fa734e92e6", "54fa79cbc8f6ceeba961ba0a423196aea1dfd4e7773e66108120a371040161ac", "aa6bd0d8dabfc4b4005fc2271771b98adf96a33dbebf8f34b85420777a99e597", "72e5d59f74101c8a85ad2414fcf8c622652af8f3695fa942b2d201ad2b584168", "9530cf2fa825fa198bca9d991e146f3828dc855e1d130f803e284485aa1481e8", "95c8600f891c0529a7cd6bfb925a6ee2596ebf8e7151115ed4a30e3b4793f727", "96fc738852419af70faa6d06cea36240f8cb4cf2991528efeb88166661a7faf9", "ad8e2b93d40d01a5be6e1acfee4c959077badba2b5b6938b1bc7f5e5b22f9378", "8a2347fce85e6da1fdb767d0f8f948fc150dff81a0b5a63621e894fcdf2b9db6", "17fedb56286bd7824d825e519d334785b2f5f3167f1b20ef1d7c636fcb585b0d", "78f6989aff20f38a6887654c815c98c3a373346f710a4d9b88dd589a98288d9d", "0e481ef6d53bd3825fa655906be3fe72788136d28390d21ae190c4745d074c3a", "38c09f4c4aedd1a29cfb59f4a296afe91ab57dd8b04b3973ba6480e6e190e041", "59eed739b5ed5228634659010f5a08da6d907f47eb96b0f8e48c06412b5daaac", "d83d9cd6be2232b120ec5d4ffd370a2a50724552116fe326f56b71286b2f25b1", "172cfa25fa3de782f89bc77e79ed50fef4c18526cb2c02a454db5f7491aa7970", "154d8af4c19b86cf4975d8fdcf8d131e1376b62602af7f663584a03f12b3716a", "913127fb4076378ce8f372101db3c72e51b8477660c14f272251956233b4a089"] }</pre> | |
| <pre>{ "type": "r_commit", "unix_time": 1561350004, "my_address": "merlin@tesis", "my_session_id": 104, "your_address": "arthur@tesis", "your_session_id": 6198, "length": 26, "my_random_commitment_vector": ["828716c8a69789041ef5a07124a7e5e8842afa1fa8bea90b9b452ae51fd7c257", "8a2347fce85e6da1fdb767d0f8f948fc150dff81a0b5a63621e894fcdf2b9db6", "17fedb56286bd7824d825e519d334785b2f5f3167f1b20ef1d7c636fcb585b0d", "78f6989aff20f38a6887654c815c98c3a373346f710a4d9b88dd589a98288d9d", "0e481ef6d53bd3825fa655906be3fe72788136d28390d21ae190c4745d074c3a", "38c09f4c4aedd1a29cfb59f4a296afe91ab57dd8b04b3973ba6480e6e190e041", "59eed739b5ed5228634659010f5a08da6d907f47eb96b0f8e48c06412b5daaac", "d83d9cd6be2232b120ec5d4ffd370a2a50724552116fe326f56b71286b2f25b1", "172cfa25fa3de782f89bc77e79ed50fef4c18526cb2c02a454db5f7491aa7970", "154d8af4c19b86cf4975d8fdcf8d131e1376b62602af7f663584a03f12b3716a", "913127fb4076378ce8f372101db3c72e51b8477660c14f272251956233b4a089"] }</pre> | |

