Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Analizando la Criptografía en la Práctica: Tuta

Tesis de Licenciatura en Ciencias de la Computación

Tomás Bertoli

Director: Fernando Virdia

Codirector: Diego Garbervetsky

Buenos Aires, 2025

CRYPTOGRAPHY IN THE WILD: TUTA

Increasingly, cloud services are making privacy a priority, with providers such as Signal, MEGA, and Nextcloud being notable examples in the field of messaging and file storage. Security is typically achieved by the use of *end-to-end encryption*. In this context it requires user data to be encrypted in a way that it is accessible only to its rightful owners, where the service provider can learn nothing about what is being stored.

This thesis analyzes the *end-to-end encryption* claims of a private email and calendar provider named Tuta, who reportedly have +10 million-users.[1] In particular, we question whether their user authentication protocol and object encryption scheme hold up to their claims of "zero-knowledge". To this end, we study Tuta's protocol in the malicious server model, where failure to meet expectations means that an misbehaving service provider has the power to compromise integrity or confidentiality of user data.

Having formalized their protocols, we found that Tuta's user authentication protocol is vulnerable to dictionary attacks if a malicious server deviates from the usual login procedure. Additionally, we partially break object integrity in the object encryption scheme, meaning that the server could, for example, hide events from a user's calendar. We then propose two encryption schemes which mitigate the aforementioned issues and provide authenticated encryption for Tuta objects.

Keywords: provable cryptography, end-to-end encryption, symmetric encryption, password hashing, Tuta

[1] According to `https://www.reddit.com/r/tutanota/comments/14at3mh/10_million_people_encrypt_their_emails_calendars/`

ANALIZANDO LA CRIPTOGRAFÍA EN LA PRÁCTICA: TUTA

Cada vez más, los servicios en la nube están priorizando la privacidad. Proveedores como Signal, MEGA y Nextcloud son ejemplos destacados en el ámbito de la mensajería y el almacenamiento de archivos, donde la seguridad suele lograrse mediante el uso de *cifrado de extremo a extremo*. En este contexto, eso significa que los datos del usuario estén cifrados de tal forma que solo los usuarios dueños puedan acceder a ellos, sin que el proveedor del servicio pueda obtener información alguna sobre lo que se almacena. Esta tesis analiza las afirmaciones de *cifrado de extremo a extremo* de un proveedor privado de correo electrónico y calendario llamado Tuta que cuenta con más de 10 millones de usuarios.[2] En particular, nos preguntamos si su protocolo de autenticación de usuarios y su esquema de cifrado de objetos están realmente a la altura de sus afirmaciones de "zero-knowledge encryption". Para ello, estudiamos el protocolo de Tuta en el modelo de servidor malicioso, donde un sistema inseguro en este contexto significa que un proveedor de servicios que se comporta de forma maliciosa tiene la capacidad de comprometer la integridad o la confidencialidad de los datos de usuario.

Al formalizar sus protocolos, encontramos que el protocolo de autenticación de usuarios de Tuta es vulnerable a ataques de diccionario si un servidor malicioso se desvía del procedimiento habitual de inicio de sesión. Además, logramos comprometer parcialmente la integridad de los objetos en el esquema de cifrado, lo que significa que el servidor podría, por ejemplo, ocultar eventos del calendario de un usuario.

Keywords: provable cryptography, end-to-end encryption, symmetric encryption, password hashing, Tuta

[2]Según `https://www.reddit.com/r/tutanota/comments/14at3mh/10_million_people_encrypt_their_emails_calendars/`

Índice

1 Introducción **4**
1.1 Modelos de amenaza 5
1.2 Trabajo relacionado 5
1.3 Contribuciones y organización 6

2 Preliminares **8**
2.1 Notación y Convenciones 8
2.2 Confidencialidad 8
2.3 Integridad 10
2.4 Cifrado Autenticado 11
2.5 Adivinanza de Contraseñas y Funciones Hash Criptográficas 11
2.6 Lemas Útiles 13

3 Protocolo de Objetos Tuta **16**

4 Autenticación de Usuarios **20**
4.1 Seguridad contra un Adversario Snapshot 20
4.2 Seguridad del Esquema de Cifrado en el Modelo Semi-Honesto 21
4.3 Rompiendo el hash con sal en el Modelo de Servidor Malicioso 25

5 Cifrado **30**
5.1 Rompiendo el Esquema de Cifrado de Tuta 32
5.1.1 Impacto Práctico 34
5.2 Una Solución Simple 34
5.3 Manteniendo la Estructura Clave-Valor 36
5.3.1 Consideraciones Prácticas 38
5.4 Usando el Formato JSON 39

6 Conclusión **44**
6.1 Trabajo Futuro 44

A Acotando $\Pr[\mathsf{Guess}_{pw}]$ **48**

B Cota AI-INT sobre el Protocolo de Autenticación de Tuta **49**

C Relacionando $\mathsf{Game}^{\text{S-SUF-CMA}}$ y $\mathsf{Game}^{\text{SUF-CMA}}$ **50**

D Ejemplo de TypeModel **52**

E Código del Ataque de Integridad **54**

1 Introducción

Cada vez más servicios en la nube priorizan la privacidad, con proveedores como Signal, MEGA y Nextcloud como ejemplos destacados en los ámbitos de la mensajería y el almacenamiento de archivos, respectivamente. En estas plataformas la seguridad suele lograrse mediante *cifrado de extremo a extremo* (E2EE): los datos del usuario deben estar cifrados de modo que sólo sus dueños legítimos puedan acceder a ellos, y el proveedor no aprenda nada del contenido almacenado. Esta noción es más fuerte que la seguridad provista por el protocolo de *seguridad en capa de transporte* (TLS) que se utiliza normalmente en la web; éste último sólo protege los datos en *tránsito* y asume que el proveedor de servicios es confiable.

La mayoría de los proveedores de servicios en la nube no dan garantías sobre la privacidad de datos de usuario, en muchos casos utilizándolos con fines comerciales, para *machine-learning*, u otros propósitos, sin permiso explícito del usuario (e.g. habilitando esta funcionalidad por defecto, escondido dentro de los términos de servicio). Otra preocupación es la participación de organizaciones estatales, como servicios de inteligencia, que pueden obligar a proveedores a entregar datos de usuario. El cifrado de extremo a extremo mitiga estos riesgos directamente al evitar el acceso a los datos de usuario por parte del proveedor.

Algunos de los sistemas más prominentes de almacenamiento cifrado han resultado vulnerables (MEGA [Alb+23; BHP23] y Nextcloud [Alb+24]); otros, como el protocolo de mensajería Signal, han sido formalmente demostrados seguros [Coh+17]. En general, si un protocolo no ha sido objeto de un análisis formal, su seguridad efectiva es desconocida y no debería asumirse sólo en base al marketing.

El espacio de calendarios privados está poco estudiado: la mayoría de los trabajos se centra en superponer cifrado sobre plataformas existentes (e.g. Google Calendar) [Dia+12; VK18; VHK20; San+12]. Dado que los calendarios son una ventana a la vida personal y profesional de los usuarios, sostenemos que la privacidad debe estar contemplada desde el diseño inicial, por eso analizamos un sistema que ya promete privacidad desde el principio. En esta tesis analizamos formalmente un calendario con cifrado de extremo a extremo desplegado en producción: Tuta [Gmba]. Este fue elegido entre alternativas como Proton [AG], OurCal [Our] o la ya discontinuada Skiff [Ski] por lo llamativo de sus afirmaciones de marketing: “cifrado de extremo a extremo” y una “arquitectura de conocimiento cero” [Gmbb].

Evaluaremos sus afirmaciones en el protocolo de autenticación de usuarios y en su esquema de cifrado de objetos, y concluiremos que su diseño actual no es totalmente seguro bajo el modelo de *servidor malicioso*. Nuestros hallazgos principales son:

1. El protocolo de autenticación de usuarios es vulnerable a ataques de recuperación de contraseñas con preprocesamiento si un servidor comprometido entrega una sal maliciosa.

2. El esquema de cifrado permite ataques que rompen parcialmente la integridad de los ciphertexts, posibilitando ocultar selectivamente eventos de un calendario.

Si bien no encontramos una solución clara para el primer punto, proponemos una solución que mitiga el segundo, proporcionando un esquema de cifrado adecuado.

Calendarios como almacenamiento de archivos. Aunque nos centramos en el servicio de calendario, observamos que el diseño de Tuta sigue un paradigma en el cual todos los datos cifrados del usuario equivalen a “archivos” y Tuta actúa como servicio de almacenamiento. Se puede pensar que un evento se guarda como un “archivo” cifrado en un “directorio” que representa el calendario. Esto nos permite acotar el análisis a dos aspectos esenciales: (i) autenticación y gestión de claves y (ii) cifrado de objetos.

1.1 Modelos de amenaza

Tuta se presenta como un servicio "Secure & Private" y "Zero-knowledge" [Gmbb]. Logran esto gestionando con el mencionado cifrado de extremos a extremo, cifrando y descifrando los datos sensibles íntegramente del lado cliente (el servidor no conoce claves de descifrado) y solo almacenando datos *cifrados* en el servidor.

Mientras que las propuestas de Tuta son fuertes, analizaremos sus protocolos bajo tres modelos de amenaza con distintas capacidades para entender mejor sus garantías de seguridad. Los tres modelos de amenaza con distintas capacidades son los siguientes:

Adversario snapshot. Se puede considerar como un tercero (i.e. no el proveedor de servicios) que roba una copia de la base de datos en un momento dado; no puede interactuar ni espiar conexiones. Buscamos que los protocolos provean confidencialidad de datos sensibles incluso ante esta intrusión. No consideramos integridad porque el acceso no es persistente.

Servidor semi-honesto. Este adversario puede ser el proveedor de servicios, un adversario que puede observar todo pero debe seguir el protocolo preestablecido. Evaluamos qué información sensible puede inferir sin modificar los protocolos. Este tipo de adversario tiene acceso a las comunicaciones con clientes en tiempo real, y tiene acceso a todo el estado del servidor, pero no puede alterar el protocolo cuando interactúa con usuarios.

Servidor malicioso. Extendiendo las habilidades del servidor semi-honesto, el servidor malicioso puede alterar arbitrariamente su rol y comunicaciones: mezclar, omitir o falsificar respuestas. Un servicio seguro en este modelo requiere mínima confianza y resiste manipulación activa, necesitando de mínima confianza de los usuarios.

1.2 Trabajo relacionado

La mayoría de trabajos sobre calendarios seguros añade capas de cifrado a servicios existentes mediante proxies o middleware [Dia+12; San+12; VK18; VHK20]. Aunque viables, su adopción masiva es poco realista por barreras operativas. Además suelen asumir un servidor honesto-pero-curioso (o semi-honesto). A diferencia de estos trabajos, y dadas las afirmaciones fuertes de Tuta, analizaremos sus protocolos bajo el modelo del servidor malicioso.

Calendarios seguros. Vörös y Kiss [VK18] proponen dos opciones de proxy de cifrado en el lado cliente: (i) extensiones de navegador (más fricción, control granular) y (ii) un proxy confiable centralizado (más simple para organizaciones, mayor necesidad de confianza).

Para el primer enfoque, todo cliente que busca cifrar su calendario debe instalar la extensión y configurar su clave de cifrado, lo que puede ser una barrera para usuarios menos técnicos. El beneficio es la modularidad: el cliente decide que sitios web son cifrados, deshabilitando la extensión cuando no se requiere.

El segundo enfoque requiere que los usuarios confíen en un proxy central, donde cada usuario conectado a una red confiada (e.g. una red empresarial) es protegido por un único proxy que se encarga del cifrado. Esto es más simple de implementar y administrar, y está más en línea con expectativas corporativas, pero requiere de confianza en el proxy de parte de los usuarios: un administrador de red podría potencialmente acceder a datos sensibles.

Su trabajo posterior con Hudoba [VHK20] introduce esteganografía para evitar detección como usuario de cifrado. En este trabajo, utilizan una codificación para formar oraciones en base a los datos cifrados.

Diallo et al. [Dia+12] diseñan un *middleware* extensible que actúa entre usuario y proveedor, de manera genérica para todas las plataformas. Lamentablemente, su sistema puede, bajo casos específicos, proveer los datos descifrados al servidor, algo que es inaceptable en nuestro modelo de amenaza.

Por último, Sanamrad et al. [San+12] profundizan el análisis criptográfico formal creando un proxy de cifrado para Google Calendar. Ellos se enfocan en aspectos que diferencian a un calendario de un almacenamiento de archivos común y corriente; en particular búsqueda de eventos, e invitación a eventos. Para la búsqueda, utilizan listas de palabras clave por evento. Respecto al manejo de invitaciones ellos anonimizan las identidades de los invitados, utilizando un servidor de email intermediario: cada invitado recibe un identificador de email único en el intermediario y el proxy de cifrado asigna este nuevo email en la invitación de google calendar. Al comunicarse, el intermediario reenvia los emails entre google calendar y el invitado.

Almacenamiento en la nube E2EE. Mientras que los trabajos mencionados se enfocan en los calendarios, muchos no demuestran la seguridad de sus protocolos. Existe una conexión natural entre calendarios y almacenamiento de archivos (e.g. eventos como archivos `.ics`). Por este fenómeno, existen protocolos de red que proveen servicios de calendario en la nube *sin garantías de privacidad*, e.g. CalDAV [DDD07]. Por esto, trabajos que tratan formalmente el almacenamiento de archivos cifrado de extremo a extremo como [Alb+23; Alb+24; BHP23; Bac+24] son relevantes al análisis de Tuta, y a los calendarios cifrados en general.

Cifrado basado en contraseñas. Hoy en día, la mayoría de los servicios en la nube utilizan sistemas de autenticación de usuarios basados en contraseñas con *salted hashing*, la práctica de agregar un valor aleatorio (sal) a la contraseña antes del proceso de *hashing* con el objetivo de mitigar ataques de precómputo. Tuta no solo sigue esta práctica, almacenando un hash con sal de la contraseña del usuario para autenticarlo, si no que también se deriva la clave maestra de cifrado a partir de la contraseña.

Farshim y Tessaro [FT21] analizan la seguridad de los esquemas de recuperación de contraseñas en escenarios con preprocesamiento, considerando tres casos: (i) hashing sin sal, (ii) sal unívoca por usuario y (iii) sales uniformemente aleatorias por usuario. Ellos demuestran que la ventaja de un adversario disminuye con un esquema de sal apropiado (con las sales aleatorias dando el mejor resultado), pero depende de la entropía efectiva de la distribución de contraseñas.

1.3 Contribuciones y organización

Primero describimos el protocolo de objetos Tuta, detallando como los datos son cifrados, almacenados y manejados en el servidor.

Luego, analizamos en profundidad el protocolo de autenticación de usuarios, enfocándonos en su seguridad bajo los tres distintos modelos de adversario. Mostramos el razonamiento detrás de ciertas decisiones de diseño y también de la seguridad del esquema de cifrado con hash de contraseña. Mientras que esos análisis son positivos, nuestra aplicación de resultados de Backendal et al. [Bac+24] demostró una vulnerabilidad que podría habilitar ataques de recuperación de contraseñas en el modelo del servidor malicioso.

Después analizamos el esquema de cifrado de objetos de Tuta, demostrando un ataque parcial de integridad. Proponemos dos esquemas alternativos para resolver las vulnerabilidades encontradas: un esquema simple basado en serialización, y uno más complejo que mantiene la estructura clave-valor de los objetos. Finalmente, proponemos los inicios de un esquema

de cifrado basado en JSON que podría ser utilizado en el futuro, buscando solucionar el problema de las claves de objeto duplicadas.

La Sección 2 contiene las preliminares utilizadas en este trabajo. La Sección 3 describe cómo está estructurado el protocolo de objetos de Tuta, haciendo foco en la gestión de claves, el compartido y cifrado de objetos, y la autenticación de usuarios.

Luego, en la Sección 4 analizamos el protocolo de autenticación bajo los tres modelos de amenaza. Seguido de la Sección 5 donde, tras una breve introducción al esquema de cifrado, la Subsección 5.1 explica la vulnerabilidad encontrada, y la Subsección 5.2 y la Subsección 5.3 muestran posibles soluciones. La Subsección 5.4 provee un punto de partida para un esquema MAC compatible con JSON.

Finalmente, en la Sección 6, concluimos el trabajo con un breve resumen y un punto de partida para futuros trabajos.

2 Preliminares

Este trabajo utiliza el marco de pruebas de seguridad basadas en juegos propuesto por [BR04], en el que definimos un *juego* donde un retador (*challenger*) interactúa con un adversario siguiendo un procedimiento establecido. De esta manera, nuestro objetivo es acotar la ventaja del adversario en *ganar el juego de seguridad*, donde la condición de "ganar" la establece el retador. Aunque intentamos explicar los conceptos básicos de las pruebas basadas en juegos, se puede consultar [Ros; BS23; KL14] para una introducción a este tema.

2.1 Notación y Convenciones

En el contexto de los algoritmos, escribimos $x \leftarrow\!\!\$\ \mathcal{R}$ para decir "la variable x toma un valor aleatorio de la distribución $\mathcal{R}$". Abusando de la notación, podemos escribir $x \leftarrow\!\!\$\ \mathcal{S}$ para un conjunto $\mathcal{S}$ para indicar que $x \leftarrow\!\!\$\ U(\mathcal{S})$, o "la variable x toma un valor aleatorio uniforme del conjunto $\mathcal{S}$". También usamos la notación $[\![\phi]\!]$, que se define como $[\![\phi]\!] = 1$ si y solo si la expresión booleana ϕ es verdadera, y $[\![\phi]\!] = 0$ en caso contrario. Para arreglos escribiremos $\vec{v}$ y accederemos a su i-ésimo elemento con $\vec{v}[i]$.

La mayoría de las veces, las variables de un juego son *globales* en el sentido de que todos los procedimientos llamados desde el procedimiento principal del juego pueden acceder directamente a ellas. Esta propiedad *no* se extiende al adversario: si el adversario $\mathcal{A}$ es instanciado dentro de un juego, no obtiene acceso directo a esas variables.

2.2 Confidencialidad

Vamos a trabajar con esquemas de cifrado de clave secreta (SKES, por sus siglas en inglés), que se pueden definir como un par de algoritmos que cumplen con la siguiente definición.

Definición 1 (Esquema de cifrado de clave secreta con datos asociados [BS23])**.** *Un esquema de cifrado de clave secreta con datos asociados $\mathcal{E} = (E, D)$, con un espacio de claves $\mathcal{K}$, un espacio de mensajes $\mathcal{M}$, un espacio de datos asociados $\mathcal{A}$ y un espacio de textos cifrados $\mathcal{C}$, consiste en:*

- *Un algoritmo de cifrado $E : \mathcal{K} \times \mathcal{M} \times \mathcal{A} \to \mathcal{C}$*
- *Un algoritmo de descifrado $D : \mathcal{K} \times \mathcal{C} \times \mathcal{A} \to \mathcal{M} \cup \{\perp\}$*

tal que para toda clave $k \in \mathcal{K}$, mensaje $m \in \mathcal{M}$ y dato asociado $a \in \mathcal{A}$,

$$\Pr[D(k, E(k, m, a), a) = m] = 1$$

Aunque definimos el concepto de *datos asociados*, la mayor parte del tiempo lo ignoraremos y trabajaremos con el caso especial sin datos asociados. En tal caso, el algoritmo de cifrado se reemplaza por $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ y el de descifrado por $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$. Podemos verlo como definir $\mathcal{A} = \{\varepsilon\}$ y que E ignora ese parámetro. El rol de los *datos asociados* es: datos a los que no se les garantiza confidencialidad, sino únicamente *integridad*. Que los datos tengan integridad significa que puede verificarse si fueron manipulados. Formalizaremos esto más adelante.

Para formalizar la noción de *confidencialidad*, definimos el juego de indistinguibilidad bajo ataques de texto plano elegido (de ahora en adelante, IND-CPA) $\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}$. Este juego captura la noción de *seguridad semántica*, es decir, la capacidad de un esquema de cifrado para mantener secreto el significado de un mensaje (en lugar de su sintaxis o longitud).

Definición 2 (Seguridad IND-CPA)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta. Definimos el juego IND-CPA de la siguiente manera:*

$\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}(\mathcal{A}, b)$	$\mathcal{O}(m_0, m_1)$
$k \leftarrow\!\!\$\ \mathcal{K}$	**assert** $\lvert m_0\rvert = \lvert m_1\rvert$
$\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}}()$	$c \leftarrow E(k, m_b)$
return $\hat{b}$	**return** c

Decimos que $\mathcal{E}$ es (ε, t, q)-IND-CPA seguro si cualquier adversario $\mathcal{A}$ que corre en un tiempo $\leq t$ y hace $\leq q$ consultas al oráculo tiene una ventaja:

$$\mathsf{Adv}(\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}, \mathcal{A}) := \left|\Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}(\mathcal{A}, 0) \Rightarrow 1\right] - \Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}(\mathcal{A}, 1) \Rightarrow 1\right]\right| \leq \varepsilon$$

Comentario. *Como este es el primer juego que presentamos, vamos a explicar cómo funciona el sistema de pruebas. Acá, el procedimiento llamado $\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}$ es el* retador*, que instancia a un* adversario *$\mathcal{A}$. El otro parámetro de entrada, b, es un bit que se usa para elegir entre el mensaje de la izquierda y el de la derecha en el oráculo de cifrado $\mathcal{O}$.*

Cuando decimos que un esquema de cifrado $\mathcal{E}$ es IND-CPA *seguro, en realidad nos referimos a algo más profundo. En particular, omitimos (y seguiremos omitiendo) la noción de* parámetro de seguridad*. Los esquemas de cifrado se parametrizan con un parámetro λ (longitud de clave, etc.). Un esquema es seguro si para todo λ y todo adversario probabilístico de tiempo polinomial $\mathcal{A}$, la ventaja de $\mathcal{A}$ está acotada por una función inversa polinómica en λ (función* despreciable*). Esta definición rigurosa nos permite razonar sobre el esquema ante incrementos de poder computacional. Para más detalles ver [BS23, Sección 2.3] y [KL14, Sección 3.1]. Una definición equivalente usa un oráculo que, dado m, según b devuelve un valor aleatorio uniforme o un cifrado real de m [Bel+97].*

Comentario. *El concepto detrás de esta definición es la* seguridad semántica*: un atacante no puede aprender información sobre el mensaje (por ejemplo bits individuales, paridad, etc.). Se exige $|m_0| = |m_1|$ para evitar la fuga trivial de b por longitud. La longitud es una fuga habitual; ocultarla requiere mecanismos adicionales (padding, etc.). Para más ejemplos ver [BS23, Sección 2.2].*

Comentario. *También es posible encontrar sistemas que afirman tener λ* bits de seguridad *para cierta noción. Esto significa que deberían proveer (ε, t)-seguridad para todo $t \geq 2^{\lambda} \cdot \varepsilon$.*

Y para el caso de datos asociados, definimos el juego de indistinguibilidad bajo ataques de texto plano elegido con datos asociados (IND-CPA-AD) de la siguiente manera.

Definición 3 (Seguridad IND-CPA con datos asociados [BS23])**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta con datos asociados. Definimos el juego IND-CPA con datos asociados así:*

$\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA-AD}}(\mathcal{A}, b)$	$\mathcal{O}(m_0, m_1, a)$
$k \leftarrow\!\!\$\ \mathcal{K}$	**assert** $\lvert m_0\rvert = \lvert m_1\rvert$
$\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}}()$	$c \leftarrow E(k, m_b, a)$
return $\hat{b}$	**return** c

Decimos que $\mathcal{E}$ es (ε, t, q)-IND-CPA seguro con datos asociados si cualquier adversario $\mathcal{A}$ que corre en tiempo $\leq t$ y hace $\leq q$ consultas al oráculo tiene una ventaja:

$$\mathsf{Adv}(\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA-AD}}, \mathcal{A}) := \begin{array}{l}\Big|\Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA-AD}}(\mathcal{A}, 0) \Rightarrow 1\right] \\ \quad - \Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA-AD}}(\mathcal{A}, 1) \Rightarrow 1\right]\Big|\end{array} \leq \varepsilon$$

2.3 Integridad

Para poder confiar en que un mensaje, cifrado o no, fue enviado sin modificaciones, nos interesa garantizar su autenticidad. La primitiva usual es el código de autenticación de mensajes (MAC): permite etiquetar un mensaje con una clave secreta y verificar luego la etiqueta. Formalmente, un MAC es un par (T, V) con $T : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ y $V : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{0,1\}$ —siendo $\mathcal{T}$ el espacio de etiquetas— tal que para toda clave k y mensaje m, $\Pr[V(k, m, T(k, m)) = 1] = 1$ y satisface infalsificabilidad fuerte (SUF-CMA).

Definición 4 (SUF-CMA)**.** *Sea $\mathcal{M} = (T, V)$ un MAC. Definimos el juego SUF-CMA de la siguiente manera:*

$\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}}(\mathcal{A})$	$\mathcal{O}(m)$
$Q \leftarrow \{\}$	$\tau \leftarrow T(k, m)$
$k \leftarrow\!\!\$\ \mathcal{K}$	$Q \leftarrow Q \cup \{(m, \tau)\}$
$(m^*, \tau^*) \leftarrow \mathcal{A}^{\mathcal{O}}()$	**return** τ
return $[\![V(k, m^*, \tau^*) = 1 \wedge (m^*, \tau^*) \notin Q]\!]$	

Decimos que $\mathcal{M}$ proporciona (ε, t, q)-SUF-CMA si cualquier adversario $\mathcal{A}$ que corre en tiempo $\leq t$ y hace $\leq q$ consultas al oráculo tiene una ventaja:

$$\mathsf{Adv}(\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}}, \mathcal{A}) := \Pr\left[\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}}(\mathcal{A}) \Rightarrow 1\right] \leq \varepsilon$$

Adaptando la definición anterior a los esquemas de cifrado, obtenemos la integridad del texto cifrado, que busca garantizar la resistencia de los esquemas a atacantes que alteran el texto cifrado.

Definición 5 (Integridad del texto cifrado)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta donde $D(k, c)$ devuelve $\perp$ si se detecta que c fue manipulado. Definimos el juego de integridad de texto cifrado $\mathsf{Game}^{\text{INT}}_{\mathcal{E}}$ así:*

$\mathsf{Game}^{\text{INT}}_{\mathcal{E}}(\mathcal{A})$	$\mathcal{O}(m)$
$Q \leftarrow \{\}$	$c \leftarrow E(k, m)$
$k \leftarrow\!\!\$\ \mathcal{K}$	$Q \leftarrow Q \cup c$
$c^* \leftarrow \mathcal{A}^{\mathcal{O}}()$	**return** c
$m^* \leftarrow D(k, c^*)$	
return $[\![m^* \neq \perp \wedge c^* \notin Q]\!]$	

Decimos que $\mathcal{E}$ proporciona (ε, t, q)-integridad de texto cifrado si cualquier adversario que corre en tiempo $\leq t$ y hace $\leq q$ consultas al oráculo tiene una ventaja:

$$\mathsf{Adv}(\mathsf{Game}^{\text{INT}}_{\mathcal{E}}, \mathcal{A}) := \Pr\left[\mathsf{Game}^{\text{INT}}_{\mathcal{E}}(\mathcal{A}) \Rightarrow 1\right] \leq \varepsilon$$

Una noción similar adaptada al caso de datos asociados se puede definir de la siguiente manera.

Definición 6 (Integridad del texto cifrado con datos asociados [BS23])**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta con datos asociados donde $D(k, c, a)$ devuelve $\perp$ si se detecta que c fue manipulado. Definimos el juego de integridad de texto cifrado con datos asociados $\mathsf{Game}^{\text{INT-AD}}_{\mathcal{E}}$ así:*

$\mathsf{Game}_{\mathcal{E}}^{\text{INT-AD}}(\mathcal{A})$	$\mathcal{O}(m, a)$
$Q \leftarrow \{\}$	$c \leftarrow E(k, m, a)$
$k \leftarrow\!\!\$\ \mathcal{K}$	$Q \leftarrow Q \cup \{(c, a)\}$
$(c^*, a^*) \leftarrow \mathcal{A}^{\mathcal{O}}()$	**return** c
$m^* \leftarrow D(k, c^*, a^*)$	
return $[\![m^* \neq \perp \wedge (c^*, a^*) \notin Q]\!]$	

Decimos que $\mathcal{E}$ proporciona (ε, t, q)-integridad de texto cifrado con datos asociados si cualquier adversario que corre en tiempo $\leq t$ y hace $\leq q$ consultas al oráculo tiene una ventaja:

$$\mathsf{Adv}(\mathsf{Game}_{\mathcal{E}}^{\text{INT-AD}}, \mathcal{A}) := \Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{INT-AD}}(\mathcal{A}) \Rightarrow 1\right] \leq \varepsilon$$

2.4 Cifrado Autenticado

Un esquema de cifrado de clave secreta que combina la integridad del texto cifrado con la indistinguibilidad de texto plano elegido se conoce como un *esquema de cifrado autenticado.*

Definición 7 (Cifrado autenticado [KL14])**.** *Let $\mathcal{E}$ be a secret key encryption scheme. Si $\mathcal{E}$ proporciona seguridad $(\varepsilon_{CPA}, t, q)$*-IND-CPA *e integridad de texto cifrado $(\varepsilon_{INT}, t, q)$, entonces decimos que $\mathcal{E}$ proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t, q)$-cifrado autenticado.*

Si el esquema de cifrado soporta datos asociados, podemos definir el cifrado autenticado con datos asociados (AEAD) de la siguiente manera.

Definición 8 (Cifrado autenticado con datos asociados (AEAD) [Rog02])**.** *Sea $\mathcal{E}$ un esquema de cifrado de clave secreta con datos asociados. Si $\mathcal{E}$ proporciona seguridad $(\varepsilon_{CPA}, t, q)$-IND-CPA-AD e integridad de texto cifrado con datos asociados $(\varepsilon_{INT}, t, q)$, entonces decimos que $\mathcal{E}$ proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t, q)$-cifrado autenticado con datos asociados.*

2.5 Adivinanza de Contraseñas y Funciones Hash Criptográficas

Ahora pasamos al problema de las contraseñas. Es bien sabido que las contraseñas suelen ser fáciles de adivinar: deben memorizarse, son cortas y se basan en palabras o frases comunes. Un paso para mitigar esto es almacenarlas en forma "enmascarada" usando funciones hash: funciones deterministas que toman una string de longitud arbitraria y producen una salida de longitud fija de la que es computacionalmente difícil invertir la operación (hallar un preimagen). Otra propiedad es que incluso pequeños cambios en la entrada producen hashes completamente diferentes, dificultando la tarea de inversión.

Se mejora la seguridad añadiendo una sal aleatoria por usuario antes de hashear y guardando esta sal junto al hash. Así, usuarios con contraseñas iguales obtienen hashes distintos. Esto dificulta ataques de diccionario basados en tablas precomputadas, donde un adversario podría usar una tabla de evaluaciones a H en valores comunes, pudiendo recuperar fácilmente una preimagen. El *salt* obliga al atacante a recomputar dicha tabla por cada sal.

El diseño interno de una función hash es complejo; para nuestro análisis la modelaremos como una función aleatoria ideal: un *oráculo aleatorio.* Usaremos el Modelo del Oráculo Aleatorio (ROM) [BR93], que reemplaza $H : \mathcal{X} \to \mathcal{Y}$ por un oráculo RO uniformemente aleatorio, muestreado de $\mathsf{Funcs}[\mathcal{X}, \mathcal{Y}]$.

Formalmente, un oráculo aleatorio se define por un procedimiento que muestrea un valor del codominio sólo cuando se consulta una nueva entrada por primera vez. Esta construcción ("gnomo fiel" [BS23]) se da así:

$RO(x)$

if $\exists y((x,y) \in L)$
  return y
$y \leftarrow\!\!\$\ \mathcal{Y}$
$L \leftarrow L \cup \{(x,y)\}$
return y

Las salidas de RO son uniformes, pero el registro L lo hace determinista: para un x fijo, $RO(x)$ siempre es el mismo y. Esto permite modelar funciones hash ideales incluso con dominios infinitos. También es importante aclarar que este procedimiento nos da una implementación concreta del oráculo aleatorio que corre en tiempo polinómico.

Usaremos las definiciones de [FT21] para distribuciones de contraseñas y juegos de adivinanza. Comenzamos con los *muestreadores de contraseñas* (también conocidos como m-samplers), un algoritmo aleatorizado $\mathcal{PW}$ que devuelve un vector de m contraseñas y una fuga z.

Definición 9 (Adivinanza de contraseñas [FT21])**.** *Dado un m-muestreador $\mathcal{PW}$, definimos el juego de adivinanza de contraseñas de la siguiente manera:*

$\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}(\mathcal{A})$

$\overrightarrow{win} \leftarrow \vec{0}$ // $win \in \{0,1\}^m$
$(\overrightarrow{pw}, z) \leftarrow \mathcal{PW}$
$y \leftarrow \mathcal{A}^{\mathsf{Test},\mathsf{Cor}}(z)$
return $\bigwedge_{i=1}^{m} [\![\overrightarrow{win}[i]]\!]$

$\mathsf{Test}(i, pw)$

$\overrightarrow{win}[i] \leftarrow (pw = \overrightarrow{pw}[i])$
return $\overrightarrow{win}[i]$

$\mathsf{Cor}(i)$

assert $i \in [1, m]$
return $\overrightarrow{pw}[i]$

Decimos que $\mathcal{PW}$ es (ε, t, q, c)-resistente a ataques de adivinanza de contraseñas si cualquier adversario $\mathcal{A}$ que corre en tiempo $\leq t$, haciendo $\leq q$ consultas al oráculo Test y $\leq c$ consultas de corrupción, tiene una ventaja:

$$\mathsf{Adv}(\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}, \mathcal{A}) := \Pr\left[\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}(\mathcal{A}) \Rightarrow 1\right] \leq \varepsilon$$

Lema 1 ([FT21])**.** *Sea $\mathcal{A}$ un atacante a $\mathcal{PW}$, un m-muestreador $(\varepsilon, t, q, 0)$-resistente a ataques de adivinanza de contraseñas. Entonces, podemos acotar la capacidad de no ser adivinado de $\mathcal{PW}$ por la min-entropía de caso promedio:*

$$\mathsf{Adv}(\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}, \mathcal{A}) \leq 2^{-\tilde{\mathbf{H}}_\infty(\mathcal{PW}|\mathcal{Z})}$$

donde $\tilde{\mathbf{H}}_\infty(\mathcal{PW}|\mathcal{Z}) := -\log \mathbb{E}_{\mathcal{Z}}[2^{-\mathbf{H}_\infty(\mathcal{PW}|\mathcal{Z}=z)}]$

La noción de que una contraseña sea fácilmente adivinable se captura en el siguiente lema, que acota la ventaja total por la min-entropía promedio de la distribución (dado el leakage z). Si la distribución dista de uniforme, el atacante gana ventaja. Además, Farshim et al. [FT21] proporcionan una definición para atacantes que pueden preprocesar información sobre un oráculo aleatorio.

Definición 10. *Sea $\mathcal{PW}$ un m-muestreador, ℓ la cantidad de sales (*salts*) por cada una de las m contraseñas,* Gen *una función de generación de sales y KD^H una función de derivación de clave basada en el oráculo aleatorio H. Definimos el juego de* recuperación de contraseñas con entrada auxiliar *(ai-rec) de la siguiente manera:*

$\mathsf{Game}^{\text{ai-rec}}_{\mathcal{P},\ell,\mathsf{Gen},\mathsf{KD}}(\mathcal{A})$	$\mathsf{Cor}(i)$
$H \leftarrow\!\!\$\ \mathsf{Funcs}[\mathcal{PW},\mathcal{H}]$	**return** $\overrightarrow{pw}[i]$
$\sigma \leftarrow \mathcal{A}_0(H)$	
$(\overrightarrow{pw}, z) \leftarrow\!\!\$\ \mathcal{PW}$	
for $(i,j) \in [m] \times [\ell]$	
$\quad \overrightarrow{sa}[i,j] \leftarrow \mathsf{Gen}(i,j)$	
$\quad \vec{k}[i,j] \leftarrow \mathsf{KD}^H(\overrightarrow{pw}[i], \overrightarrow{sa}[i,j])$	
$\overrightarrow{pw}' \leftarrow \mathcal{A}_1^{\mathsf{Cor},H}(\overrightarrow{sa}, \vec{k}, \sigma, z)$	
return $[\![\overrightarrow{pw}' = \overrightarrow{pw}]\!]$	

Decimos que $\mathcal{PW}$ es (ε, S, T, c)-resistente a ataques de recuperación de contraseñas si cualquier adversario $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ tiene $|\sigma|_{bits} \leq S$, $\mathcal{A}_1$ corre en tiempo $\leq T$, y hace $\leq c$ consultas al oráculo de corrupción, con una ventaja:

$$\mathsf{Adv}(\mathsf{Game}^{\text{ai-rec}}_{\mathcal{P},\ell,\mathsf{Gen},\mathsf{KD}}, \mathcal{A}) := \Pr\left[\mathsf{Game}^{\text{ai-rec}}_{\mathcal{P},\ell,\mathsf{Gen},\mathsf{KD}}(\mathcal{A}) \Rightarrow 1\right] \leq \varepsilon$$

Este juego define una función de derivación de clave KD^H que se encarga de tomar una contraseña y una sal como entrada, y devolver una clave. En nuestro análisis, KD^H será instanciada como $\mathsf{KD}^H(r,s) = H(r\|s)$, donde $\|$ denota concatenación. Otros esquemas pueden usar funciones de derivación de clave más complejas, como múltiples rondas de hashing, un escenario explorado por Farshim y Tessaro [FT21].

2.6 Lemas Útiles

Para cada juego $\mathsf{Game}(\mathcal{A}, b)$ que toma un bit b, podemos definir la variante de *adivinanza de bit* de dicho juego de la siguiente manera.

Definición 11 (Juegos de adivinanza de bit)**.** *Sea* Game *un juego que toma como entrada un adversario $\mathcal{A}$ y un bit b, donde la ventaja se puede calcular como:*

$$\mathsf{Adv}(\mathsf{Game}, \mathcal{A}) = |\Pr[\mathsf{Game}(\mathcal{A}, 0) \Rightarrow 1] - \Pr[\mathsf{Game}(\mathcal{A}, 1) \Rightarrow 1]|$$

Entonces, definimos la variante de adivinanza de bit de Game*, denotada como* $\overline{\mathsf{Game}}$*, de la siguiente forma:*

$\overline{\mathsf{Game}}(\mathcal{A})$
$b \leftarrow\!\!\$\ \{0,1\}$
$\hat{b} \leftarrow \mathsf{Game}(\mathcal{A}, b)$
return $[\![b = \hat{b}]\!]$

Donde la ventaja en $\overline{\mathsf{Game}}$ *está dada por:*

$$\mathsf{Adv}(\overline{\mathsf{Game}}, \mathcal{A}) = \left|\Pr\left[\overline{\mathsf{Game}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2}\right|$$

Además, podemos relacionar la ventaja de un juego con la de su variante de adivinanza de bit mediante el siguiente lema.

Lema 2 (Ventaja de adivinanza de bit)**.** *Sea* Game *un juego que toma un bit b como entrada. Entonces, para un adversario $\mathcal{A}$ dado, la ventaja de* $\overline{\mathsf{Game}}$ *se relaciona con la de la versión que no es de adivinanza de bit mediante:*

$$\mathsf{Adv}(\mathsf{Game}, \mathcal{A}) = 2 \cdot \mathsf{Adv}(\overline{\mathsf{Game}}, \mathcal{A})$$

Demostración. Primero, descomponemos la ventaja de Game en los dos valores posibles de b.

$$\begin{aligned}
\Pr\left[\overline{\mathsf{Game}}(\mathcal{A}) \Rightarrow 1\right] &= \Pr\left[b = \hat{b}\right] \\
&= \Pr\left[b = \hat{b} \mid b = 0\right] \Pr[b = 0] \\
&\quad + \Pr\left[b = \hat{b} \mid b = 1\right] \Pr[b = 1] && \text{(ley de probabilidad total)} \\
&= \frac{1}{2} \Pr\left[b = \hat{b} \mid b = 0\right] + \frac{1}{2} \Pr\left[b = \hat{b} \mid b = 1\right] && (b \text{ se elige uniformemente})
\end{aligned}$$

Luego, podemos ver que $\Pr\left[b = \hat{b} \mid b = 1\right]$ corresponde a $\Pr[\mathsf{Game}(\mathcal{A}, 1) \Rightarrow 1]$, es decir, $\mathcal{A}$ adivinando 1 cuando el bit b tenía valor 1. De manera similar, $\Pr\left[b = \hat{b} \mid b = 0\right]$ corresponde a $\Pr[\mathsf{Game}(\mathcal{A}, 0) \Rightarrow 0]$ o, lo que es igual, $1 - \Pr[\mathsf{Game}(\mathcal{A}, 0) \Rightarrow 1]$. Por lo tanto:

$$\begin{aligned}
\mathsf{Adv}(\overline{\mathsf{Game}}, \mathcal{A}) &= \left|\Pr\left[\overline{\mathsf{Game}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2}\right| \\
&= \left|\frac{1}{2}\left(1 - \Pr[\mathsf{Game}(\mathcal{A}, 0) \Rightarrow 1]\right) + \frac{1}{2}\Pr[\mathsf{Game}(\mathcal{A}, 1) \Rightarrow 1] - \frac{1}{2}\right| \\
&= \frac{1}{2}\left|\Pr[\mathsf{Game}(\mathcal{A}, 0) \Rightarrow 1] - \Pr[\mathsf{Game}(\mathcal{A}, 1) \Rightarrow 1]\right| \\
&= \frac{1}{2}\mathsf{Adv}(\mathsf{Game}, \mathcal{A})
\end{aligned}$$

□

Adaptado del lema fundamental del *game playing* [BR06], el siguiente lema nos permite acotar la ventaja de un adversario en dos juegos diferentes por la probabilidad de que el adversario diferencie cada uno.

Lema 3 (Lema de la diferencia)**.** *Sean E_1, E_2, Z eventos donde $E_1 \wedge \overline{Z} \iff E_2 \wedge \overline{Z}$ (es decir, E_1 y E_2 son idénticos a menos que ocurra Z). Entonces,* $|\Pr[E_1] - \Pr[E_2]| \leq \Pr[Z]$.

Demostración. Recordemos que si $A \iff B$, entonces $\Pr[A] = \Pr[B]$. Primero, expandimos E_1 y E_2 en términos de Z:

$$\begin{aligned}
|\Pr[E_1] - \Pr[E_2]| &= \left|\Pr[E_1 \wedge Z] + \Pr\left[E_1 \wedge \overline{Z}\right] - \left(\Pr[E_2 \wedge Z] + \Pr\left[E_2 \wedge \overline{Z}\right]\right)\right| \\
&= |\Pr[E_1 \wedge Z] - \Pr[E_2 \wedge Z]|
\end{aligned}$$

Luego, vemos que $\Pr[E_1 \wedge Z] = \Pr[E_1 \mid Z] \Pr[Z]$ (lo mismo vale para E_2). Además, dadas dos probabilidades a y b, se cumple que $0 \leq |a - b| \leq 1$. Por lo tanto, podemos acotar la diferencia de la siguiente manera:

$$\begin{aligned}
|\Pr[E_1 \wedge Z] - \Pr[E_2 \wedge Z]| &= |\Pr[E_1 \mid Z] \Pr[Z] - \Pr[E_2 \mid Z] \Pr[Z]| \\
&= \Pr[Z] \cdot |\Pr[E_1 \mid Z] - \Pr[E_2 \mid Z]| \\
&\leq \Pr[Z]
\end{aligned}$$

□

Un último lema que necesitaremos es la *desigualdad de Boole* o *union bound*, que establece que la probabilidad de la unión de dos eventos es menor o igual a la suma de sus probabilidades.