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 4

```
"9e8aba6ab0f5150775a68633cd6514f2716dafb8b76a87ad4bf2866a7ba7b743",
"62dd94b708544e5a0ec038b194c2b90bb36bae26ebbf03c306e628a0e248ef7",
"a747de96d197cf5077f0d8ed9d0620c04926f6e4fbd95c24c442158a8bc54505",
"b600eea392076cfa1ab387f2afc3b9dacb14479dbbc8cc957018ba7a287b0b9e",
"8ee724d497565f18f1f5e526f9634479ba2886cdeb942faf194b1f73686e8dc",
"c1e202ca64266e36581d8b22b668edc969caf5139075acd71f2a51d91c123e67",
"ec4180514cb798c22463050ec5bfe4b56fb4320cd1330653a4a2dea7343a9321",
"099447b900df4019a374886d680f55bd63ac4a4a9b435e4b262df7fb1d9663f8",
"b8a945081a30dc5b6d9e045590ebafa29d1b26ba579693ed7a17afe8e89ba2c2",
"67ba77ce643c1a07638ec241bb5638fd91d9717df715216586217edeb5fa14f8",
"a1604828838c8016daaea5f7b34cec20ac3b5b64599b534f5ce0351626087d7f",
"1cf8769b924390fe502baf425366fe998884057fe33592abe6dbf725563f1e1",
"da0ffa40fc39e1beb3d600a376c262d0d0d0e09581a157a2e2acdf86c425ad72",
"60b32e34561edca8d13809f340eaa9bb58c939d1edb11a3967bfa5bf547067af",
"3c022821aebb123a8b4e5cf6ab3e37e314dae75b74efae914adfb57f2fe12dbd"]}]
{"type": "r_reveal", "unix_time": 1561350005, "my_address": "arthur@tesis", "my_session_id": 6198,
"your_address": "merlin@tesis", "your_session_id": 104, "length": 26, "my_random_vector": [6181, 8329,
713, 5372, 9726, 6953, 8420, 7379, 1582, 5742, 3800, 8604, 4234, 7877, 9669, 6678, 5472, 5006, 4162,
8637, 5207, 4454, 8593, 9528, 5501, 5053], "my_random_nonce_vector": [8834, 4811, 1170, 5518, 474,
1646, 893, 8906, 1939, 3418, 3434, 6882, 9388, 4740, 3542, 4158, 809, 6711, 5571, 57, 8559, 2389,
5186, 1197, 4637, 559]}

{"type": "w_commit", "unix_time": 1561350005, "my_address": "merlin@tesis", "my_session_id": 104,
"your_address": "arthur@tesis", "your_session_id": 6198, "length": 26, "my_word_commitment_vector":
["7f554882eb43b453b57f23b79994387802c63c7df6a2a177a6ae6d310fe009bb",
"1eb00981a9592ba803e2a33886c8f8a5e30ad23815591059f632b75c4b490d25",
"83982556027f1097661569f6589e65e39f3f394b4ccfa418b07689f7b6120098",
"76d6c353527d212ec3799ea375f945e3ae3760257bd2ef5a75441090df0ddfacc",
"10bade5d9c74e548a97b96537db02716e910ab1ecc8ebc072b1e76a509588f44",
"e2d467b272218df47183f651e085223d8bf57f93f23e3472d9983f5e6401a245",
"e27e5eb93a24a2d866e30bf027e4f0c3da9fae8968cf5eb69446e7f668356164",
"05a727a68d6ea1d540d74aed700c9aaadc2db65a4879770c28c69c8b5c7bfbfe",
"fc93073697d3a45583b478932f1a2b7a291cc9e64bcfa57fdcf27107fc107e79",
"cda02199b3471100e47439d60fa60e206e4e2b2de52da5a55ec3e23eeba47afd",
"c0f8897bae266eca9d26a7f9845b45e3d3c702eeaff8eadb728999552caabe2a",
"e7ecebcb590bc88b3761fa6cd03d749f87463dabb67021a5c6768c25ec68b3f2",
"ec7f269feafa5fe49293f6e77ba1c29e9c4ab7969607ba2a435f6df392e7439b",
"0a603db03f8dd2b58aa212091a7dd785f1a23e85bf31839624e15969cc5fc8a7",
"982e59655c52892fb3df0a2d034295d5d5cd775556b3442cedfe688f4b96b04",
"1dc4aab6ff084595d2ddf592aaac8ac004dd1c8236ab29045f684b095f2e0d5",
"458e6be73e8d5e56f2b67c9bb63b32eca40bf9434736d2ac838c1a4651634982",
"eb83e1aef91b5a2ef84074c7d4470d7a7b142df7409896132bcaad3140b1e19c",
"53c26240f787fbc905d0ada0d2876b0fc0f95a4767f641a61abab4f6dfad182b",
"5934faacd5b8f9ea6e80cf27fb248378626a73d9fbfd195b75a2b7e579b0b7e",
"270c39c70041b41e40e65d41e1bc61a2f41fca67b279b62f6e5b30e028896e77",
"6aa580d8f564e8049bcf7d522833272910ca6ad07ac79f87fbc1d0308ecf5780",
"fea1351990c7a63d3275f39ae8ac15e07626b933764a4da95ecd270638f26966",
"e258fc78e23908bdf0123123cb31e7a81008118ab1188ddcb740360727add4f",
"788dd9b2aec2f9bc945ee6adcd48a71e96328bf315242718f92bae3d2a44d5",
"650198ae8e720205763e5e66dfe5205f7ef3256273c1ed801c7e4bd40111d479"}]
{"type": "w_ack", "unix_time": 1561350006, "my_address": "arthur@tesis", "my_session_id": 6198,
"your_address": "merlin@tesis", "your_session_id": 104}

{"type": "make_a_try", "unix_time": 1561350006, "my_address": "merlin@tesis", "my_session_id": 104,
"your_address": "arthur@tesis", "your_session_id": 6198, "round": 1, "status": "_____", "lives_left": 10}
{"type": "new_try", "unix_time": 1561350008, "my_address": "arthur@tesis", "my_session_id": 6198,
"your_address": "merlin@tesis", "your_session_id": 104, "round": 1, "guess": "e"}
{"type": "reply", "unix_time": 1561350008, "my_address": "merlin@tesis", "my_session_id": 104,
"your_address": "arthur@tesis", "your_session_id": 6198, "round": 1, "result": true, "status": "e_e_e_e",
```