Lema 4 (Desigualdad de Boole)**.** *Sean $X_0 \ldots X_n$ eventos. Entonces,*

$$\Pr\left[\bigvee_{i=0}^{n} X_i\right] \leq \sum_{i=0}^{n} \Pr[X_i]$$

Demostración. Probamos el lema por inducción sobre n. Para $n = 0$, tenemos $\Pr[X_0] \leq \Pr[X_0]$, lo cual es trivialmente cierto. Ahora, supongamos que el lema es válido para n y probémoslo para $n+1$. Sea $Z = \bigvee_{i=0}^{n} X_i$. Entonces Z es simplemente otro evento y $\Pr[Z] \leq \sum_{i=0}^{n} \Pr[X_i]$ por la hipótesis inductiva.

$$\begin{aligned}
\Pr\left[\bigvee_{i=0}^{n+1} X_i\right] &= \Pr\left[\bigvee_{i=0}^{n} X_i \vee X_{n+1}\right] \\
&= \Pr[Z \vee X_{n+1}] \\
&= \Pr[Z] + \Pr[X_{n+1}] - \Pr[Z \wedge X_{n+1}] \\
&\leq \Pr[Z] + \Pr[X_{n+1}] && (\Pr[E] \geq 0 \ \forall E) \\
&\leq \sum_{i=0}^{n} \Pr[X_i] + \Pr[X_{n+1}] && \text{(hipótesis inductiva)} \\
&= \sum_{i=0}^{n+1} \Pr[X_i]
\end{aligned}$$

Por lo tanto, el lema se cumple para $n+1$. Por inducción, el lema se cumple para todo $n \geq 0$. □

3 Protocolo de Objetos Tuta

Tuta proporciona un cliente de calendario de código abierto y multiplataforma en TypeScript, disponible en [Gmba]. Estamos analizando el commit `8ede2cdcd20bde` del repositorio.

En general, la información se almacena en *objetos* que se representan en el cliente como objetos de JavaScript. Estos objetos incluyen datos específicos de usuario, como eventos de calendario, invitaciones y los propios calendarios. Todos los objetos necesarios para el funcionamiento del software se almacenan cifrados en el servidor y, como Tuta se esfuerza por ofrecer cifrado de extremo a extremo, es el cliente quien se encarga del cifrado *y* descifrado de los datos. A continuación describimos el protocolo mediante el cual se almacenan los objetos, el sistema de gestión de claves utilizado para administrar las claves de cifrado, el protocolo de autenticación de usuarios, así como el esquema de cifrado utilizado para los objetos de Tuta. Más adelante, en Sección 4 y Sección 5, analizaremos el protocolo de inicio de sesión y el esquema de cifrado utilizados por Tuta, respectivamente.

Gestión de Claves. Figura 1 explica la jerarquía de claves utilizada para el cifrado de datos, donde cada usuario tiene una única *clave de cifrado de claves* (kek, por *key-encryption-key*) que se utiliza para cifrar una clave maestra (la *clave de grupo de usuario*). A su vez, esta clave maestra cifra las claves de sesión por objeto. Si quisiéramos almacenar un objeto en el servidor, la clave de sesión es la que finalmente cifra el objeto en cuestión utilizando un esquema de cifrado autenticado. La kek se deriva de la contraseña del usuario mediante una función hash con sal, y tanto la clave de grupo de usuario como las claves de sesión se almacenan cifradas en el servidor, según el diagrama. El esquema es bastante similar al utilizado por el esquema CSS de Backendal et al. [Bac+24]. Allí, las "claves de archivo", comparables a nuestras claves de sesión, se cifran con una "clave maestra" (la clave de grupo de usuario en el caso de Tuta) que, a su vez, se cifra con una kek. Aunque uno podría sentirse seguro por la similitud de los esquemas y el hecho de que Backendal et al. demostraron que su esquema era seguro, mostraremos cómo Tuta aun así logra contener fallas de seguridad.

Este esquema hace que tanto la kek como la clave de grupo de usuario actúen como una especie de clave maestra, ya que pueden descifrar transitivamente cada objeto en posesión del usuario. Dado que el usuario puede cambiar de contraseña a voluntad, la kek le permite cambiarla sin necesidad de volver a cifrar todos los objetos del sistema; solo es necesario volver a cifrar la clave de grupo de usuario.

Aunque una rotación global de claves podría hacerse eficiente utilizando un mecanismo similar al de la actualización de la contraseña de usuario, si el usuario actualizara la "clave de grupo de usuario" los únicos datos que deberían recifrarse serían las claves de sesión; excepto que Tuta solo admite la recuperación e inserción de objetos completos y las claves se almacenan junto con los objetos. Esto implica que, al cambiar la clave de grupo de usuario, cada objeto debe descargarse y subirse completamente para reemplazar la clave de sesión cifrada. Aunque la porción de datos cambiada es pequeña, la sobrecarga total es significativa e ineficiente.

Compartir Objetos. Tuta basa su sistema de gestión de claves en el concepto de *grupos*. Cada usuario tiene un llamado *grupo de usuario*, que es un grupo unitario que contiene únicamente al propio usuario. Además, los usuarios pueden crear grupos que contengan a otros usuarios, lo que permite compartir objetos, como calendarios y eventos. Esta organización permite compartir objetos personales de forma fluida, ya que los objetos propios de un usuario, al formar parte ya de un grupo (es decir, el "grupo de usuario"), habilitan el uso compartido con solo cambiar el grupo propietario del objeto.

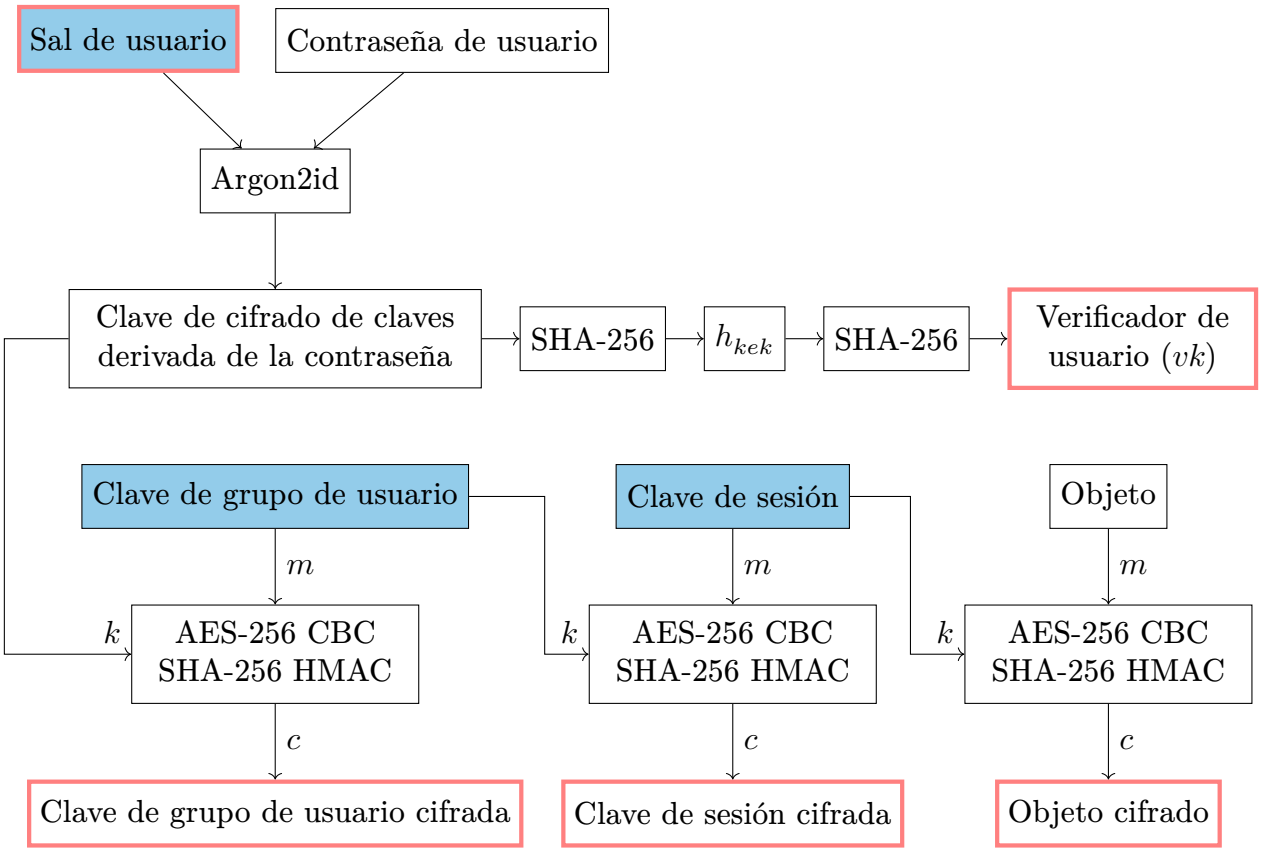

Figura 1: Sistema de gestión de claves de Tuta. La clave de grupo de usuario es una clave maestra que cifra la clave de sesión para cada objeto que posee el grupo. Todos los campos marcados en rojo se almacenan en el servidor sin ninguna modificación adicional. Los campos con fondo azul se generan aleatoriamente del lado del cliente. Las etiquetas indican los parámetros de los algoritmos de cifrado/descifrado: k clave, m mensaje en claro y c ciphertext.

Figura 2 muestra la estructura usuario-grupo utilizada por Tuta. Podemos ver cómo el uso compartido de objetos es posible gracias a la estructura de pertenencia a grupos. Si Alice quiere compartir un evento con Bob, puede crear un grupo que los contenga a ambos. Una vez que Bob acepta la invitación al grupo (que incluye la clave del grupo), ambos pueden descifrar el evento. Además, ambos usuarios pueden acceder a cualquier evento bajo el mismo grupo, ya que todas las claves de sesión utilizadas para cifrarlos están envueltas con la clave de grupo compartida, a la cual ambos tienen acceso.

Cada usuario almacena su copia de la clave de grupo cifrada con su clave maestra dentro de sus "pertenencias a grupo". Adicionalmente, las entidades "grupo" almacenan de nuevo la clave del grupo, pero esta vez cifrada bajo la clave del "grupo propietario". Este "grupo propietario" puede estar representado por un único usuario, o por otro grupo que actúa como administrador con derechos de acceso equivalentes a los datos del grupo con el resto de los administradores.

Cifrado de Objetos. En Tuta, todos los objetos, incluidos los eventos, se representan como objetos de JavaScript y se serializan entre el back-end y el front-end utilizando el formato JSON según un *Type Model*. Estos son descripciones JSON autogeneradas del esquema de la base de datos y se utilizan para guiar el cifrado. El cifrado de objetos se realiza utilizando una "clave de sesión" generada aleatoriamente para cada objeto. En la Apéndice D proporcionamos un TypeModel para el objeto `CalendarEvent` extraído del código fuente de Tuta.

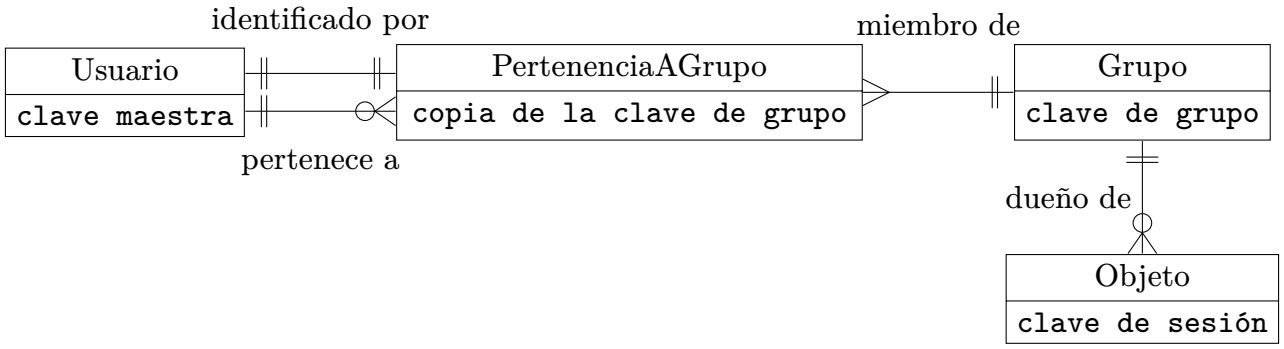

Figura 2: Estructura de usuario-grupo de Tuta presentada en notación "pata de gallo" [Eve76]. La relación "identificado por" da cuenta del grupo unitario que se requiere para cada usuario. Cada clave especificada debajo de cada entidad se almacena cifrada en el servidor, siguiendo la Figura 1.

Se han omitido algunos campos que se presume están relacionados con la migración de la base de datos para mayor claridad. Cada propiedad en un TypeModel describe, entre otras cosas, su tipo; si el campo está cifrado con la clave de sesión; si el campo puede ser nulo o es un arreglo (el campo "cardinality"); y si el campo es constante (inmutable). Los TypeModels definen dos tipos diferentes de campos:

- Los llamados `values`: datos de tipos primitivos. Para tipos no primitivos de JSON, se necesita alguna codificación (por ejemplo, fechas de JS como marcas de tiempo de UNIX, cadenas de bytes como cadenas en base 64, etc.).
- Y las `associations`: arreglos (de longitud arbitraria, unitarios o vacíos) de otros objetos de Tuta. Se pueden considerar como uniones en una base de datos relacional.

Es importante notar que algunos campos no están cifrados, notablemente el ID del objeto. Otros, como `_ownerEncSessionKey`, están *marcados* como no cifrados, pero recordemos que esta afirmación es solo con respecto a la clave de *sesión*; en realidad, el valor viene cifrado con la clave del *grupo propietario*, como su nombre indica. Entonces, el campo `encrypted` se establece en `false` para permitir que el usuario acceda a la clave de sesión con su clave de propietario; de lo contrario, habría un problema del huevo y la gallina al recuperar la clave de sesión. Si bien no tenemos acceso al código del lado del servidor de Tuta, la estructura del TypeModel nos lleva a creer que el servidor se basa en una base de datos NoSQL, donde los objetos se almacenan tal cual, después de separar los campos de `association`.

Autenticación de Usuarios. Nuestro último punto de interés es el protocolo de autenticación de usuarios, es decir, el proceso de inicio de sesión y registro de usuarios. La Figura 3 muestra el protocolo utilizado por Tuta para autenticar a los usuarios. Podemos correlacionar la variable *kek* con la clave de cifrado de claves derivada de la contraseña de la Figura 1.

El proceso de autenticación de usuarios de Tuta involucra dos funciones hash: Argon2id [BDK16] y SHA-256 [ST15]. Por simplicidad, las denotaremos como H_1 y H_2, respectivamente. El cliente utiliza la función hash con sal H_1 para derivar la clave de cifrado de claves *kek*. Luego, obtiene un hash de la *kek* aplicando H_2 y envía el resultado h_{kek} al servidor. Finalmente, el servidor calcula el verificador *vk* aplicando H_2 nuevamente al valor recibido, almacenándolo en una base de datos.

Este proceso es similar a un esquema de hash de contraseñas habitual, donde la contraseña del usuario se hashea con una sal antes de ser almacenada en una base de datos. La principal

diferencia es que no solo estamos tratando con contraseñas, sino con claves de cifrado derivadas de contraseñas. Como veremos en la Sección 4, esta es una diferencia crucial, ya que permite a un atacante romper el cifrado de un usuario.

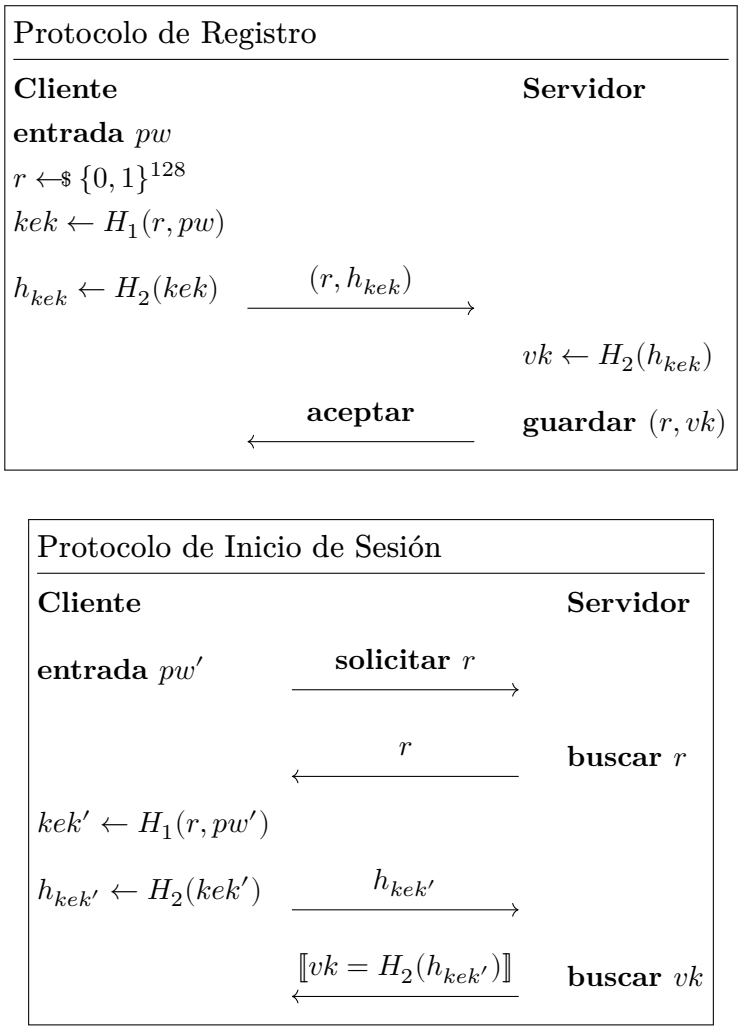

Figura 3: Protocolo de autenticación de usuario de Tuta. H_1 y H_2 se instancian con Argon2id y SHA-256, respectivamente. Cada sal r y verificador vk dependen del usuario. Aquí, la verificación $[\![vk = H_2(h_{kek'})]\!]$ devuelve **aceptar** si la igualdad se cumple y **rechazar** en caso contrario.

Perder la seguridad en cualquiera de ellos puede llevar a una brecha de seguridad en todo el sistema.

4 Autenticación de Usuarios

En esta sección, analizaremos el esquema de autenticación de Tuta propuesto en la Figura 3. Analizaremos la seguridad del protocolo de autenticación en escenarios con adversarios de supuestos progresivamente más débiles. Primero, analizaremos la seguridad del protocolo en sí contra un adversario snapshot, es decir, uno que ha obtenido acceso a la base de datos de cuentas de usuario de Tuta pero no tiene acceso adicional a los sistemas de Tuta.

Luego, continuaremos con el caso semi-honesto, donde ahora Tuta es el adversario pero debe seguir el protocolo al pie de la letra. En este caso, nos centramos en la seguridad de los archivos cifrados que se almacenan en el servidor, pero con el agregado del hash de la contraseña del usuario como entrada.

Finalmente, analizamos el escenario de servidor malicioso, donde el adversario tiene control total sobre el servidor y el protocolo podría no seguirse estrictamente, por ejemplo, calculando mal los pasos del protocolo, mintiendo sobre cierta información, etc.

4.1 Seguridad contra un Adversario Snapshot

Recordemos el protocolo de autenticación presentado en la Figura 3 y, en particular, el protocolo de inicio de sesión de usuario. A primera vista, parece un esquema de autenticación estándar, excepto por el uso de un "triple hash", ya que el verificador final es $vk := H_2(H_2(H_1(r, pw)))$. Explicaremos la razón detrás de este triple hash construyendo desde los casos más simples.

Sin hashing o hashing simple. Si el cliente enviara directamente pw al servidor para su verificación, cualquier fuga de la base de datos o del canal revelaría la contraseña en claro. Además, un compromiso del servidor expone inmediatamente pw. También, si el servidor almacenara $kek := H_1(r, pw)$, el esquema sería inseguro, dado que kek funciona como una de las claves maestras.

Doble hash. Para el doble hash, hay dos alternativas: o bien el cliente computa ambos hashes $h_{kek} := H_2(H_1(r, pw))$ y lo envía al servidor; o bien el cliente envía $kek := H_1(r, pw)$ al servidor, que computa $h_{kek} := H_2(kek)$ y lo almacena como verificador.

Cómputo de h_{kek} del lado del cliente. En este caso, h_{kek} se computa del lado del cliente y el servidor lo almacena directamente, sin modificarlo, en la base de datos de usuarios. Al iniciar sesión, el servidor vuelve a recibir un intento de inicio de sesión h_{kek} y lo compara contra el valor almacenado. El problema radica en que un actor malicioso que haya obtenido acceso a la base de datos de Tuta puede usar el h_{kek} almacenado para autenticarse como cualquier usuario, simplemente transmitiendo h_{kek} como intento de login. Como no se realizan comprobaciones adicionales del lado del servidor, este permitiría el acceso, lo cual es indeseable. Este problema surge de que no hay "cegado" del verificador a nivel de base de datos.

Pero, ¿qué se gana con este ataque? Si asumimos que las afirmaciones de Tuta sobre el cifrado de extremo a extremo son ciertas, el atacante no podría descifrar ningún dato del usuario, ya que no tiene acceso ni a la clave de grupo de usuario ni a la kek. Por lo tanto, el atacante solo obtendría acceso a los datos en texto plano del usuario. En particular, obtendría acceso a los datos que se almacenan sin cifrar, lo que, según el software cliente de Tuta, incluye las horas de inicio de los eventos en formato aleatorio,[3] entre otros. Aunque esta

[3]Según `CalendarUtils.ts`, línea 361

fuga de información puede no ser crítica, sigue siendo una preocupación de privacidad, ya que podría permitir a un atacante tercero inferir alguna información sobre el usuario, como su zona horaria, horario de trabajo, etc.. Esto significa que, para mantener *todos* los datos del usuario privados, almacenar $vk := H_2(h_{kek})$ es lo correcto, ya que de lo contrario podría producirse alguna fuga de datos.

Cómputo de h_{kek} del lado del servidor. Si computar ambos hashes en el cliente no fue suficiente, podemos intentar dividir el trabajo entre el servidor y el cliente. Ahora analizamos el caso en que el cliente computa $kek := H_1(r, pw)$ y el servidor computa el verificador $h_{kek} := H_2(kek)$ y lo guarda en la base de datos. Al computar h_{kek} en el servidor, evitamos que un adversario pueda iniciar sesión enviando simplemente su verificador: en su lugar, tendría que enviar la kek del usuario.

Sigue habiendo un problema, igual. Como en el login el servidor recibe la kek sin protección, un *servidor malicioso* o *semi-honesto* puede acceder a todos los datos del usuario, dado que la kek actúa como clave maestra.

Triple hash. Para mitigar los problemas de ambos esquemas anteriores, podemos añadir una capa adicional, almacenando $vk := H_2(h_{kek})$ en la base de datos de usuarios y enviando h_{kek} al servidor en el login. Esto resuelve los problemas previos: se almacena un verificador no utilizable directamente para autenticación y, a la vez, se evita exponer la kek del usuario al servidor. Por lo tanto, concluimos que almacenar $vk := H_2(h_{kek})$ y transmitir únicamente h_{kek} al servidor es el esquema más resiliente—incluso en el peor de los casos, los datos cifrados del usuario se mantienen a salvo.

4.2 Seguridad del Esquema de Cifrado en el Modelo Semi-Honesto

Ahora centramos nuestra atención en analizar la ventaja que este protocolo le da a un atacante que sí sigue el protocolo, pero que aun así puede intentar acceder a la información privada del usuario. Demostraremos que proporcionar a un atacante (ya sea para los juegos de IND-CPA o de integridad de texto cifrado) la salida del protocolo no le dará una ventaja adicional, si no es por la adivinanza de contraseñas. En particular, tomamos como "la salida del protocolo" a $H_2(H_1(r, pw))$, ya que en el contexto de un atacante de servidor semi-honesto, reciben exactamente $H_2(H_1(r, pw))$. En otras palabras, ignoramos el hecho de que una segunda aplicación de H_2 pueda ser utilizada en la base de datos, ya que tal paso no es relevante para un atacante de servidor semi-honesto porque controla el canal de comunicación. Adicionalmente, modelaremos H_1 y H_2 como oráculos aleatorios RO_1 y RO_2, respectivamente, lo que nos permitirá razonar sobre la seguridad del esquema con funciones hash idealizadas.

Para este fin, presentamos la noción de *IND-CPA con entrada auxiliar* (AI-IND-CPA para abreviar), y de *integridad de texto cifrado con entrada auxiliar* (AI-INT).

Definición 12 (Seguridad IND-CPA con entrada auxiliar)**.** *Dado un esquema de cifrado de clave secreta $\mathcal{E} = (E, D)$ con espacio de claves $\mathcal{K}$, un 1-muestreador $\mathcal{PW}$, y un espacio de hash $\mathcal{H}$, definimos el juego IND-CPA con entrada auxiliar para un $b \in \{0, 1\}$ de la siguiente manera:*

$\mathsf{Game}^{\text{AI-IND-CPA}}_{\mathcal{E},\mathcal{PW},\mathcal{H},b}$	$\mathcal{O}(m_0, m_1)$
$RO_1 \leftarrow\!\!\$\ \text{Funcs}[\mathcal{PW}, \mathcal{K}]$	**assert** $\lvert m_0\rvert = \lvert m_1\rvert$
$RO_2 \leftarrow\!\!\$\ \text{Funcs}[\mathcal{K}, \mathcal{H}]$	$c \leftarrow E(k, m_b)$
$(pw, z) \leftarrow\!\!\$\ \mathcal{PW}$	**return** c
$k \leftarrow RO_1(pw)$	
$h \leftarrow RO_2(k)$	
$\hat{b} \leftarrow \mathcal{A}^{RO_1, RO_2, \mathcal{O}}(h, z)$	
return $[\![b = \hat{b}]\!]$	

Decimos que $(\mathcal{E}, \mathcal{PW})$ proporciona seguridad $(\varepsilon, t, q_o, q_{RO_1}, q_{RO_2})$-AI-IND-CPA si un adversario $\mathcal{A}$ con un tiempo de ejecución $\leq t$, $\leq q_o$ consultas a $\mathcal{O}$, $\leq q_{RO_1}$ consultas a RO_1, y $\leq q_{RO_2}$ consultas a RO_2 tiene una ventaja $\mathsf{Adv}(\mathsf{Game}^{\text{AI-IND-CPA}}, \mathcal{A})$ definida por

$$\left|\Pr\left[\mathsf{Game}^{\text{AI-IND-CPA}}_{\mathcal{E},\mathcal{PW}}(\mathcal{A}, 0) \Rightarrow 1\right] - \Pr\left[\mathsf{Game}^{\text{AI-IND-CPA}}_{\mathcal{E},\mathcal{PW}}(\mathcal{A}, 1) \Rightarrow 1\right]\right| \leq \varepsilon$$

Esta noción de seguridad es muy similar a la IND-CPA usual, excepto que podemos cuantificar la ventaja que se obtiene cuando el retador le proporciona la clave de cifrado hasheada h al atacante. En cierto modo, esta definición integra el protocolo de autenticación de Tuta en el juego IND-CPA. Esto nos permite razonar sobre él como si fuera un juego regular. Para aplicar este razonamiento al protocolo de autenticación de Tuta, reemplazamos H_1 y H_2 por RO_1 y RO_2, respectivamente. En el caso de H_1, podemos pensar en ella como una familia de funciones, donde la sal selecciona una función de la familia. Si quisiéramos imitar este comportamiento en el modelo de oráculo aleatorio, el oráculo aleatorio se elegiría uniformemente al azar del conjunto $\text{Funcs}[\mathcal{S} \times \mathcal{PW}, \mathcal{K}]$. Luego, se generaría una sal aleatoria para indexar el oráculo aleatorio y se le daría a $\mathcal{A}$, ya que es información pública. Con o sin la sal, obtenemos el mismo resultado: un oráculo aleatorio $RO_1 : \mathcal{PW} \to \mathcal{K}$. Omitiendo la sal, $\mathcal{A}$ recibe un argumento menos en su procedimiento, y el resto permanece igual.

Después de modelar el protocolo de autenticación como un esquema AI-IND-CPA, podemos demostrar el siguiente resultado.

Teorema 1 (Efectos del protocolo de autenticación sobre IND-CPA)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta que proporciona seguridad $(\varepsilon_{CPA}, t, q_{CPA})$-IND-CPA, y $\mathcal{PW}$ un 1-muestreador de contraseñas $(\varepsilon_{PG}, t, q_{PG}, 0)$-resistente a ataques de recuperación. Entonces, $(\mathcal{E}, \mathcal{PW})$ proporciona seguridad $(\varepsilon', t', q_{CPA}, q_{PG}, q_{RO_2})$-AI-IND-CPA con $t' \approx t$, y*

$$\varepsilon' = \varepsilon_{CPA} + 2\varepsilon_{PG} + \frac{2q_{RO_2}}{|\mathcal{K}|}$$

Demostración. En esta prueba nos centraremos en la versión de adivinanza de bit del juego AI-IND-CPA (de aquí en adelante Game_1). Proponemos un nuevo par de juegos Game_2 y Game_3, similares a Game_1. En el primero, la evaluación de RO_2 se reemplaza por un muestreo aleatorio, desconectando la clave k de la salida de RO_2. Además de los cambios en Game_2, Game_3 tendrá el mismo efecto sobre RO_1, haciendo que k ya no dependa de la contraseña, sino que sea completamente aleatoria.

$\mathsf{Game}_1(\mathcal{A})$	$\mathsf{Game}_2(\mathcal{A})$	$\mathsf{Game}_3(\mathcal{A})$
$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$	$b \leftarrow\$ \{0,1\}$
$RO_1 \leftarrow\$ \text{Funcs}[\mathcal{PW}, \mathcal{K}]$	$RO_1 \leftarrow\$ \text{Funcs}[\mathcal{PW}, \mathcal{K}]$	$RO_1 \leftarrow\$ \text{Funcs}[\mathcal{PW}, \mathcal{K}]$
$RO_2 \leftarrow\$ \text{Funcs}[\mathcal{K}, \mathcal{H}]$	$RO_2 \leftarrow\$ \text{Funcs}[\mathcal{K}, \mathcal{H}]$	$RO_2 \leftarrow\$ \text{Funcs}[\mathcal{K}, \mathcal{H}]$
$pw \leftarrow\$ \mathcal{PW}$	$pw \leftarrow\$ \mathcal{PW}$	$k \leftarrow\$ \mathcal{K}$
$k \leftarrow RO_1(pw)$	$k \leftarrow\$ RO_1(pw)$	$h \leftarrow\$ \mathcal{H}$
$h \leftarrow RO_2(k)$	$h \leftarrow\$ \mathcal{H}$	$\hat{b} \leftarrow \mathcal{A}^{RO_1, RO_2, \mathcal{O}}(h)$
$\hat{b} \leftarrow \mathcal{A}^{RO_1, RO_2, \mathcal{O}}(h)$	$\hat{b} \leftarrow \mathcal{A}^{RO_1, RO_2, \mathcal{O}}(h)$	**return** $[\![b = \hat{b}]\!]$
return $[\![b = \hat{b}]\!]$	**return** $[\![b = \hat{b}]\!]$	

Adivinando la clave. Inspeccionando Game_1 y Game_2, podemos ver que $\mathcal{A}$ no puede distinguirlos, excepto por la consulta de RO_2 evaluado en k. Este es el caso porque en Game_1, $\mathcal{A}$ recibiría h de la evaluación, y en Game_2, $\mathcal{A}$ recibiría un valor aleatorio, no relacionado con h o k. Llamamos al evento correspondiente a la consulta de $\mathcal{A}$ de RO_2 evaluado en k, Guess_k.

Debido a que RO_1 es un oráculo aleatorio, k está uniformemente distribuida, es decir, para una clave k uniformemente aleatoria, un intento k' tiene una probabilidad $\Pr_{k \sim U(\mathcal{K})}[k' = k] = |\mathcal{K}|^{-1}$. Denotamos los intentos realizados por $\mathcal{A}$ como $\vec{k} = (k_1, \ldots, k_{q_{RO_2}})$. Entonces, la probabilidad de que cualquiera de los intentos en $\vec{k}$ sea igual a una k uniformemente aleatoria (y, por lo tanto, la probabilidad de que ocurra Guess_k) está acotada por la desigualdad de Boole como

$$\Pr[\mathsf{Guess}_k] = \Pr_{k \sim \mathcal{K}}\left[\bigvee_{i=1}^{q_{RO_2}} k_i = k\right] \leq \frac{q_{RO_2}}{|\mathcal{K}|}$$

Adivinando la contraseña. En cuanto a Game_2, $\mathcal{A}$ no puede distinguirlo de Game_3, excepto por la consulta de RO_1 evaluado en pw (es decir, una adivinanza correcta de la contraseña). A tal evento lo llamamos Guess_{pw}. Usando el Lemma 6 de la Apéndice A, podemos acotar $\Pr\left[\mathsf{Guess}_{pw}\right]$ por la ventaja de un adversario $\mathcal{B}_{\mathsf{Guess}}$ en el juego de adivinanza de contraseñas contra $\mathcal{PW}$.

$$\Pr\left[\mathsf{Guess}_{pw}\right] \leq \mathsf{Adv}(\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}, \mathcal{B}_{\mathsf{Guess}})$$

Atacando $\mathcal{E}$. Adicionalmente, podemos ver Game_3 como una instancia del juego IND-CPA debido a la falta de relevancia tanto de pw como de h para la mecánica del juego. Simplemente podemos usar $\mathcal{A}$ en la versión de adivinanza de bit del juego IND-CPA de $\mathcal{E}$, presentando un atacante $\mathcal{B}$ que es un envoltorio elemental alrededor de $\mathcal{A}$ que crea una h aleatoria y se la reenvía a $\mathcal{A}$. Esto significaría que la ventaja de $\mathcal{A}$ en Game_3 es la misma que la ventaja de $\mathcal{B}$ en la versión de adivinanza de bit del juego IND-CPA contra $\mathcal{E}$. Entonces,

$$\mathsf{Adv}(\mathsf{Game}_3, \mathcal{A}) = \mathsf{Adv}(\overline{\mathsf{Game}^{\text{IND-CPA}}_{\mathcal{E}}}, \mathcal{B}) \leq \varepsilon/2 \qquad \text{(por hipótesis)}$$

Acotando la ventaja de $\mathcal{A}$. Podemos acotar la ventaja de $\mathcal{A}$ en el juego AI-IND-CPA de la siguiente manera. Denotemos $\Pr[\mathsf{Game}_i(\mathcal{A}) \Rightarrow 1]$ como $\Pr[W_i]$.

$$\begin{aligned}
\mathsf{Adv}(\mathsf{Game}_1, \mathcal{A}) &= \left|\Pr[W_1] - \frac{1}{2}\right| = \left|\Pr[W_1] - \Pr[W_2] + \Pr[W_2] - \frac{1}{2}\right| \\
&\leq \Pr[\mathsf{Guess}_k] + \left|\Pr[W_2] - \frac{1}{2}\right| && \text{(lema de la diferencia)} \\
&= \Pr[\mathsf{Guess}_k] + \left|\Pr[W_2] - \Pr[W_3] + \Pr[W_3] - \frac{1}{2}\right| \\
&\leq \Pr[\mathsf{Guess}_k] + \Pr\left[\mathsf{Guess}_{pw}\right] + \mathsf{Adv}(\mathsf{Game}_3, \mathcal{A}) && \text{(lema de la diferencia)} \\
&= \Pr[\mathsf{Guess}_k] + \Pr\left[\mathsf{Guess}_{pw}\right] + \mathsf{Adv}(\overline{\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}}}, \mathcal{B}) \\
&\leq \frac{q_{RO_2}}{|\mathcal{K}|} + \varepsilon_{PG} + \frac{\varepsilon_{CPA}}{2}
\end{aligned}$$

Luego, podemos usar el Lemma 2 para acotar la ventaja de $\mathcal{A}$ en el juego AI-IND-CPA de la siguiente manera:

$$\begin{aligned}
\mathsf{Adv}(\mathsf{Game}_{\mathcal{E},\mathcal{PW}}^{\text{AI-IND-CPA}}, \mathcal{A}) = 2 \cdot \mathsf{Adv}(\mathsf{Game}_1, \mathcal{A}) &\leq 2\left(\Pr[\mathsf{Guess}_k] + \Pr\left[\mathsf{Guess}_p w\right]\right) + \varepsilon_{CPA} \\
&\leq \frac{2q_{RO_2}}{|\mathcal{K}|} + 2\varepsilon_{PG} + \varepsilon_{CPA}
\end{aligned}$$

□

Este resultado está en línea con la expectativa de que, en los esquemas basados en contraseñas, la seguridad de la adivinanza de la distribución de contraseñas es el factor limitante en la seguridad de todo el sistema. Como tal, la seguridad del esquema en su conjunto se ve degradada tanto por este hecho como por el ataque de fuerza bruta a la clave maestra $k := RO_1(pw)$.

Sin embargo, como no hay vulnerabilidades estructurales, podemos concluir que el protocolo es tan seguro como puede serlo, dadas las restricciones del cifrado basado en contraseñas y el hecho de compartir la clave hasheada.

Se puede definir una noción similar de integridad donde al atacante se le da la salida del protocolo de autenticación y luego se le pide que falsifique un texto cifrado. A continuación, enunciamos la definición de AI-INT, así como los límites de integridad del protocolo, pero dejamos la prueba en la Apéndice B.

Definición 13 (Integridad de texto cifrado con entrada auxiliar (AI-INT-CTXT))**.** *Dado un esquema de cifrado de clave secreta $\mathcal{E} = (E, D)$ con espacio de claves $\mathcal{K}$ y símbolo de rechazo $\perp$, un 1-muestreador $\mathcal{PW}$, y un espacio de hash $\mathcal{H}$, definimos el juego de integridad de texto cifrado con entrada auxiliar de la siguiente manera:*

$\mathsf{Game}_{\mathcal{E},\mathcal{PW},\mathcal{H}}^{\text{AI-INT-CTXT}}(\mathcal{A})$	$\mathcal{O}(m)$
$RO_1 \leftarrow\!\!\$\ \text{Funcs}[\mathcal{PW}, \mathcal{K}]$	$c \leftarrow E(k, m)$
$RO_2 \leftarrow\!\!\$\ \text{Funcs}[\mathcal{K}, \mathcal{H}]$	$Q \leftarrow Q \cup \{c\}$
$pw \leftarrow\!\!\$\ \mathcal{PW}$	**return** c
$k \leftarrow RO_1(pw)$	
$h \leftarrow RO_2(k)$	
$Q \leftarrow \emptyset$	
$c^* \leftarrow \mathcal{A}^{RO_1, RO_2, \mathcal{O}}(h)$	
return $[\![D(k, c^*) \neq \perp \wedge c^* \notin Q]\!]$	

Decimos que $(\mathcal{E}, \mathcal{PW})$ *proporciona* $(\varepsilon, t, q_o, q_{RO_1}, q_{RO_2})$*-AI-INT-CTXT si un adversario* $\mathcal{A}$ *con un tiempo de ejecución* $\leq t$*, haciendo* $\leq q_o$ *consultas a* $\mathcal{O}$*,* $\leq q_{RO_1}$ *consultas a* RO_1*, y* $\leq q_{RO_2}$ *consultas a* RO_2 *tiene una ventaja*

$$\Pr\left[\mathsf{Game}^{\text{AI-INT-CTXT}}_{\mathcal{E},\mathcal{PW}}(\mathcal{A}) \Rightarrow 1\right] \leq \varepsilon$$

Esta definición está en línea con la de AI-IND-CPA, adaptada al juego regular de integridad de texto cifrado. La diferencia es que al atacante se le da la salida del protocolo de autenticación, que es un hash de la clave de cifrado.

Teorema 2. *Sea* $\mathcal{E} = (E, D)$ *un esquema de cifrado de clave secreta que proporciona* $(\varepsilon_{INT}, t, q_{INT})$*-INT-CTXT, y* $\mathcal{PW}$ *un 1-muestreador* $(\varepsilon_{PG}, t, q_{PG}, 0)$*-resistente a ataques de recuperación. Entonces,* $(\mathcal{E}, \mathcal{PW})$ *proporciona* $(\varepsilon', t', q_{INT}, q_{PG}, q_{RO_2})$*-AI-INT-CTXT con* $t' \approx t$*, y*

$$\varepsilon' = \varepsilon_{INT} + \varepsilon_{PG} + \frac{q_{RO_2}}{|\mathcal{K}|}$$

Este resultado es análogo al Teorema 1, y muestra que el protocolo de autenticación no filtra ninguna información que permita a un atacante falsificar un texto cifrado, excepto por la ventaja de adivinanza de contraseñas y por el ataque de fuerza bruta a la clave.

4.3 Rompiendo el hash con sal en el Modelo de Servidor Malicioso

Al centrarnos en el paradigma del servidor malicioso, uno puede notar que el valor de la sal, aunque inicialmente es generado por el cliente, luego es solicitado a demanda en el momento del inicio de sesión y transmitido por el servidor, lo que presenta un punto donde un servidor malicioso puede actuar de manera desleal. En tal caso, el servidor puede enviar una sal diferente r', igual para todos los usuarios, obteniendo un nuevo valor $RO_2(RO_1(r', pw))$. Esto llevaría a que un servidor malicioso compile una base de datos de hashes de contraseñas con la misma sal, eludiendo las protecciones contra ataques de diccionario que proporciona una sal aleatoria por usuario.

A efectos de análisis, modelaremos las funciones H_1 y H_2 como oráculos aleatorios independientes RO_1 y RO_2, respectivamente, de modo que la composición $RO_2 \circ RO_1$ se comporte como una única función aleatoria ante el adversario.

Sintaxis preliminar. En un intento de formalizar este ataque, consideramos tanto el protocolo de autenticación como el de registro de cuentas en la sintaxis de Backendal et al., como se muestra en la Figura 4. Esto nos permite analizar los protocolos de una manera más estructurada, así como proporcionar una prueba formal del ataque.

Esencialmente, esta sintaxis se basa en el concepto de un *protocolo*, un algoritmo que es computado por dos partes en una configuración cliente-servidor. En el escenario de servidor honesto, el protocolo se ejecuta intercalando los pasos del cliente (que inicia el proceso) y del servidor. En las pruebas, sin embargo, estamos tratando de probar la seguridad del protocolo bajo la hipótesis de un servidor malicioso y, por lo tanto, el atacante puede no seguir estrictamente los pasos marcados como S (que significa Servidor). Tengan en cuenta que todos los pasos del cliente son seguidos fielmente por el retador. Cada protocolo toma como entrada un estado global $st_{\{C,S\}}$, separado entre cliente y servidor; un "estado temporal" local, por ejecución, $st^{tmp}_{\{C,S\}}$; un mensaje de la otra parte $m_{\{C,S\}}$; y, solo en el primer mensaje del protocolo, datos de entrada $in_{\{C,S\}}$. La variable de mensaje se completa automáticamente con lo que la otra parte produce en el paso anterior. La finalización de la ejecución se indica mediante un **success** o **fail**, que establece la variable de *decisión* de la parte correspondiente

$\mathsf{T}_{\mathsf{areg}}{}^{(C:1)}(\varepsilon, \varepsilon, in_C, \varepsilon)$

1 : $pre : in_C.\{aid, pw\}$
2 : $r \leftarrow\!\!\$\ \{0,1\}^{128}$
3 : $kek \leftarrow RO_1(r, in_C.pw)$
4 : $h_{kek} \leftarrow RO_2(kek)$
5 : $m_C \leftarrow (aid, r, h_{kek})$
6 : **success**$_C(m_C)$

$\mathsf{T}_{\mathsf{areg}}{}^{(S:1)}(st_S, \varepsilon, in_S, m_C)$

1 : $pre : st_S.\{acc\}$
2 : $(aid, r, h_{kek}) \leftarrow m_C$
3 : $vk \leftarrow RO_2(h_{kek})$
4 : $acc[aid] \leftarrow (r, vk)$
5 : **success**$_S$

$\mathsf{T}_{\mathsf{auth}}{}^{(C:1)}(st_C, \varepsilon, in_C, \varepsilon)$

1 : $pre : in_C.\{aid, pw\}$
2 : $st_C^{tmp} \leftarrow pw$
3 : $m_C \leftarrow aid$
4 : **return** $(st_C, st_C^{tmp}, \varepsilon, m_C)$

$\mathsf{T}_{\mathsf{auth}}{}^{(C:2)}(st_C, st_C^{tmp}, m_S)$

1 : $r \leftarrow m_S$
2 : $pw \leftarrow st_C^{tmp}$
3 : $kek \leftarrow RO_1(r, pw)$
4 : $st_C \leftarrow kek$
5 : $h_{kek} \leftarrow RO_2(kek)$
6 : $m_C \leftarrow h_{kek}$
7 : **success**$_C(m_C)$

$\mathsf{T}_{\mathsf{auth}}{}^{(S:1)}(st_S, \varepsilon, in_S, m_C)$

1 : $pre : st_S.\{acc\}$
2 : $(aid) \leftarrow m_C$
3 : **if** $aid \notin acc$:
4 : **fail** $_S$
5 : $(r, vk) \leftarrow acc[aid]$
6 : $st_S^{tmp} \leftarrow vk$
7 : $m_S \leftarrow r$
8 : **return** $(st_S, st_S^{tmp}, \varepsilon, m_S)$

$\mathsf{T}_{\mathsf{auth}}{}^{(S:2)}(st_S, st_S^{tmp}, m_C)$

1 : $h_{kek} \leftarrow m_C$
2 : $vk \leftarrow RO_2(h_{kek})$
3 : **if** $vk \neq st_S^{tmp}.vk$:
4 : **fail** $_S$
5 : **success**$_S$

Figura 4: Protocolos de registro de cuenta y autenticación de usuario de Tuta en la sintaxis de [Bac+24]. Omitimos el resto de los protocolos ya que no son relevantes para este análisis. La sintaxis $pre : x.\{y, z\}$ significa el desempaquetado de las variables y y z del objeto x, que se marcan como requeridas al instanciar el protocolo.