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 4

```
"guess": "e", "partial_length": 1, "partial_indices": [4], "partial_random_vector": [1828],
"partial_nonce_vector": [3923]}
{"type": "r_ack", "unix_time": 1561350009, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 1, "validation": true}
{"type": "make_a_try", "unix_time": 1561350010, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 2, "status": "e_e__e", "lives_left": 10}
{"type": "new_try", "unix_time": 1561350010, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 2, "guess": "t"}
{"type": "reply", "unix_time": 1561350011, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 2, "result": true, "status": "e_e__te",
  "guess": "t", "partial_length": 1, "partial_indices": [19], "partial_random_vector": [8104],
  "partial_nonce_vector": [1643]}
{"type": "r_ack", "unix_time": 1561350011, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 2, "validation": true}
{"type": "make_a_try", "unix_time": 1561350012, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 3, "status": "e_e__te", "lives_left": 10}
{"type": "new_try", "unix_time": 1561350012, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 3, "guess": "a"}
{"type": "reply", "unix_time": 1561350013, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 3, "result": false, "status": "e_e__te",
  "guess": "a", "partial_length": 1, "partial_indices": [0], "partial_random_vector": [765],
  "partial_nonce_vector": [6420]}
{"type": "r_ack", "unix_time": 1561350013, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 3, "validation": true}
{"type": "make_a_try", "unix_time": 1561350014, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 4, "status": "e_e__te", "lives_left": 9}
{"type": "new_try", "unix_time": 1561350014, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 4, "guess": "c"}
{"type": "reply", "unix_time": 1561350015, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 4, "result": true, "status": "e_ec_te",
  "guess": "c", "partial_length": 1, "partial_indices": [2], "partial_random_vector": [9222],
  "partial_nonce_vector": [7168]}
{"type": "r_ack", "unix_time": 1561350015, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 4, "validation": true}
{"type": "make_a_try", "unix_time": 1561350016, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 5, "status": "e_ec_te", "lives_left": 9}
{"type": "new_try", "unix_time": 1561350016, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 5, "guess": "u"}
{"type": "reply", "unix_time": 1561350017, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 5, "result": true, "status": "e_e_cute",
  "guess": "u", "partial_length": 1, "partial_indices": [20], "partial_random_vector": [4434],
  "partial_nonce_vector": [2233]}
{"type": "r_ack", "unix_time": 1561350017, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 5, "validation": true}
{"type": "make_a_try", "unix_time": 1561350018, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 6, "status": "e_e_cute", "lives_left": 9}
{"type": "new_try", "unix_time": 1561350018, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 6, "guess": "x"}
{"type": "reply", "unix_time": 1561350019, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "round": 6, "result": true, "status": "execute",
  "guess": "x", "partial_length": 1, "partial_indices": [23], "partial_random_vector": [9699],
  "partial_nonce_vector": [3862]}
{"type": "r_ack", "unix_time": 1561350019, "my_address": "arthur@tesis", "my_session_id": 6198,
  "your_address": "merlin@tesis", "your_session_id": 104, "round": 6, "validation": true}
{"type": "game_ended", "unix_time": 1561350020, "my_address": "merlin@tesis", "my_session_id": 104,
  "your_address": "arthur@tesis", "your_session_id": 6198, "secret": "execute", "result": true, "length": 26,
  "my_random_vector": [765, 5323, 9222, 6269, 1828, 9016, 9307, 6414, 3156, 2386, 185, 8594, 6999],
```