$\mathsf{Game}^{\text{proto-ai-rec}}_{\mathcal{P},\ell}(\mathcal{A})$

$T_p \leftarrow \{\}; T_{stC} \leftarrow \{\}$
$n_p \leftarrow 0; n_c \leftarrow 0$
$RO_1 \leftarrow\!\!\$\ \mathsf{Funcs}[\mathcal{PW}, \mathcal{K}]$
$RO_2 \leftarrow\!\!\$\ \mathsf{Funcs}[\mathcal{K}, \mathcal{H}]$
$\sigma \leftarrow \mathcal{A}_0(RO_1, RO_2)$
$(\overrightarrow{pw}, z) \leftarrow\!\!\$\ \mathcal{PW}$
for $(i,j) \in [m] \times [\ell]$
  // Registrar usuario con contraseña $pw[i]$
  $\mathsf{Exec}_{\mathsf{T}_{\mathsf{areg}}}((i,j), \overrightarrow{pw}[i])$
$\overrightarrow{pw}' \leftarrow \mathcal{A}_1^{\mathsf{Cor}, RO_1, RO_2, \mathsf{Step}, \mathsf{Auth}_1}(\sigma, z)$
return $[\![\overrightarrow{pw}' = \overrightarrow{pw}]\!]$

$\mathsf{Cor}(i)$

return $\overrightarrow{pw}[i]$

$\mathsf{Step}(i_p, m)$

$(i_c, st_C^{tmp}, in_C, r) \leftarrow T_p[i_p]$
if $r > 4$: **return** $\perp$ // El protocolo Auth tiene 4 pasos
$(T_{stC}[i_c], st_C^{tmp}, out_C, m') \leftarrow \mathsf{T}_{auth}^{(C:r)}(T_{stC}[i_c], st_C^{tmp}, m)$
$T_p[i_p] \leftarrow (i_c, st_C^{tmp}, in_C, r+1)$
return m'

$\mathsf{Auth}_1((i,j), in_C)$

$in_C.aid \leftarrow (i,j); in_C.pw \leftarrow \overrightarrow{pw}[i]$
$i_c \leftarrow n_c{+}{+}; i_p \leftarrow n_p{+}{+}$
$(T_{stC}[i_c], st_C^{tmp}, out_C, m) \leftarrow \mathsf{T}_{\mathsf{auth}}{}^{(C:1)}(\varepsilon, \varepsilon, in_C, \varepsilon)$
$T_p[i_p] \leftarrow (i_c, st_C^{tmp}, in_C, 2)$
return (i_p, m)

$\mathsf{Exec}_{\Pi}(st_C, st_S, in_C, in_S)$

$(st_C, st_C^{\text{tmp}}, \text{out}_C, \mathbf{m}[1]) \leftarrow \Pi^{(C:1)}(st_C, \varepsilon, in_C, \varepsilon)$
$(st_S, st_S^{\text{tmp}}, \text{out}_S, \mathbf{m}[2]) \leftarrow \Pi^{(S:1)}(st_S, \varepsilon, \text{in}_S, \mathbf{m}[1])$
for $i = 3 \ldots \Pi.\text{SN}$ **do** // Hasta el número máx. de pasos en Π
  $r \leftarrow \lceil i/2 \rceil$; **if** i es impar **do** $P \leftarrow C$ **else** $P \leftarrow S$
  if $\text{out}_P.\text{dec} = \perp$
    $(st_P, st_P^{\text{tmp}}, \text{out}_P, \mathbf{m}[i]) \leftarrow\!\!\$\ \Pi^{(P:r)}(st_P, st_P^{\text{tmp}}, \mathbf{m}[i-1])$
return $(\mathbf{m}, st_C, st_S, \text{out}_C, \text{out}_S)$

Figura 5: El juego de adivinanza de contraseñas con entrada auxiliar para el protocolo de Tuta. Aprovechamos el protocolo creando usuarios con el procedimiento Exec, cortesía de Backendal et al. [Bac+24]

(*dec*) en un valor proporcionado, o $\perp$ en caso contrario. Para una explicación algorítmica detallada de la ejecución del protocolo, véase el procedimiento Exec en la Figura 5. Como referencia, se puede comparar la Figura 3 y la Figura 4, ya que constituyen los mismos protocolos en sintaxis diferente.

Juego de recuperación de contraseñas con preprocesamiento. Definimos el juego de recuperación de contraseñas para protocolos, $\mathsf{Game}^{\text{proto-ai-rec}}_{\mathcal{P},\ell}$, como se muestra en la Figura 5. Esencialmente, este es el juego de recuperación de contraseñas original (AI-REC), adaptado para usar oráculos aleatorios RO_1 y RO_2 (modelando a H_1 y H_2) junto con las tablas T_p y T_{stC} para llevar un registro del estado del protocolo para cada cliente simulado, cada vez que se inicia una ejecución del protocolo Auth.

La Figura 6 muestra un adversario $\mathcal{A}$ para el juego PROTO-AI-REC que utiliza un adversario $\mathcal{B}$ que juega $\mathsf{Game}^{\text{ai-rec}}_{\mathcal{P},\ell,s_{\mathcal{B}},\mathsf{KD}}$. Asumimos que $s_{\mathcal{B}}$ es una sal constante conocida públicamente, como la sal cero constante. Con esto en mente, podemos usar el atacante para probar el siguiente teorema.

$\mathcal{A}_0(RO_1, RO_2)$

return $\mathcal{B}_0(RO_2 \circ RO_1)$

$\mathcal{A}_1^{\mathsf{Cor}, RO_1, RO_2}(\sigma, z)$

1 : $H \leftarrow RO_2 \circ RO_1$
2 : $\vec{k} \leftarrow \mathsf{Keys}(s_{\mathcal{B}})$
3 : $\overrightarrow{pw}' \leftarrow \mathcal{B}_1^{\mathsf{Cor}, H}(s_{\mathcal{B}}, \vec{k}, \sigma', z)$
4 : **return** $\overrightarrow{pw}'$

$\mathsf{Keys}(s_{\mathcal{B}})$

1 : $\vec{k} \leftarrow \vec{0}$ // $\vec{k} \in \mathcal{K}^m$
2 : **for** $i \in [m]$
3 : // Fijar el índice de la sal a 0
4 : $(i_p, aid) \leftarrow \mathsf{Auth}_1((i, 0), \varepsilon)$
5 : $h_{kek} \leftarrow \mathsf{Step}(i_p, s_{\mathcal{B}})$
6 : $\vec{k}[i] \leftarrow h_{kek}$
7 : **return** $\vec{k}$

Figura 6: Adversario para el protocolo de autenticación, basado en un adversario $\mathcal{B}$ de AI-REC para el caso donde $\mathsf{Gen}() := s_{\mathcal{B}}$, $\ell = 1$, y $\mathsf{KD}^H(pw, s) = H(pw \| s)$. El índice de la sal se fija en 0, ya que el caso sin sal es equivalente a un caso con sal constante.

Teorema 3 (Cota inferior de la seguridad del protocolo de autenticación de Tuta)**.** *Sean $\mathcal{PW}$ un m-muestreador, $\ell \in \mathbb{N}$ la cantidad de sales por contraseña, $\mathsf{Gen}() := s_{\mathcal{B}}$, $\mathsf{KD}^H(pw, s) := H(pw \| s)$ y $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ un atacante que juega $\mathsf{Game}^{\text{ai-rec}}_{\mathcal{P},\ell,s_{\mathcal{B}},\mathsf{KD}}$. Entonces, el atacante $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ como se define en la Figura 6 que juega $\mathsf{Game}^{\text{proto-ai-rec}}_{\mathcal{P},\ell}$ con los protocolos definidos en la Figura 4 tiene una ventaja*

$$\mathsf{Adv}(\mathsf{Game}^{\text{proto-ai-rec}}_{\mathcal{PW},\ell}, \mathcal{A}) = \mathsf{Adv}(\mathsf{Game}^{\text{ai-rec}}_{\mathcal{PW},\ell,s_{\mathcal{B}},H}, \mathcal{B})$$

Demostración. Inspeccionando a $\mathcal{A}$, podemos ver que simula adecuadamente el caso donde $\mathsf{Gen}() := s_{\mathcal{B}}$ y $\mathsf{KD}^H(pw, s) := RO_2(RO_1(pw \| s))$, ya que obtiene una derivación de clave para todas las m contraseñas usando la misma sal constante $s_{\mathcal{B}}$ y se la reenvía a $\mathcal{B}$.

Argumentamos que $RO_2 \circ RO_1$ es en sí mismo un oráculo aleatorio, ya que es una composición de dos funciones aleatorias distintas y, por lo tanto, desde el punto de vista de $\mathcal{B}$, su entorno es indistinguible del que se encuentra en $\mathsf{Game}^{\text{ai-rec}}_{\mathcal{P},\ell,\mathsf{Gen},RO_2 \circ RO_1}$.

Luego, como las contraseñas en sí son exactamente las mismas para ambos adversarios, podemos concluir que $\mathcal{A}$ gana si y solo si $\mathcal{B}$ gana, y por lo tanto los dos juegos son equivalentes. □

Comentario. *Es importante mencionar que este resultado está sujeto a la suposición de que el servidor malicioso ha recibido un intento de inicio de sesión de m usuarios que usan las m contraseñas diferentes, y ha manipulado el protocolo a su favor (el procedimiento* Keys *inicia un protocolo $\mathsf{T}_{\mathsf{auth}}$ para cada contraseña). Esto implica que todos estos usuarios han sufrido un único intento de inicio de sesión fallido debido al ataque, ya que proporcionarles una sal incorrecta hace que su inicio de sesión falle, impidiendo que los usuarios puedan descifrar sus datos.*

Este resultado muestra que el protocolo de autenticación de Tuta no proporciona ningún beneficio de seguridad sobre el caso sin sal, ya que un atacante siempre puede recuperar la contraseña con un atacante sin sal (por ejemplo, un ataque de diccionario sobre la función hash $H(pw) := \mathsf{SHA\text{-}256}(\mathsf{Argon2id}(pw \| \vec{0}))$). Las sales no añaden ninguna seguridad y, por lo tanto, el protocolo es vulnerable a ataques de recuperación de contraseñas que se presumían inviables.

Concretamente, podemos derivar corolarios de Farshim y Tessaro [FT21] para tener una idea de la debilidad que introduce este diseño de protocolo. En particular, estos resultados asumen que las contraseñas se muestrean de un m-muestreador uniforme $\mathcal{N}$ de un espacio de

tamaño N,[4] un paso de precomputación de tamaño $S \geq 3m$, y γ elegido de tal manera que $\log \gamma^{-1} \leq m$.

Una simple instanciación del [FT21, Teorema 10] nos da una cota para los sistemas que involucran sales uniformemente aleatorias.

Corolario 1. *[FT21, Teorema 10] Sean e la constante de Napier, $\ell \in \mathbb{N}$, K un espacio de sales y H un oráculo aleatorio. Entonces, dado un m-muestreador uniforme $\mathcal{N}$, S y γ como los anteriores,*

$$\mathsf{Adv}(\mathsf{Game}^{\text{ai-rec}}_{\mathcal{N},\ell,[K],H}, \mathcal{A}) \leq \left(\frac{2e \cdot T}{mN}\right)^m + \frac{m\ell}{K} \cdot \left(\frac{6e \cdot ST}{m^2 N}\right)^m + \frac{m^2\ell^2}{K} + \gamma$$

Por el contrario, los protocolos equivalentes a esquemas sin sal, como el implementado por Tuta, tienen una seguridad comparablemente degradada, según una instanciación del [FT21, Teorema 8].

Corolario 2. *[FT21, Teorema 8] Sean e la constante de Napier, $\ell \in \mathbb{N}$, $\perp$ una única sal constante y H un oráculo aleatorio. Entonces, dado un m-muestreador uniforme $\mathcal{N}$, S y γ como los anteriores,*

$$\mathsf{Adv}(\mathsf{Game}^{\text{ai-rec}}_{\mathcal{N},\ell,\perp,H}, \mathcal{A}) \leq \left(\frac{6ST}{mN}\right)^m + \gamma$$

Si comparamos los términos principales (que tienen mN como denominador), podemos ver que, en un ataque práctico, la degradación en el caso sin sal es significativa, ya que permite al atacante intercambiar tiempo por espacio en el paso de precomputación. En contraste, en el caso de la sal uniforme, este término solo contiene un factor de tiempo, y por lo tanto el atacante se ve obligado a invertir realmente el tiempo para recuperar las contraseñas.

[4] Esto constituye el mejor de los casos ya que, según los volcados de contraseñas existentes, los muestreadores de contraseñas son mucho más predecibles que una distribución uniforme [Nap25].

5 Cifrado

Como el cliente de Tuta está desarrollado en TypeScript, un superconjunto de JavaScript, han adaptado un esquema de cifrado para que funcione sobre el formato de pares clave-valor de los objetos de JavaScript. Además, utilizan JSON como formato de intercambio para comunicarse con el servidor. Mostraremos cómo Tuta adapta su esquema de cifrado a los objetos de JavaScript, demostrando que la integridad del texto cifrado se pierde parcialmente en el proceso.

El formato JSON, especificado en RFC 8259 [Bra17], es un formato de intercambio de datos legible por humanos cuyo uso se disparó desde la popularización de los frameworks de JavaScript para desarrollo web y más allá. Dado que el front-end de Tuta está desarrollado en TypeScript, es natural que el formato JSON se use extensivamente. Esto plantea el problema de su manejo al interactuar con primitivas de cifrado.

Definimos las dos funciones de conversión JSON $JSON.stringify$ y $JSON.parse$, que se utilizan ampliamente en la base de código de Tuta y son las funciones de conversión primarias en JavaScript. La primera es lo que el estándar denomina un *generador* de JSON: dada un objeto de JavaScript o devuelve una representación textual conforme a JSON de o. Nótese que, en JavaScript, esto ignora las propiedades cuyo valor es una función o `undefined`. La segunda es un *parser*: toma una representación conforme a JSON de un objeto y devuelve un objeto de JavaScript con las mismas propiedades. Como el formato JSON permite claves duplicadas, un texto JSON puede no decodificar a un único objeto de JavaScript, dependiendo del parser.[5]

En la Figura 7 se presenta un diagrama de los formatos del proceso de cifrado. Tanto el espacio de los textos en claro como el de los textos cifrados son el conjunto de objetos de JavaScript limitados a objetos que contienen tipos primitivos (cadenas de caracteres, números, booleanos, nulos), otros objetos (hasta una profundidad $d < \infty$), y arreglos.

Aunque Tuta utiliza el formato JSON para la comunicación de red, podemos ignorarlo sin problemas, ya que asumimos su transmisión correcta (es decir, no intervienen parsers o generadores de JSON adicionales en el almacenamiento del texto cifrado). Además, JavaScript nos obliga a utilizar claves únicas bajo una representación textual (por ejemplo, el número 1 y la cadena `"1"` acceden a la misma clave en el objeto). Más adelante, en la Subsección 5.4, capturaremos los matices del formato JSON, sus parsers, y finalmente definiremos un esquema de cifrado resiliente sobre dicho formato.

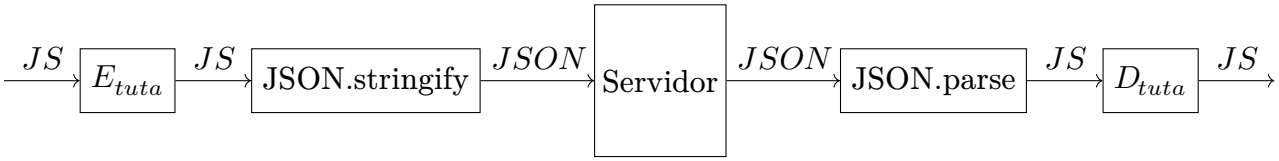

Figura 7: Formatos de objeto utilizados en el esquema de cifrado de Tuta $\mathcal{E}_{tuta} = (E_{tuta}, D_{tuta})$.

Antes de definir el esquema de cifrado en sí, debemos considerar qué constituye formalmente el espacio de objetos de JavaScript como se definió previamente. Adoptamos el enfoque de conjuntos anidados de pares clave-valor, donde las claves son cadenas y los valores son o bien valores primitivos (cadenas de caracteres, números, booleanos o nulos) u otros objetos de JavaScript de una profundidad menor, para acotar la profundidad máxima de un objeto en su conjunto.

[5] Si bien el estándar recomienda que los objetos tengan claves únicas, la aplicación de esta regla depende del parser. La especificación de JSON establece que: "When the names within an object are not unique, the behavior of software that receives such an object is unpredictable." [Bra17]

```
{
    "name": "Tomás Bertoli",
    "age": 24,
    "isStudent": true,
    "courses": ["CS", "Math"],
    "address": {
        "street": "123 Main St",
        "city": "Springfield"
    }
}
```

```
{
    ("name", "Tomás Bertoli"),
    ("age", 24),
    ("isStudent", true),
    ("courses", {
        ("0", "CS"),
        ("1", "Math")
    }),
    ("address", {
        ("street", "123 Main St"),
        ("city", "Springfield")
    })
}
```

Listing 1: Comparación entre un objeto JavaScript de profundidad 2 en JavaScript nativo y su representación como conjunto. Notar que no todos los valores de los campos deben tener la misma profundidad.

Definición 14 (Objetos JavaScript de profundidad d)**.** *Sea* string *el conjunto de todas las cadenas UTF-8 [Yer03],* number *el conjunto de los números de punto flotante de doble precisión IEEE-754 [19], y $d \in \mathbb{N}_0$ una profundidad de objeto. Entonces, definimos recursivamente el conjunto de objetos de JavaScript de profundidad como máximo d, $\mathcal{JS}_d$, como*

$$\mathcal{JS}_0 = \textit{string} \cup \textit{number} \cup \{\texttt{false}, \texttt{true}\} \cup \{\texttt{null}\}$$

$$\mathcal{JS}_d = \left\{ S \mid \forall (k, v) \in S \left((\nexists (k', v') \in S \, (k = k' \wedge v \neq v')) \wedge k \in \textit{string} \wedge v \in \bigcup_{j=0}^{d-1} \mathcal{JS}_j \right) \right\}$$

Podemos omitir la profundidad d cuando quede clara por el contexto. En esta definición, cada conjunto S es un ejemplo de un objeto de JavaScript que contiene pares clave-valor cuyo componente izquierdo es una cadena de caracteres y cuyo componente derecho es cualquier otro objeto de JavaScript de menor profundidad. En particular, llamamos "profundidad" a la longitud máxima de la secuencia de objetos anidados en un objeto de JavaScript, donde los valores primitivos se consideran de profundidad 0.

Como en JavaScript los arreglos se definen literalmente como objetos con claves en $\mathbb{N}_0$, omitimos la necesidad de incluir una definición especial para ellos; están incluidos en la definición de $\mathcal{JS}$ tal cual. Denotamos $S[k]$ como el valor en el par clave-valor con clave k en el objeto S, y $S[k] = v$ como la asignación del valor v a la clave k en el objeto S. Adicionalmente, tratamos a ϵ como la clave vacía y a $S[k_1 \cdot k_2 \cdots k_n]$ para una secuencia de claves $k_1 \cdot k_2 \cdots k_n$ como la aplicación sucesiva del operador de acceso a pares clave-valor, es decir, $S[k_1 \cdot k_2 \cdots k_n] = S[k_1][k_2] \cdots [k_n]$.

Le recordamos al lector que, aunque mayormente usaremos la nueva sintaxis para referirnos a los objetos de JavaScript, estos todavía se definen como conjuntos de pares clave-valor. Como tal, aún podemos usar la notación $S = \{(k_1, v_1), (k_2, v_2), ...\}$ para denotar un objeto con claves de nivel superior k_1, k_2 y valores v_1 y v_2, respectivamente. Un ejemplo de un objeto de JavaScript que sigue la Definición 14 se muestra en el Listado 1.

Como el recorrido de objetos será una operación recurrente en las siguientes secciones, definimos el algoritmo de recorrido de objetos en anchura (BFS) para que esté completo.

Definición 15 (Recorrido de objetos JavaScript)**.** *Definimos el recorrido de objetos mediante la aplicación del algoritmo de búsqueda en anchura (*BFS*) sobre un objeto de JavaScript no primitivo de profundidad finita. Al recorrer la estructura de S,* BFS(S, f) *devuelve una*

evaluación de la función f para cada valor primitivo, con la ruta de los valores, su clave y el valor mismo como argumentos para f.

```
BFS(S, f)
1 :  P ← EmptyQueue
2 :  O ← {}
3 :
4 :  for (m_k, m_v) ∈ S do
5 :     P.enqueue((ϵ, m_k, m_v))
6 :
7 :  while P ≠ EmptyQueue do
8 :     (p, m_k, m_v) ← P.dequeue()
9 :     if m_v ∈ JS_0
10 :       O[p·m_k] = f(p, m_k, m_v)
11 :    else
12 :       O[p·m_k] = {}
13 :       for (m'_k, m'_v) ∈ m_v do
14 :          P.enqueue((p·m_k, m'_k, m'_v))
15 :  return O
```

En el procedimiento BFS, omitimos la notación y tratamos el objeto de JavaScript como un conjunto, iterando sobre sus elementos (que son pares clave-valor).

La aplicación inmediata de este algoritmo será definir el esquema de cifrado de Tuta. En el Listado 2 se muestra un ejemplo de aplicación del algoritmo BFS. Como es de esperar, la función f solo se aplica a los nodos hoja, es decir, a los valores primitivos, y la ruta se construye concatenando las claves del objeto a medida que lo recorremos.

Definición 16 (SKES $\mathcal{E}_{tuta}$)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado autenticado con espacio de claves $\mathcal{K}$, espacio de mensajes $\{0,1\}^*$ y espacio de textos cifrados $\{0,1\}^*$. Denotemos E_k como la aplicación parcial de la función de cifrado E con la clave k (es decir, $E(k,\cdot)$). Entonces, podemos definir el esquema de cifrado de Tuta $\mathcal{E}_{tuta} = (E_{tuta}, D_{tuta})$ con espacio de mensajes y textos cifrados de objetos $\mathcal{JS}_d \setminus \mathcal{JS}_0$ (es decir, objetos de JavaScript no primitivos de profundidad como máximo d), para una constante $d \in \mathbb{N}_0$.*

```
OmitirClave_F(p, m_k, m_v)
1 :  return F(m_v)
```

```
E_tuta(k, S)
1 :  C ← BFS(S, OmitirClave_{E_k})
2 :  return C
```

```
D_tuta(k, C)
1 :  S ← BFS(C, OmitirClave_{D_k})
2 :  return S
```

En el caso de Tuta, $\mathcal{E}$ es un esquema Encrypt-then-MAC que utiliza AES-256 en modo CBC [ST99] y HMAC [Kra10] basado en SHA-256 [ST15].

5.1 Rompiendo el Esquema de Cifrado de Tuta

Al inspeccionar E_{tuta} en Definición 16, se observa que el esquema no asegura la integridad del objeto como un todo: si bien cada valor primitivo está autenticado por el cifrado autenticado de $\mathcal{E}$, la correspondencia entre valores y sus claves no se verifica. A continuación formalizamos este problema.

```
{
    "name": "Tomás Bertoli",
    "age": 24,
    "isStudent": true,
    "courses": ["CS", "Math"],
    "address": {
        "street": "123 Main St",
        "city": "Springfield"
    }
}
```

```
{
    "name": f(ε, "name", "Tomás Bertoli"),
    "age": f(ε, "age", 24),
    "isStudent": f(ε, "isStudent", true),
    "courses": [
        f("courses", "0", "CS"),
        f("courses", "1", "Math")
    ],
    "address": {
        "street": f(
            "address",
            "street",
            "123 Main St"
        ),
        "city": f(
            "address",
            "city",
            "Springfield"
        )
    }
}
```

Listing 2: Comparación del antes y el después de una aplicación de $\mathsf{BFS}(S, f)$, en notación JavaScript

Teorema 4 ($\mathcal{E}_{tuta}$ no es seguro)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta que proporciona (ε, t, q)-integridad de texto cifrado. Entonces, existe un adversario eficiente que realiza $q = 1$ consulta al oráculo en $t \approx 4$ operaciones básicas y gana el juego de integridad de texto cifrado contra un retador de $\mathcal{E}_{tuta}$, como se define en Definición 16, con probabilidad $1 - \Pr_{EQ}$ donde $\Pr_{EQ}$ es la probabilidad de que el cifrado de dos entradas diferentes produzca el mismo texto cifrado.*

Demostración. Recordemos el juego de *integridad de texto cifrado* presentado en la Definición 5. El objetivo del adversario es producir una falsificación c^* tal que $D_{tuta}(k, c^*) \neq \perp$ y $c^* \neq c$ para todo c devuelto por el oráculo de cifrado $\mathcal{O}$. Presentamos el siguiente atacante $\mathcal{A}$ para el juego de *integridad de texto cifrado*.

$\mathcal{A}^{\mathcal{O}}$

1: $m \leftarrow \{(\texttt{"a"}, 0), (\texttt{"b"}, 1)\}$
2: $c_1 \leftarrow \mathcal{O}(m)$
3: $\{(\texttt{"a"}, c_a), (\texttt{"b"}, c_b)\} \leftarrow c_1$
4: $c_2 \leftarrow \{(\texttt{"a"}, c_b), (\texttt{"b"}, c_a)\}$
5: **return** c_2

Podemos ver que $\mathcal{A}$ solo intercambia los textos cifrados de las dos claves, c_a y c_b, en el objeto devuelto por el oráculo $\mathcal{O}$ e inmediatamente devuelve el nuevo objeto al retador.

Es fácil ver que el ataque de $\mathcal{A}$ tiene éxito en todos los casos, siempre que $c_a \neq c_b$, ya que de lo contrario la falsificación de $\mathcal{A}$ no contaría como tal—c_2 sería igual al c_1 consultado al oráculo. Debido a la forma en que se implementa D_{tuta}, el algoritmo de descifrado del esquema de cifrado subyacente, D, solo se evalúa sobre valores primitivos, a saber, c_a y c_b, sin tener en cuenta a qué clave corresponden. Esto hace que D no arroje errores de descifrado

y, a su vez, que D_{tuta} acepte, aunque los valores hayan sido intercambiados, lo que constituye una falsificación.

Entonces, existe un adversario eficiente para $\mathcal{E}_{tuta}$ que realiza 1 consulta al oráculo y tiene una probabilidad de ganar de $1 - \Pr_{EQ}$. En otras palabras, $\mathcal{E}_{tuta}$ no proporciona integridad del objeto completo. □

Aunque no lo utilizamos en esta descripción simplificada, $\mathcal{E}_{tuta}$ podría ser vulnerable a otros ataques que aprovechen el comportamiento de los formatos JSON. En particular, debido a que Tuta envía objetos cifrados a través de la red en formato JSON, es posible que otros sistemas interpreten los datos de maneras inesperadas. Lamentablemente, no hemos encontrado tales ataques en nuestra investigación. Al romper la integridad del texto cifrado, notamos inmediatamente que $\mathcal{E}_{tuta}$ no proporciona un cifrado autenticado seguro.

5.1.1 Impacto Práctico

En la práctica, este ataque permite a un atacante malicioso (posiblemente un servidor malicioso) alterar los datos de los objetos de formas inesperadas. Habiendo probado diferentes alteraciones en los eventos de Tuta, encontramos que, por ejemplo, intercambiar la hora de inicio y la hora de finalización de un evento hace que este desaparezca del calendario del usuario. En el peor de los casos, hemos observado que los eventos simplemente desaparecen del calendario si se produce un error de tipo u otra inconsistencia.

Aunque esto constituye un ataque de denegación de servicio, ya que el usuario no podrá acceder más a su evento, no podemos decir que sea de alto impacto, ya que el atacante debe ser el propio servidor o un proxy, ambos entidades en las que el usuario confía implícitamente. Esta confianza implícita proviene del hecho de que el usuario elige Tuta como aplicación de calendario y espera que tenga un tiempo de actividad y una disponibilidad razonables; un ataque trivial de denegación de servicio sería entonces que Tuta se negara a autenticar a los usuarios por completo, pero no es algo digno de mención.

Es importante destacar que esta vulnerabilidad no conduce a otros tipos de ataques, hasta donde hemos probado. Gracias a la estructura de gestión de claves de Tuta, cualquier intercambio de valores de campo que pueda ocurrir se limita únicamente a los objetos cifrados con la misma *clave de sesión*. Esto significa que un atacante no puede, por ejemplo, intercambiar el título de un evento con la descripción de otro, ya que están cifrados con claves de sesión diferentes. Proporcionamos un parche de ejemplo para el código fuente de Tuta previo al descifrado en el Apéndice E que demuestra el ataque. Como el cambio se realiza antes del descifrado del objeto, podemos imaginar que el ataque se ejecuta en un servidor malicioso.

5.2 Una Solución Simple

El ataque presentado en el Teorema 4 es posible en parte porque $\mathcal{E}_{tuta}$ mantiene la estructura clave-valor del objeto cifrado, sin la necesaria autenticación de las claves de los objetos. Uno podría resolver este problema de un par de maneras diferentes. El enfoque más simple es abandonar la estructura clave-valor sobre los textos cifrados. Concretamente, proponemos $\mathcal{E}_{sec} = (E_{sec}, D_{sec})$ con $E_{sec} : \mathcal{JS}_d \to \{0,1\}^*$ como en el Figura 8. El algoritmo utiliza un par de funciones serialize y deserialize para 'ver' el objeto de JavaScript como la cadena de bytes que lo constituye y viceversa, permitiendo operar sobre él directamente utilizando un esquema de cifrado binario estándar.

El esquema conformado por E_{sec} y D_{sec} aprovecha al máximo las propiedades de $\mathcal{E}$. Esto se formaliza en el siguiente resultado.

Teorema 5 ($\mathcal{E}_{sec}$ es seguro)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta que proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t, q)$-cifrado autenticado. Entonces, el esquema de cifrado construido*

$E_{sec}(k, m)$	$D_{sec}(k, c)$
1 : $m' \leftarrow \mathsf{serialize}(m)$	1 : $m' \leftarrow D(k, c)$
2 : $c \leftarrow E(k, m')$	2 : $m \leftarrow \mathsf{deserialize}(m')$
3 : **return** c	3 : **return** m

Figura 8: El esquema de cifrado $\mathcal{E}_{sec}$

sobre $\mathcal{E}$, $\mathcal{E}_{sec} = (E_{sec}, D_{sec})$ como se define en el Figura 8, proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t', q)$-cifrado autenticado con $t' \approx t$.

Demostración. La prueba procederá de la siguiente manera. Primero, presentaremos una reducción del juego IND-CPA de $\mathcal{E}_{sec}$ al juego IND-CPA de $\mathcal{E}$, demostrando que el primero es tan fuerte como el segundo. Después de esto, presentaremos una reducción del juego de integridad de texto cifrado de $\mathcal{E}_{sec}$ al juego de $\mathcal{E}$, obteniendo resultados similares. En conjunto, esto mostrará que $\mathcal{E}_{sec}$ proporciona un grado similar de seguridad autenticada que $\mathcal{E}$.

Comenzamos con el adversario $\mathcal{B}_{CPA}$ que juega el juego IND-CPA, desafiado por un retador de $\mathcal{E}$. Construiremos $\mathcal{B}_{CPA}$ de manera que permita a un adversario $\mathcal{A}_{CPA}$ de IND-CPA para $\mathcal{E}_{sec}$ obtener la misma ventaja que obtendría si jugara contra un retador real de $\mathcal{E}_{sec}$, haciendo que la ventaja de $\mathcal{B}_{CPA}$ sea igual a la de $\mathcal{A}_{CPA}$.

$\mathcal{B}^{\mathcal{O}}_{CPA}$	$\widetilde{\mathcal{O}}(m_0, m_1)$
1 : $\hat{b} \leftarrow \mathcal{A}^{\widetilde{\mathcal{O}}}_{CPA}$	1 : $m'_0 \leftarrow \mathsf{serialize}(m_0)$
2 : **return** $\hat{b}$	2 : $m'_1 \leftarrow \mathsf{serialize}(m_1)$
	3 : $c \leftarrow \mathcal{O}(m'_0, m'_1)$
	4 : **return** c

Desde el punto de vista de $\mathcal{A}$, está jugando contra un retador de $\mathcal{E}_{sec}$, ya que $\widetilde{\mathcal{O}}$ emula perfectamente a E_{sec} al convertir adecuadamente los mensajes al formato JSON. Entonces, como $\mathcal{E}$ proporciona seguridad $(\varepsilon_{CPA}, t, q)$-IND-CPA, ningún atacante tendrá una ventaja mayor que ε_{CPA}. De esto se sigue que $\mathcal{A}$ no puede tener una ventaja mayor que ε_{CPA} con un tiempo de ejecución de t, realizando q consultas al oráculo, y por lo tanto $\mathcal{E}_{sec}$ proporciona seguridad $(\varepsilon_{CPA}, t', q)$-IND-CPA, donde $t' \approx t$, difiriendo en el tiempo de ejecución solo por las llamadas a serialize.

Ahora nos centramos en la integridad del texto cifrado de $\mathcal{E}_{sec}$. De manera similar a la prueba IND-CPA anterior, presentamos un adversario $\mathcal{B}_{INT}$ que jugará el juego de integridad de texto cifrado contra un retador de $\mathcal{E}$. Aquí, $\mathcal{B}_{INT}$ emulará el entorno de $\mathcal{E}_{sec}$ para un adversario de integridad de texto cifrado $\mathcal{A}_{INT}$ de $\mathcal{E}_{sec}$.

$\mathcal{B}^{\mathcal{O}}_{INT}$	$\widetilde{\mathcal{O}}(m)$
1 : $c^* \leftarrow \mathcal{A}^{\widetilde{\mathcal{O}}}_{INT}$	1 : $m' \leftarrow \mathsf{serialize}(m)$
2 : **return** c^*	2 : $c \leftarrow \mathcal{O}(m')$
	3 : **return** c

Nuevamente, $\mathcal{B}_{INT}$ emula perfectamente a E_{sec} en $\widetilde{\mathcal{O}}$ con la ayuda del oráculo $\mathcal{O}$, haciendo que $\mathcal{A}_{INT}$ crea que está jugando contra un retador de $\mathcal{E}_{sec}$. Como los textos cifrados en $\mathcal{E}_{sec}$ también son textos cifrados válidos en $\mathcal{E}$, $\mathcal{B}_{INT}$ puede usar la falsificación c^* de $\mathcal{A}_{INT}$ en el juego contra el retador de $\mathcal{E}$ para obtener una falsificación válida. Notamos que un atacante

contra $\mathcal{E}_{sec}$ necesitaría vencer la integridad de texto cifrado de $\mathcal{E}$ para falsificar un mensaje. Como $\mathcal{E}$ proporciona $(\varepsilon_{INT}, t, q)$-integridad de texto cifrado, $\mathcal{A}$ debe tener una ventaja de como máximo ε_{INT}. Por lo tanto, $\mathcal{E}_{sec}$ proporciona $(\varepsilon_{INT}, t', q)$-integridad de texto cifrado, donde $t' \approx t$, difiriendo en el tiempo de ejecución solo por las llamadas a serialize en el oráculo $\widetilde{\mathcal{O}}$.

Dado que $\mathcal{E}_{sec}$ proporciona tanto seguridad $(\varepsilon_{CPA}, t', q)$-IND-CPA como integridad de texto cifrado $(\varepsilon_{INT}, t', q)$, se sigue inmediatamente que $\mathcal{E}_{sec}$ también proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t', q)$-cifrado autenticado. □

Aunque este esquema es seguro, no es satisfactorio para el caso de uso de Tuta, ya que su espacio de texto cifrado no mantiene la estructura clave-valor del objeto original. Además, no está claro cómo podrían fusionar los campos cifrados y no cifrados para su transmisión al servidor. Esto nos lleva a pensar en un esquema más complejo, que mantendría la estructura clave-valor y, al mismo tiempo, proporcionaría cifrado autenticado o una garantía suficientemente similar.

5.3 Manteniendo la Estructura Clave-Valor

Concretamente, queremos construir un SKES que tome objetos de JavaScript como entrada y devuelva un texto cifrado que también sea un objeto de JavaScript. No solo queremos que el espacio de texto cifrado sea el de los objetos de JavaScript, sino también que tengan la misma estructura que los objetos de entrada (por ejemplo, que sigan el mismo Type Model). Como buscamos la seguridad IND-CPA (para el cifrado autenticado), esta nueva consideración de formato nos obliga a tener en cuenta los TypeModels *dentro* del juego. Esto se debe a un ataque ingenuo en el juego IND-CPA simple, donde un adversario que solicita dos mensajes `{"a": 1}` y `{"b": 2}` distinguiría trivialmente sus textos cifrados si se preserva el formato, ya que las claves no estarían cifradas. Estos problemas alientan las siguientes definiciones.

Definición 17 (Estructura de claves y $\mathcal{JS}_\omega$)**.** *Una estructura de claves ω es un conjunto de rutas sobre* string^**, donde cada ruta es una secuencia de claves de cadenas de caracteres que un conjunto en $\mathcal{JS}$ que se ajuste a ω debe tener. Decimos que $\mathcal{JS}_\omega$ es el conjunto de todos los objetos de JavaScript que se ajustan a una estructura de claves ω y, por lo tanto, satisfacen, para cada $S \in \mathcal{JS}_\omega$*

$$\forall p \in \mathsf{string}^* : (S[p] \in \mathcal{JS}_0 \iff p \in \omega)$$

Intuitivamente, la longitud de la estructura de claves $|\omega|$ se define como el número de rutas o, equivalentemente, el número de valores que define la estructura de claves.

Definición 18 (Seguridad IND-CPA$_\omega$)**.** *Fijemos una estructura de claves ω. Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta con $\mathcal{M} = \mathcal{JS}_\omega$. Definimos el juego* IND-CPA$_\omega$ *de la siguiente manera:*

$\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}_\omega}(\mathcal{A}, b)$	$\mathcal{O}(m_0, m_1)$
1: $k \leftarrow\!\!\$\ \mathcal{K}$	1: **assert** $\lvert m_0[p]\rvert = \lvert m_1[p]\rvert\ \forall p \in \omega$
2: $\hat{b} \leftarrow \mathcal{A}^{\mathcal{O}}()$	2: $c \leftarrow \mathsf{BFS}(m_b, E_k)$
3: **return** $\hat{b}$	3: **return** c

*Decimos que $\mathcal{E}$ es (ε, t, q)-*IND-CPA$_\omega$ *seguro si para cada adversario $\mathcal{A}$ que se ejecuta en tiempo t y realiza como máximo q consultas al oráculo de cifrado $\mathcal{O}$, tenemos*

$$\mathsf{Adv}(\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}_\omega}, \mathcal{A}) := \left|\Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}_\omega}(\mathcal{A}, 0) \Rightarrow 1\right] - \Pr\left[\mathsf{Game}_{\mathcal{E}}^{\text{IND-CPA}_\omega}(\mathcal{A}, 1) \Rightarrow 1\right]\right| \leq \varepsilon$$

Esta definición es similar a la definición original de IND-CPA. La diferencia radica en cómo verificamos la longitud: aunque sabemos que m_0 y m_1 se ajustan a la estructura de claves ω, necesitamos verificar que para cada ruta $p \in \omega$ los valores de $m_0[p]$ y $m_1[p]$ tengan la misma longitud. Este nuevo concepto de IND-CPA nos lleva a una modificación del cifrado autenticado regular.

Definición 19 (Cifrado Autenticado-ω)**.** *Sea $\mathcal{E} = (E, D)$ un esquema de cifrado de clave secreta. Decimos que $\mathcal{E}$ es $(\varepsilon_{CPA}, \varepsilon_{INT}, t, q)$-$AE_\omega$ si proporciona seguridad $(\varepsilon_{CPA}, t, q)$-IND-CPA$_\omega$ e integridad de texto cifrado $(\varepsilon_{INT}, t, q)$.*

Nótese que para esta definición de AE, $\mathcal{E}$ solo necesita proporcionar integridad de texto cifrado regular, ya que la estructura de claves y cualquier requisito adicional deben ser fijados por el algoritmo de descifrado D, relajando la necesidad de definir un nuevo juego de integridad.

Definición 20 ($\mathcal{E}_\omega$)**.** *Fijemos una estructura de claves ω. Sea $\mathcal{E} = (E, D)$ un SKES con datos asociados (por ejemplo, un AEAD) con espacio de claves $\mathcal{K}$, y espacios de mensaje, texto cifrado y datos asociados $\mathcal{JS}_0$. Entonces, definimos el esquema de cifrado $\mathcal{E}_\omega = (E_\omega, D_\omega)$ con espacio de claves $\mathcal{K}$, y espacio de texto en claro y texto cifrado $\mathcal{JS}_\omega$, de la siguiente manera:*

$E_\omega(k, S)$

1: $C \leftarrow \mathsf{BFS}(S, \mathsf{Anteponer}_{E_k})$
2: **return** C

$D_\omega(k, C)$

1: $S \leftarrow \mathsf{BFS}(C, \mathsf{Anteponer}_{D_k})$
2: $\mathsf{BFS}(S, \mathsf{VerificarFalla})$
3: **return** S

$\mathsf{Anteponer}_F(p, m_k, m_v)$

1: **return** $F(m_v, p \cdot m_k)$

$\mathsf{VerificarFalla}(p, m_k, m_v)$

1: **if** $m_v = \perp$
2: **fail**

En este caso, adaptamos el cifrado original $\mathcal{E}$ a objetos de JavaScript de manera similar a Tuta, excepto que verificamos la autenticidad de la estructura del mensaje por medio del campo de datos asociados (AD) del AEAD. Por lo tanto, podemos asegurar que cada valor esté donde se supone que debe estar, preservando la integridad del texto cifrado. Si bien esto es suficiente en el caso de objetos de JavaScript, este enfoque no considera la posibilidad de claves duplicadas en el objeto. En particular, este esquema fallaría en el dominio de los objetos JSON. Analizamos este problema en la siguiente sección. Podemos dar fe de las afirmaciones de cifrado autenticado de $\mathcal{E}_\omega$ mediante el siguiente teorema.

Teorema 6 ($\mathcal{E}_\omega$ es un esquema de cifrado autenticado seguro)**.** *Fijemos una estructura de claves ω. Sea $\mathcal{E} = (E, D)$ un SKES que proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t, q)$-AEAD. Entonces, el esquema de cifrado construido sobre $\mathcal{E}$, $\mathcal{E}_\omega = (E_\omega, D_\omega)$ como en Definición 20, proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t', q')$-$AE_\omega$ donde*

$$t' \approx t \quad q' = \frac{q}{|\omega|}$$

Demostración. Comenzamos probando que $\mathcal{E}_\omega$ es $(\varepsilon_{CPA}, t', q)$-IND-CPA$_\omega$ seguro. Sea $\mathcal{A}_{CPA}$ un adversario IND-CPA$_\omega$ contra $\mathcal{E}_\omega$. Entonces, podemos construir un adversario IND-CPA contra $\mathcal{E}$, $\mathcal{B}_{CPA}$, de la siguiente manera:

$\mathcal{B}^{\mathcal{O}}_{CPA}$

1: $\hat{b} \leftarrow \mathcal{A}^{\widetilde{\mathcal{O}}}_{CPA}()$
2: **return** $\hat{b}$

$\widetilde{\mathcal{O}}(m_0, m_1)$

1: $C \leftarrow \{\}$
2: **for** $p \in \omega$
3: $C[p] \leftarrow \mathcal{O}(m_0[p], m_1[p], p)$
4: **return** C

Podemos ver que $\mathcal{B}_{CPA}$ simula el entorno $\mathcal{E}_\omega$ para $\mathcal{A}$ emulando a E_ω, cifrando cada valor de la estructura de claves con el oráculo de cifrado. Esto garantiza la consistencia entre el bit oculto del retador IND-CPA y los valores cifrados, ya que el oráculo de cifrado siempre devolverá el mensaje de la izquierda o el de la derecha, según el bit oculto. Como ambos mensajes m_0 y m_1 están en $\mathcal{JS}_\omega$, la única forma en que un $\mathcal{A}_{CPA}$ puede distinguirlos es por sus valores. Entonces, como una victoria de $\mathcal{A}_{CPA}$ se correlaciona con una victoria de $\mathcal{B}_{CPA}$, $\mathcal{E}_\omega$ es $(\varepsilon_{CPA}, t', q')$-IND-CPA seguro, con $t' \approx t$ y $q' = \frac{q}{|\omega|}$ para tener en cuenta el paso de autenticación y las consultas adicionales por cada valor de ω al oráculo, respectivamente.