Cuerpo JSON de los mensajes de la partida "execute" (95), corrida 4

```
2708, 9900, 1404, 8123, 4735, 4809, 8104, 4434, 7687, 9807, 9699, 3933, 1099],  
"my_random_nonce_vector": [6420, 2167, 7168, 9486, 3923, 3683, 4251, 6869, 9895, 2836, 7982, 2117,  
6024, 1711, 2169, 76, 3456, 192, 1734, 1643, 2233, 7097, 3848, 3862, 4125, 1480]]  
  {"type": "result_validation", "unix_time": 1561350020, "my_address": "arthur@tesis", "my_session_id":  
6198, "your_address": "merlin@tesis", "your_session_id": 104, "result": true}  
{"type": "result_ack", "unix_time": 1561350021, "my_address": "merlin@tesis", "my_session_id": 104,  
"your_address": "arthur@tesis", "your_session_id": 6198}  
  {"type": "quit", "unix_time": 1561350021, "my_address": "arthur@tesis", "my_session_id": 6198,  
"your_address": "merlin@tesis", "your_session_id": 104}
```

Fuente: Elaboración Propia, 2018.

Anexo C: Listado de palabras de prueba

A continuación se listan las palabras que fueron seleccionadas de forma aleatoria para los experimentos controlados:

- | | | |
|--------------------|--------------------|-------------------|
| 0. flickers | 34. holstered | 68. autopsy |
| 1. liquifies | 35. bizarre | 69. pours |
| 2. indexing | 36. sera | 70. kindness |
| 3. superficially | 37. industrialized | 71. reshuffled |
| 4. gyros | 38. invaliding | 72. logins |
| 5. catching | 39. calves | 73. prosecuting |
| 6. intrude | 40. woodpeckers | 74. roving |
| 7. pelagic | 41. unbelievable | 75. find |
| 8. roadbeds | 42. stroll | 76. wirelasses |
| 9. pregnant | 43. handwritten | 77. concatenation |
| 10. authenticate | 44. commissariats | 78. gossips |
| 11. iamb | 45. naves | 79. demo |
| 12. parries | 46. atomizer | 80. blackballing |
| 13. subprogram | 47. cockpits | 81. circus |
| 14. liters | 48. tomb | 82. aristocracies |
| 15. counterculture | 49. ornately | 83. cavils |
| 16. stationer | 50. masterpieces | 84. pushes |
| 17. appraise | 51. schmooze | 85. sober |
| 18. provisioned | 52. inhalant | 86. spasmodically |
| 19. wronger | 53. barter | 87. rattlesnakes |
| 20. stringent | 54. castoff | 88. inert |
| 21. reupholstering | 55. subtracts | 89. auditory |
| 22. insolently | 56. contrivance | 90. growl |
| 23. lankest | 57. abysmally | 91. instruments |
| 24. forbid | 58. proficiently | 92. betide |
| 25. nonhazardous | 59. modes | 93. variegating |
| 26. charismatic | 60. balk | 94. braves |
| 27. embattled | 61. somnolence | 95. execute |
| 28. preceptors | 62. generative | 96. concurrences |
| 29. benefitted | 63. longingly | 97. leniency |
| 30. prospecting | 64. thyme | 98. biting |
| 31. secondhand | 65. conferring | 99. debasing |
| 32. cretins | 66. considerable | |
| 33. tsarinas | 67. throe | |