Ahora pasamos nuestra atención a la integridad del texto cifrado de $\mathcal{E}_\omega$. Sea $\mathcal{A}_{INT}$ un adversario INT-CTXT contra $\mathcal{E}_\omega$. Construimos $\mathcal{B}_{\text{INT-AD}}$, que juega el $\mathsf{Game}^{\text{INT-AD}}_{\mathcal{E}}$, de la siguiente manera:

$\mathcal{B}^{\mathcal{O}}_{\text{INT-AD}}$

1 : $Q \leftarrow \{\}$
2 : $C^* \leftarrow \mathcal{A}^{\widetilde{\mathcal{O}}}_{INT}()$
3 : $\mathsf{BFS}(C^*, \mathsf{ObtenerIntento})$
4 : // Si win no está definido, perdemos
5 : **return** win

$\mathsf{ObtenerIntento}(p, k, c)$

1 : **if** $(c, p{\cdot}k) \notin Q$
2 : win $\leftarrow (c, p{\cdot}k)$

$\widetilde{\mathcal{O}}(m)$

1 : $C \leftarrow \mathsf{BFS}(m, \mathsf{Consultar})$
2 : **return** C

$\mathsf{Consultar}(p, k, v)$

1 : $c \leftarrow \mathcal{O}(v, p{\cdot}k)$
2 : $Q \leftarrow Q \cup \{(c, p{\cdot}k)\}$
3 : **return** c

Podemos ver que $\mathcal{B}_{\text{INT-AD}}$ simula el entorno $\mathsf{Game}^{\text{INT}}_{\mathcal{E}_\omega}$ para $\mathcal{A}_{INT}$ emulando a E_ω, cifrando cada valor del objeto mensaje con la ayuda del oráculo $\mathcal{O}$. Notablemente, sabemos que si $\mathcal{A}_{INT}$ ha de ganar el juego de integridad de texto cifrado, entonces C^* se descifra correctamente y, como C^* no es igual a una salida del oráculo $\widetilde{\mathcal{O}}$, al menos una y como máximo $|\omega|$ rutas en la estructura de claves de C^* tienen un valor que o no fue devuelto por $\mathcal{O}$, o fue devuelto bajo una ruta diferente. De ello se deduce que todos dichos valores constituyen una falsificación de $\mathcal{E}$ bajo un par valor-ruta nunca antes visto para el retador de INT-AD. Como cada falsificación de $\mathcal{E}_\omega$ debe contener una falsificación válida de $\mathcal{E}$, podemos ver que

$$(\mathsf{Game}^{\text{INT}}_{\mathcal{E}_\omega}(\mathcal{A}_{INT}) \Rightarrow 1) \implies (\mathsf{Game}^{\text{INT-AD}}_{\mathcal{E}}(\mathcal{B}_{\text{INT-AD}}) \Rightarrow 1)$$

Por lo tanto, $\mathcal{E}_\omega$ proporciona $(\varepsilon_{INT}, t', q')$-integridad de texto cifrado, con $t' \approx t$ para tener en cuenta el paso de cifrado y $q' = \frac{q}{|\omega|}$ debido a las consultas al oráculo por cada valor de ω.

Finalmente, podemos concluir que, debido a las garantías de IND-CPA y de integridad de texto cifrado, $\mathcal{E}_\omega$ proporciona $(\varepsilon_{CPA}, \varepsilon_{INT}, t', q')$-cifrado autenticado, como se deseaba. □

5.3.1 Consideraciones Prácticas

Aunque en el caso de Tuta se requieren campos no cifrados, se puede crear fácilmente un esquema que considere campos no cifrados utilizando un MAC sobre el par ruta-valor de la misma manera que ciframos los valores. El único obstáculo en la implementación de este enfoque es el del almacenamiento de la etiqueta, ya que necesitamos que tanto el texto en claro como la etiqueta estén ubicados juntos. Esto se puede lograr simplemente concatenando el valor después de la etiqueta, o incluso almacenando un objeto en lugar del valor primitivo, donde dicho objeto contiene un campo de "valor" con el valor original, y un campo de "etiqueta" con su etiqueta correspondiente. No analizamos estas opciones aquí, ya que requerirían

```python
import json
d = {1: "one", "1": "two"}
s = json.dumps(d)
print(s)  # {"1": "one", "1": "two"}
d2 = json.loads(s)
print(d2)  # {'1': 'two'}
```

Listing 3: El módulo JSON de Python no garantiza que la salida de *json.dumps* seguida de *json.loads* devuelva el diccionario original.

que Tuta cambie algunas de sus rutinas. Por ejemplo, la búsqueda de objetos tendría que cambiar, ya que los ID no están cifrados y estarían estructurados de manera diferente bajo un nuevo esquema. Notamos, sin embargo, que este enfoque es compatible con bases de datos NoSQL, permitiendo que la base de datos almacene el objeto, o una versión codificada de él, tal cual. En conjunto, este enfoque es simple, seguro y transparente para una base de datos de documentos, resolviendo el problema presentado en el Teorema 4 con la misma funcionalidad que $\mathcal{E}_{tuta}$, a excepción de los valores no cifrados.

5.4 Usando el Formato JSON

Como discutimos previamente, el formato JSON presenta peculiaridades que pueden llevar a comportamientos inesperados cuando se usa en sistemas con parsers distintos. En particular, nos preocupa la presencia de claves duplicadas en objetos JSON, ya que esto puede provocar pérdida de datos al convertir la cadena JSON en un objeto. Si bien en JavaScript la combinación generador/parser es biyectiva por diseño del lenguaje, esto no se cumple en general.

Por ejemplo, el módulo *json* de Python no garantiza que una aplicación de *json.dumps* (el generador) seguida de *json.loads* (el parser) devuelva el diccionario original: un diccionario puede tener claves distintas bajo dos tipos diferentes, pero al convertir a JSON las claves se convierten en cadenas de caracteres y, por lo tanto, pueden colisionar. En ese caso, *json.loads* devuelve una única clave con el valor de la última. Concretamente, el Listado 3 ilustra este problema con un ejemplo.

Esto es un problema si queremos considerar el uso del formato JSON para el transporte y, en particular, para el cifrado, ya que los datos pueden perderse ya sea por combinaciones generador/parser entre lenguajes e incluso, en el caso de Python, por combinaciones dentro del mismo lenguaje. Aunque se podría argumentar que es responsabilidad del usuario asegurarse de que no haya claves duplicadas en el objeto JSON, esto no siempre es fácil, como en Python. Además, como JavaScript impone claves únicas en los objetos, uno podría considerar que el formato JSON no es un problema en el caso de Tuta.

A pesar de todo, nos gustaría definir un esquema de cifrado seguro que sea compatible con el formato JSON, independientemente de la combinación particular de generador o parser utilizada. Imaginamos un esquema de cifrado que opera a nivel de objeto (no de JSON). Uno cifraría un objeto y luego usaría un generador de JSON sobre él para transmitirlo (por ejemplo, a través de la red). Después, el objeto sería parseado, y el algoritmo de descifrado se ejecutaría sobre el objeto generado. Luego, devolvería o bien el objeto original con todas sus claves duplicadas (si fue parseado fielmente), o bien, en caso de ignorar claves, el esquema podría devolver el subconjunto parseado del objeto original, o devolver un error, según lo que cada aplicación requiera. Solo consideraremos el caso en que se permite ignorar algunos (pero no todos) los valores para claves duplicadas.

Una construcción que preserva subconjuntos. Definimos intuitivamente la noción de seguridad *SUF-CMA de Subconjuntos* (S-SUF-CMA) para un MAC $\mathcal{M} = (T, V)$ con espacio de mensajes $\mathbb{M}$, donde $\mathbb{M}$ forma un retículo (en el sentido de conjunto parcialmente ordenado) bajo la relación $\subseteq$. La función de verificación V de un MAC seguro S-SUF-CMA debería, para todas las claves k y mensajes etiquetados $S \in \mathbb{M}$ con etiqueta $\tau = T(k, S)$, devolver 1 para todo $S' \subseteq S$, tal que $V(k, S', \tau) = 1$. Es decir, la función de verificación es cerrada bajo la relación de subconjunto.

Definición 21 (SUF-CMA de Subconjuntos)**.** *Sea $\mathcal{M} = (T, V)$ un MAC con espacio de mensajes $\mathbb{M}$ donde $\mathbb{M}$ es un retículo (como en un conjunto parcialmente ordenado) de altura $n < \infty$ bajo la relación $\subseteq$, espacio de claves $\mathcal{K}$ y espacio de etiquetas $\mathcal{T}$. Definimos el juego SUF-CMA de Subconjuntos de la siguiente manera:*

$\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}}(\mathcal{A})$

1: $Q \leftarrow \{\}$
2: $k \xleftarrow{\$} \mathcal{K}$
3: $(M^*, \tau^*) \leftarrow \mathcal{A}^{\mathcal{O}}(k)$
4: **return** $[\![V(M^*, \tau^*) = 1 \wedge \nexists((M, \tau) \in Q)\,(M^* \subseteq M \wedge \tau^* = \tau)]\!]$

$\mathcal{O}(M)$

1: $\tau \leftarrow T(k, M)$
2: $Q \leftarrow Q \cup \{(M, \tau)\}$
3: **return** τ

Decimos que $\mathcal{M}$ proporciona seguridad (ε, t, q)-S-SUF-CMA si para todos los adversarios $\mathcal{A}$ que se ejecutan en tiempo t y realizan como máximo q consultas al oráculo, tienen una ventaja

$$\mathsf{Adv}(\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}}, \mathcal{A}) := \Pr[\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}}(\mathcal{A}) \Rightarrow 1] \leq \varepsilon$$

Presentamos una construcción de un MAC seguro S-SUF-CMA basado en dos MACs seguros SUF-CMA simples.

Definición 22 (MAC seguro S-SUF-CMA básico)**.** *Sean $\mathcal{M}_m = (T_m, V_m)$ un MAC seguro SUF-CMA con espacio de mensajes $\mathbb{M}$, espacio de claves $\mathcal{K}_m$ y espacio de etiquetas $\mathcal{T}_m$, y sea $\mathcal{M}_t = (T_t, V_t)$ un MAC seguro SUF-CMA con espacio de mensajes $\mathcal{T}_m^+$,[6] espacio de claves $\mathcal{K}_t$ y espacio de etiquetas $\mathcal{T}_t$.*

Definimos el MAC seguro S-SUF-CMA $\mathcal{M}_{sub} = (T_{sub}, V_{sub})$ con espacio de mensajes $\mathcal{P}(\mathbb{M}) \setminus \emptyset$, espacio de claves $\mathcal{K}_m \times \mathcal{K}_t$ y espacio de etiquetas $\mathcal{T}_m^+ \times \mathcal{T}_t$ de la siguiente manera:

$T_{sub}(k, M)$

1: $(k_m, k_t) \leftarrow k$
2: $\mathsf{tags} \leftarrow \mathsf{ListaVacía}$
3: **for** $m \in M$ **do**
4: $\quad \mathsf{tags.append}(T_m(k_m, m))$
5: $\tau_t \leftarrow T_t(k_t, \mathsf{tags})$
6: $\tau \leftarrow (\mathsf{tags}, \tau_t)$
7: **return** τ

$V_{sub}(k, M, \tau)$

1: $(k_m, k_t) \leftarrow k$
2: $(\mathsf{tags}, \tau_t) \leftarrow \tau$
3: **if** $V_t(k_t, \mathsf{tags}, \tau_t) = 0$
4: $\quad$ **return** 0
5: **for** $m \in M$ **do**
6: $\quad$ **if** $\nexists(t \in \mathsf{tags})\,(V_m(k_m, m, t) = 1)$
7: $\quad\quad$ **return** 0
8: **return** 1

[6] Tales requisitos se pueden cumplir con $\mathcal{M}_m$ teniendo etiquetas binarias de longitud fija, y $\mathcal{M}_t$ siendo un MAC binario seguro SUF-CMA tradicional, cuya entrada son las etiquetas concatenadas de $\mathcal{M}_m$, y la salida es una única etiqueta en el espacio $\mathcal{T}_t$.

Teorema 7 (Seguridad S-SUF-CMA de $\mathcal{M}_{sub}$)**.** *Si $\mathcal{M}_m$ es (ε_m, t, q)-SUF-CMA seguro y $\mathcal{M}_t$ es (ε_t, t, q)-SUF-CMA seguro, entonces $\mathcal{M}_{sub}$ es $(\varepsilon_{sub}, t_{sub}, q_{sub})$-S-SUF-CMA seguro, donde*

$$\varepsilon_{sub} = n \cdot \varepsilon_m + \varepsilon_t \quad t_{sub} \approx t \quad q_{sub} = nq$$

Demostración. La prueba procederá mostrando primero reducciones separadas del juego S-SUF-CMA a los juegos SUF-CMA de $\mathcal{M}_t$ y $\mathcal{M}_m$. Luego, mostraremos cómo la seguridad de $\mathcal{M}_{sub}$ está acotada por la suma de las ventajas en ambos juegos. Denotamos W_x como el evento de que el adversario correspondiente gane su juego de falsificación contra $\mathcal{M}_x$. De la definición de V_{sub} podemos ver que hay tres tipos posibles de falsificación para una etiqueta dada $(\mathsf{tags}, \tau_t) = \tau = \mathcal{O}(M)$.

Primero, un atacante puede producir una falsificación (M', τ') tal que la etiqueta $\tau' = \tau$ y los mensajes $M' \nsubseteq M$. Si uno intentara reducir tal falsificación a un juego SUF-CMA, solo sería válida para $\mathcal{M}_m$, ya que en el caso de $\mathcal{M}_t$ las etiquetas están intactas, y por lo tanto $\tau' = \tau = \mathcal{O}_t(m)$, y $(\mathsf{tags}, \tau) \in Q_t$ (Q_t como en el juego SUF-CMA de $\mathcal{M}_t$).

Alternativamente, un atacante puede producir una falsificación $(M', (\mathsf{tags}', \tau_t'))$ tal que $M' \subseteq M$, $\mathsf{tags}' \subseteq \mathsf{tags}$ y $\tau_t' \neq \tau_t$. Esta vez, el atacante se centra en $\mathcal{M}_t$, y su falsificación no es válida en el juego SUF-CMA de $\mathcal{M}_m$, ya que M' es un subconjunto de un conjunto previamente consultado, así como las tags.

Finalmente, un atacante puede producir una falsificación (M', τ') tal que $M' \nsubseteq M$ y $\tau' \neq \tau$. Este ataque es una combinación de los ataques mencionados anteriormente, y es válido tanto en los juegos SUF-CMA de $\mathcal{M}_m$ como de $\mathcal{M}_t$. Por lo tanto, si un atacante produce una falsificación en el juego SUF-CMA de Subconjuntos, también debe ser válida para al menos uno de los MACs internos. De alguna manera, reduciendo una falsificación de SUF-CMA de Subconjuntos a una de SUF-CMA, ya sea de $\mathcal{M}_m$ o de $\mathcal{M}_t$.

Ahora mostramos reducciones parciales para ejemplificar los mecanismos reales mediante los cuales un atacante puede obtener una ventaja en sus respectivos juegos. Más tarde, esto jugará un papel en el análisis de la seguridad S-SUF-CMA de $\mathcal{M}_{sub}$ en su conjunto. Los adversarios $\mathcal{B}_m$ y $\mathcal{B}_t$ en Figura 9 reducen el juego S-SUF-CMA al juego SUF-CMA para $\mathcal{M}_m$ y $\mathcal{M}_t$, respectivamente.

Inspeccionando a $\mathcal{B}_m$, se puede verificar que emula perfectamente el $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$ para $\mathcal{A}$. Luego, si $\mathcal{A}$ produce una falsificación que a su vez contiene una falsificación de $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}$, entonces $\mathcal{B}_m$ gana al extraer dicha falsificación. Esto depende de que la salida de $\mathcal{A}$ sea realmente una falsificación de $\mathcal{M}_m$, y no de otro tipo. Una advertencia es que una salida de $\mathcal{A}$ tiene como máximo n intentos diferentes de falsificación de $\mathcal{M}_m$ (es decir, $(m, t) \notin Q_m$), uno por cada uno de los n posibles elementos en el conjunto de salida M^*. Esto degrada la seguridad en un factor de n, ya que $\mathcal{B}_m$ tiene que comprometerse con solo *uno* de los posibles fraudes que proporciona $\mathcal{A}$, pero no todos ellos deben tener éxito para que $\mathcal{A}$ gane el juego S-SUF-CMA, ya que $\mathcal{A}$ solo tiene que proporcionar una *única* falsificación válida en el conjunto. En el peor de los casos, esto significa que $\mathcal{B}_m$ tiene una probabilidad de $\frac{1}{n}$ de elegir el intento de falsificación correcto para ganar $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}$. Por lo tanto,

$$\varepsilon_m \geq \mathsf{Adv}(\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}, \mathcal{B}_m) \geq \frac{1}{n} \Pr[W_{sub} \wedge W_m]$$

Nótese que Q_m es exactamente igual al conjunto de consultas que el retador almacena en $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}$.

En cuanto a $\mathcal{B}_t$, es un envoltorio elemental alrededor de $\mathcal{A}$ que emula $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$. A su vez, $\mathcal{B}_t$ usa a $\mathcal{A}$ para ganar el $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}$ solo si $\mathcal{A}$ produce una salida que efectivamente contiene una falsificación de $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}$, y su ventaja se cuantifica por

$$\Pr[W_{sub} \wedge W_t] = \mathsf{Adv}(\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}, \mathcal{B}_t) \leq \varepsilon_t$$

$\mathcal{B}_m^{\mathcal{O}}$

1 : $Q_m \leftarrow \{\}$
2 : $k_t \leftarrow\!\!\$\ \mathcal{K}_t$
3 : $(M^*, \tau^*) \leftarrow \mathcal{A}^{T_{\mathcal{B}_m}}()$
4 : $(\mathsf{tags}, \tau_t) \leftarrow \tau^*$
5 : **if** $\nexists (m^* \in M^*, t^* \in \mathsf{tags}) : (m^*, t^*) \notin Q_m$
6 : **fail**
7 : $(m^*, t^*) \leftarrow\!\!\$\ \{(m,t) \mid m \in M^* \wedge t \in \mathsf{tags} \wedge (m,t) \notin Q_m\}$
8 : **return** (m^*, t^*)

$T_{\mathcal{B}_m}(M)$

1 : $\mathsf{tags} \leftarrow \mathsf{ListaVacía}$
2 : **for** $m \in M$ **do**
3 : $\tau_m \leftarrow \mathcal{O}(m)$
4 : $\mathsf{tags.append}(\tau_m)$
5 : $Q_m \leftarrow Q_m \cup \{(m, \tau_m)\}$
6 : $\tau_t \leftarrow T_t(k_t, \mathsf{tags})$
7 : $\tau \leftarrow (\mathsf{tags}, \tau_t)$
8 : **return** τ

$\mathcal{B}_t^{\mathcal{O}}$

1 : $Q_t \leftarrow \{\}$
2 : $k_m \leftarrow\!\!\$\ \mathcal{K}_m$
3 : $(M^*, \tau^*) \leftarrow \mathcal{A}^{T_{\mathcal{B}_t}}()$
4 : $(\mathsf{tags}, \tau_t) \leftarrow \tau^*$
5 : **if** $(\mathsf{tags}, \tau_t) \notin Q_t$
6 : **return** (tags, τ_t)
7 : **fail**

$T_{\mathcal{B}_t}(M)$

1 : $\mathsf{tags} \leftarrow \mathsf{ListaVacía}$
2 : **for** $m \in M$ **do**
3 : $\mathsf{tags.append}(T_m(k_m, m))$
4 : $\tau_t \leftarrow \mathcal{O}(\mathsf{tags})$
5 : $Q_t \leftarrow Q_t \cup \{(\mathsf{tags}, \tau_t)\}$
6 : $\tau \leftarrow (\mathsf{tags}, \tau_t)$
7 : **return** τ

Figura 9: Adversarios que reducen $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$ a SUF-CMA

Estas equivalencias se formalizan en Apéndice C, lo que lleva al siguiente resultado.

Lema 5. *Sean $\mathcal{A}$ un adversario de $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$ y sean $\mathcal{B}_m$, $\mathcal{B}_t$ como en Figura 9. Entonces,*

$$(\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}(\mathcal{A}) \Rightarrow 1) \implies (\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}(\mathcal{B}_m) \Rightarrow 1 \vee \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}(\mathcal{B}_t) \Rightarrow 1)$$

Tenemos todas las herramientas necesarias para acotar la ventaja de un adversario $\mathcal{A}$ que juega el juego S-SUF-CMA, a saber, que una falsificación S-SUF-CMA de $\mathcal{M}_{sub}$ *debe* contener ya sea una falsificación de $\mathcal{M}_m$ o de $\mathcal{M}_t$, probado en Apéndice C. Recordemos la definición de ventaja de $\mathsf{Game}^{\text{S-SUF-CMA}}$. Entonces, podemos acotar la ventaja de un atacante mediante

$$\begin{aligned}
\mathsf{Adv}(\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}, \mathcal{A}) &:= \Pr[W_{sub}] = \Pr[W_{sub} \wedge W_t] + \Pr[W_{sub} \wedge \neg W_t] \\
&\qquad\qquad \text{(probabilidad total)} \\
&\leq \varepsilon_t + \Pr[W_{sub} \wedge \neg W_t] \qquad \text{(reducción de } \mathcal{B}_t\text{)} \\
&\leq \varepsilon_t + \Pr[W_{sub} \wedge W_m] \qquad (W_{sub} \wedge \neg W_t \implies W_m, \text{ Lema 5}) \\
&\leq \varepsilon_t + n \cdot \varepsilon_m \qquad \text{(reducción de } \mathcal{B}_m\text{)}
\end{aligned}$$

En cuanto a t_{sub}, se puede ver que una ejecución exitosa del adversario S-SUF-CMA conduce a una falsificación de $\mathcal{M}_m$ o de $\mathcal{M}_t$. Esto también se muestra en los adversarios $\mathcal{B}_{\{m,t\}}$, ya que realizan una única instanciación de $\mathcal{A}$. Por lo tanto, para romper uno de los MACs internos, basta con una única ejecución del adversario y $t_{sub} \approx t$. Por el mismo principio, $q_{sub} = nq$, ya que para tener éxito en un ataque sobre S-SUF-CMA se necesita una falsificación de $\mathcal{M}_m$ (que usa como máximo nq consultas) o una falsificación de $\mathcal{M}_t$ (que usa solo q consultas), por lo que debe haber usado como máximo nq consultas en el peor de los casos. □

Haciendo uso de esta construcción, se puede definir un esquema de cifrado compatible con JSON que sea seguro bajo la noción de seguridad S-SUF-CMA. No presentamos un esquema de cifrado completo, pero damos una intuición sobre el uso de una combinación de un cifrado AEAD y un MAC S-SUF-CMA para este fin. Esencialmente, el procedimiento de cifrado cifraría todos los valores de claves duplicadas por separado con el campo de datos asociados (AD) como en Definición 20. Después, añadiría la etiqueta MAC S-SUF-CMA calculada sobre el conjunto de todos los valores para una clave dada. De esta manera, el objeto estaría protegido contra "ataques de intercambio de claves", además de permitir el análisis sintáctico de subconjuntos de JSON.

6 Conclusión

En conclusión, encontramos dos problemas en el modelo de servidor malicioso con la forma en que está diseñado Tuta. El primero es el ataque de integridad sobre los objetos de Tuta, que permitió la manipulación de campos y puede ser utilizado para ocultar selectivamente eventos en el calendario de un usuario. Luego, tenemos el ataque de precomputación en la autenticación de usuario, que permite a un servidor malicioso eludir eficazmente la seguridad proporcionada por el hash con sal (salting), basándose en su lugar en un simple hash de contraseñas sin sal.

En cuanto a las lecciones aprendidas, encontramos que hay una brecha por analizar entre los formatos de datos y el cifrado, y que se debe prestar especial atención a este asunto al diseñar un sistema cifrado de extremo a extremo. Además, los ataques de precomputación sobre el protocolo de autenticación de usuario nos enseñaron que incluso una práctica estándar como el hash con sal no debe tomarse a la ligera, ya que su interacción con el protocolo bajo un servidor malicioso es difícil de juzgar sin un análisis formal.

6.1 Trabajo Futuro

Sobre el último punto, no encontramos una manera fácil de resolver la vulnerabilidad del protocolo de autenticación de usuario. Se debería realizar un análisis de otros esquemas, ya que una evaluación superficial muestra que este problema no tiene una solución trivial.

Otro trabajo a futuro incluye la formalización completa de un calendario cifrado de extremo a extremo, ya que el campo está poco investigado desde el punto de vista de la seguridad, con el objetivo final de diseñar un esquema prototípico de la misma manera que lo hicieron Backendal et al. con el almacenamiento en la nube y CSS.

Referencias

[19] «IEEE Standard for Floating-Point Arithmetic». En: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), págs. 1-84. DOI: 10.1109/IEEESTD.2019.8766229.

[AG] Proton AG. *Proton Calendar*. Accessed 2025-09-24. URL: https://proton.me/calendar.

[Alb+23] Martin R. Albrecht et al. *Caveat Implementor! Key Recovery Attacks on MEGA*. Cryptology ePrint Archive, Paper 2023/329. 2023. URL: https://eprint.iacr.org/2023/329.

[Alb+24] Martin R. Albrecht et al. *Share with Care: Breaking E2EE in Nextcloud*. Cryptology ePrint Archive, Paper 2024/546. 2024. URL: https://eprint.iacr.org/2024/546.

[Bac+24] Matilda Backendal et al. *A Formal Treatment of End-to-End Encrypted Cloud Storage*. Cryptology ePrint Archive, Paper 2024/989. 2024. URL: https://eprint.iacr.org/2024/989.

[BDK16] Alex Biryukov, Daniel Dinu y Dmitry Khovratovich. «Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications». En: *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2016, págs. 292-302. DOI: 10.1109/EuroSP.2016.31.

[Bel+97] Mihir Bellare et al. «A concrete security treatment of symmetric encryption». En: *Proceedings 38th annual symposium on foundations of computer science*. IEEE. 1997, págs. 394-403.

[BHP23] Matilda Backendal, Miro Haller y Kenneth G. Paterson. «MEGA: Malleable Encryption Goes Awry». En: 2023, págs. 146-163. DOI: 10.1109/SP46215.2023.10179290.

[BR04] Mihir Bellare y Phillip Rogaway. *Code-Based Game-Playing Proofs and the Security of Triple Encryption*. Cryptology ePrint Archive, Paper 2004/331. 2004. URL: https://eprint.iacr.org/2004/331.

[BR06] Mihir Bellare y Phillip Rogaway. «The security of triple encryption and a framework for code-based game-playing proofs». En: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2006, págs. 409-426.

[BR93] Mihir Bellare y Phillip Rogaway. «Random oracles are practical: a paradigm for designing efficient protocols». En: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. Fairfax, Virginia, USA: Association for Computing Machinery, 1993, págs. 62-73. ISBN: 0897916298. DOI: 10.1145/168588.168596. URL: https://doi.org/10.1145/168588.168596.

[Bra17] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dic. de 2017. DOI: 10.17487/RFC8259. URL: https://www.rfc-editor.org/info/rfc8259.

[BS23] Dan Boneh y Victor Shoup. *A Graduate Course in Applied Cryptography*. Self-published, 2023.

[Coh+17] Katriel Cohn-Gordon et al. «A Formal Security Analysis of the Signal Messaging Protocol». En: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017, págs. 451-466. DOI: 10.1109/EuroSP.2017.27.

[DDD07] Cyrus Daboo, Lisa M. Dusseault y Bernard Desruisseaux. *Calendaring Extensions to WebDAV (CalDAV)*. RFC 4791. Mar. de 2007. DOI: 10.17487/RFC4791. URL: https://www.rfc-editor.org/info/rfc4791.

[Dia+12] Mamadou H. Diallo et al. «CloudProtect: Managing Data Privacy in Cloud Applications». En: *2012 IEEE Fifth International Conference on Cloud Computing*. 2012, págs. 303-310. DOI: 10.1109/CLOUD.2012.122.

[Eve76] Gordon C Everest. «Basic data structure models explained with a common example». En: *Proc. Fifth Texas Conference on Computing Systems*. 1976, págs. 18-19.

[FT21] Pooya Farshim y Stefano Tessaro. «Password Hashing and Preprocessing». En: Springer-Verlag, 2021. DOI: 10.1007/978-3-030-77886-6_3.

[Gmba] Tutao GmbH. *Tuta client software Github*. URL: https://github.com/tutao/tutanota.

[Gmbb] Tutao GmbH. *Tuta Security Commitment*. https://web.archive.org/web/20250104183232/https://tuta.com/security. URL: https://web.archive.org/web/20250104183232/https://tuta.com/security.

[KL14] Jonathan Katz y Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. Chapman & Hall/CRC, 2014. ISBN: 1466570261.

[Kra10] Hugo Krawczyk. *Cryptographic Extraction and Key Derivation: The HKDF Scheme*. Cryptology ePrint Archive, Paper 2010/264. 2010. URL: https://eprint.iacr.org/2010/264.

[Nap25] Ernestas Naprys. «19 billion leaked passwords reveal deepening crisis: lazy, reused, and stolen». En: *Cybernews* (2025). URL: https://cybernews.com/security/password-leak-study-unveils-2025-trends-reused-and-lazy/.

[Our] OurCal. *OurCal Calendar App*. Accessed 2025-09-24. URL: https://ourcal.com.

[Rog02] Phillip Rogaway. «Authenticated-encryption with associated-data». En: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, págs. 98-107.

[Ros] Mike Rosulek. *The Joy of Cryptography*. https://joyofcryptography.com. URL: https://joyofcryptography.com.

[San+12] Tahmineh Sanamrad et al. «My Private Google Calendar and GMail.» En: *IEEE Data Eng. Bull.* 35.4 (2012), págs. 83-92.

[Ski] Skiff. *Skiff Product Overview*. Accessed 2025-09-24. URL: https://skiff.com.

[ST15] National Institute of Standards y Technology. «Secure Hash Standard (SHS)». En: (2015). DOI: 10.6028/NIST.FIPS.180-4.

[ST99] National Institute of Standards y Technology. «Advanced Encryption Standard (AES)». En: (1999). DOI: 10.6028/NIST.FIPS.197-upd1.

[VHK20] Péter Vörös, Péter Hudoba y Attila Kiss. «Steganography and Cryptography for User Data in Calendars». En: *Intelligent Information and Database Systems: Recent Developments*. Cham: Springer International Publishing, 2020, págs. 241-252. ISBN: 978-3-030-14132-5. DOI: 10.1007/978-3-030-14132-5_19. URL: https://doi.org/10.1007/978-3-030-14132-5_19.

[VK18] Péter Vörös y Attila Kiss. «OpenWebCrypt—Securing Our Data in Public Cloud». En: *Modern Approaches for Intelligent Information and Database Systems*. Cham: Springer International Publishing, 2018, págs. 479-489. ISBN: 978-3-319-76081-0. DOI: 10.1007/978-3-319-76081-0_41. URL: https://doi.org/10.1007/978-3-319-76081-0_41.

[Yer03] François Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC 3629. Nov. de 2003. DOI: 10.17487/RFC3629. URL: https://www.rfc-editor.org/info/rfc3629.

A Acotando $\Pr\left[\mathsf{Guess}_{pw}\right]$

Lema 6. *Sea $\mathcal{PW}$ un 1-muestreador que proporciona $(\varepsilon, t, q_{RO_1}, 0)$-resistencia a ataques de adivinanza de contraseñas, y sea $\mathcal{A}$ un adversario que juega Game_2 realizando q_{RO_1} consultas al oráculo RO_1. Entonces, $\mathcal{B}$, como se define a continuación, logra una ventaja $\mathsf{Adv}(\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}, \mathcal{B}) = \Pr\left[\mathsf{Guess}_{pw}\right]$ en el juego de adivinanza de contraseñas contra $\mathcal{PW}$.*

$\mathcal{B}^{\mathsf{Test},\mathsf{Cor}}$	$\widetilde{RO_1}(pw')$	$\mathcal{O}(m_0, m_1)$
$b \leftarrow\!\$\ \{0,1\}$	**if** $\mathsf{Test}(pw')$	**return** $E(k, m_b)$
$RO_1 \leftarrow\!\$\ \text{Funcs}[\mathcal{PW}, \mathcal{K}]$	**return** k	
$RO_2 \leftarrow\!\$\ \text{Funcs}[\mathcal{K}, \{0,1\}^{hl}]$	**return** $RO_1(pw')$	
$k \leftarrow\!\$\ \mathcal{K}$		
$h \leftarrow\!\$\ \{0,1\}^{hl}$		
return $\mathcal{A}^{\widetilde{RO_1}, RO_2, \mathcal{O}}(h)$		

Demostración. Comenzamos justificando la correctitud de $\widetilde{RO_1}$ con el concepto de *programación del oráculo aleatorio*, mediante el cual uno puede alterar un oráculo aleatorio dado y reemplazar algunas de sus salidas. Para que el método sea viable, se debe argumentar que el nuevo oráculo ($\widetilde{RO_1}$) sigue siendo un *oráculo aleatorio*, es decir, que es indistinguible de una función aleatoria. En este caso, la única salida reemplazada es la de $RO_1(pw)$, que se sustituye por k. Como k también es uniformemente aleatoria, argumentamos que $\widetilde{RO_1}$ sigue siendo un oráculo aleatorio, por lo que podemos proceder con la prueba.

Con este fin, analizamos el entorno de $\mathcal{A}$ dentro de $\mathcal{B}$. Podemos ver que $\mathcal{B}$ emula perfectamente el Game_2 del Teorema 1, ya que $\widetilde{RO_1}$ mantiene la consistencia de k con respecto a RO_1, es decir, $RO_1(pw) = k$. La consistencia se mantiene a través del oráculo Test, donde si la entrada a $\widetilde{RO_1}$ es efectivamente la contraseña seleccionada por el retador de $\mathcal{B}$, el oráculo devuelve k. Todos los demás elementos al alcance de $\mathcal{A}$ son idénticos a su contraparte en Game_2, por lo que $\mathcal{A}$ no puede distinguir entre ser ejecutado en el contexto de $\mathcal{B}$ y el de Game_2.

En lo que respecta al juego de adivinanza de contraseñas, se puede ver que, si ocurre $Guess_{pw}$, entonces $\mathcal{A}$ consulta a $\widetilde{RO_1}$ y, a su vez, $\widetilde{RO_1}$ consulta al oráculo Test con la contraseña correcta pw y se establece la bandera `win`. A la inversa, si $\mathcal{B}$ ganó el juego de adivinanza de contraseñas, entonces $\mathcal{A}$ debe haber consultado a $\widetilde{RO_1}(pw)$, ya que $\mathcal{B}$ es solo un envoltorio elemental alrededor de $\mathcal{A}$. Por lo tanto, por el Lemma 1, $\mathsf{Adv}(\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}, \mathcal{B}) = \Pr\left[\mathsf{Guess}_{pw}\right] = 2^{-\tilde{\mathbf{H}}_\infty(\mathcal{PW}|\mathcal{Z})}$. □

B Cota AI-INT sobre el Protocolo de Autenticación de Tuta

Esta es una prueba del Teorema 2.

Demostración. Seguimos la base establecida por la prueba del Teorema 1. De manera similar, definimos los dos juegos adicionales Game_2 y Game_3, pero en el contexto de $\mathsf{Game}^{\text{AI-INT-CTXT}}$.

$\mathsf{Game}^{\text{AI-INT-CTXT}}_{\mathcal{E},\mathcal{PW}}$	$\mathsf{Game}_1(\mathcal{A})$	$\mathsf{Game}_2(\mathcal{A})$
$RO_1 \leftarrow\!\$\ \text{Funcs}[\mathcal{PW},\mathcal{K}]$	$RO_1 \leftarrow\!\$\ \text{Funcs}[\mathcal{PW},\mathcal{K}]$	$RO_1 \leftarrow\!\$\ \text{Funcs}[\mathcal{PW},\mathcal{K}]$
$RO_2 \leftarrow\!\$\ \text{Funcs}[\mathcal{K},\mathcal{H}]$	$RO_2 \leftarrow\!\$\ \text{Funcs}[\mathcal{K},\mathcal{H}]$	$RO_2 \leftarrow\!\$\ \text{Funcs}[\mathcal{K},\mathcal{H}]$
$pw \leftarrow\!\$\ \mathcal{PW}$	$pw \leftarrow\!\$\ \mathcal{PW}$	$k \leftarrow\!\$\ \mathcal{K}$
$k \leftarrow RO_1(pw)$	$k \leftarrow\!\$\ RO_1(pw)$	$h \leftarrow\!\$\ \mathcal{H}$
$h \leftarrow RO_2(k)$	$h \leftarrow\!\$\ \mathcal{H}$	$Q \leftarrow \emptyset$
$Q \leftarrow \emptyset$	$Q \leftarrow \emptyset$	$c^* \leftarrow \mathcal{A}^{RO_1,RO_2,\mathcal{O}}(h)$
$c^* \leftarrow \mathcal{A}^{RO_1,RO_2,\mathcal{O}}(h)$	$c^* \leftarrow \mathcal{A}^{RO_1,RO_2,\mathcal{O}}(h)$	$\mathsf{win} \leftarrow D(k,c^*) \neq \perp$
$\mathsf{win} \leftarrow D(k,c^*) \neq \perp$	$\mathsf{win} \leftarrow D(k,c^*) \neq \perp$	$\mathsf{win} \leftarrow \mathsf{win} \wedge c^* \notin Q$
$\mathsf{win} \leftarrow \mathsf{win} \wedge c^* \notin Q$	$\mathsf{win} \leftarrow \mathsf{win} \wedge c^* \notin Q$	**return** $[\![\mathsf{win}]\!]$
return $[\![\mathsf{win}]\!]$	**return** $[\![\mathsf{win}]\!]$	

Como en el Teorema 1, obtenemos las mismas cotas de indistinguibilidad entre el juego AI-INT-CTXT y Game_2, así como entre Game_2 y Game_3 con probabilidades $\Pr[\mathsf{Guess}_k]$ y $\Pr[\mathsf{Guess}_{pw}]$, respectivamente.

$$\Pr[\mathsf{Guess}_k] \leq \frac{q_{RO_2}}{|\mathcal{K}|} \qquad \text{(ver la prueba del Teorema 1)}$$

$$\Pr[\mathsf{Guess}_{pw}] \leq \mathsf{Adv}(\mathsf{Game}^{\text{Guess}}_{\mathcal{PW}}, \mathcal{B}_{\mathsf{Guess}}) \qquad \text{(Lemma 6)}$$

Por lo tanto, podemos usar el lema de la diferencia para acotar la ventaja de $\mathcal{A}$. Denotemos $\Pr[\mathsf{Game}_i(\mathcal{A}) \Rightarrow 1]$ como $\Pr[W_i]$.

$$\begin{aligned}
\mathsf{Adv}(\mathsf{Game}^{\text{AI-INT-CTXT}}_{\mathcal{E},\mathcal{PW}}, \mathcal{A}) &= \left|\Pr[W_1] - \frac{1}{2}\right| = \left|\Pr[W_1] - \Pr[W_2] + \Pr[W_2] - \frac{1}{2}\right| \\
&\leq \Pr[\mathsf{Guess}_k] + \left|\Pr[W_2] - \frac{1}{2}\right| && \text{(lema de la diferencia)} \\
&= \Pr[\mathsf{Guess}_k] + \left|\Pr[W_2] - \Pr[W_3] + \Pr[W_3] - \frac{1}{2}\right| \\
&\leq \Pr[\mathsf{Guess}_k] + \Pr\left[\mathsf{Guess}_{pw}\right] + \mathsf{Adv}(\mathsf{Game}_3, \mathcal{A}) \\
& && \text{(lema de la diferencia)}
\end{aligned}$$

Ahora, argumentamos que Game_3 es indistinguible de $\mathsf{Game}^{\text{INT-CTXT}}_{\mathcal{E}}$, ya que las entradas de $\mathcal{A}$ son completamente aleatorias, no relacionadas con la contraseña o su hash. Y así, la ventaja de $\mathcal{A}$ en Game_3 está acotada por la de un adversario $\mathcal{B}$ en $\mathsf{Game}^{\text{INT-CTXT}}_{\mathcal{E}}$. Entonces, tenemos que

$$\begin{aligned}
\mathsf{Adv}(\mathsf{Game}^{\text{AI-INT-CTXT}}_{\mathcal{E},\mathcal{PW}}, \mathcal{A}) &\leq \Pr[\mathsf{Guess}_k] + \Pr\left[\mathsf{Guess}_{pw}\right] + \mathsf{Adv}(\mathsf{Game}^{\text{INT-CTXT}}_{\mathcal{E}}, \mathcal{B}) \\
&\leq \frac{q_{RO_2}}{|\mathcal{K}|} + \varepsilon_{PG} + \varepsilon_{INT}
\end{aligned}$$

□

C Relacionando $\mathsf{Game}^{\text{S-SUF-CMA}}$ y $\mathsf{Game}^{\text{SUF-CMA}}$

Lema 7. *Sean Q como en Definición 21, Q_t como en Figura 9 y sea (M^*, τ^*) una salida de $\mathcal{A}$. Entonces,*

$$(\forall (M', \tau') \in Q : \tau^* \neq \tau') \iff \tau^* \notin Q_t$$

Demostración. Inspeccionando Q_t, podemos ver que si existe un $\tau^* \in Q_t$, entonces debe estar incluido en Q como el elemento derecho de un par. Inspeccionando Q, podemos ver que todas las salidas de una llamada al oráculo de etiquetado se almacenan en el elemento derecho. Adicionalmente, en $\mathcal{B}_t$, los elementos de Q_t son todas las salidas del oráculo de etiquetado $T_{\mathcal{B}_t}$. Por lo tanto, si un elemento τ^* está en Q_t, debe estar en Q, ya que fue la salida del oráculo de etiquetado. □

Lema 8. *Sean Q como en Definición 21, Q_m como en Figura 9 y sea $(M^*, \tau^* := (tags^*, \tau_t^*))$ una salida de $\mathcal{A}$. Entonces,*

$$(\forall (M', \tau') \in Q : M^* \nsubseteq M') \implies (\exists m \in M^*, t \in tags^* : (m, t) \notin Q_m)$$

Demostración. Inspeccionando $T_{\mathcal{B}_m}$ en Figura 9, se puede ver que todas las etiquetas para $m \in M$ se almacenan en Q_m. Entonces, como $M^* \nsubseteq M'$ para cualquier conjunto de mensajes consultado al oráculo de etiquetado, debe existir al menos un $m^* \in M^*$ que no provino del oráculo de etiquetado. Por construcción, m^* no debe estar incluido en un elemento izquierdo de Q_m, ya que este último solo contiene elementos consultados al oráculo de etiquetado. □

Lema 5. *Sean $\mathcal{A}$ un adversario de $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$ y sean $\mathcal{B}_m$, $\mathcal{B}_t$ como en Figura 9. Entonces,*

$$(\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}(\mathcal{A}) \Rightarrow 1) \implies (\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}(\mathcal{B}_m) \Rightarrow 1 \vee \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}(\mathcal{B}_t) \Rightarrow 1)$$

Demostración. Recordemos que la salida de $\mathcal{A}$ es $(M^*, \tau^* = (tags^*, \tau_t^*))$. Adicionalmente, extraemos la condición de victoria de cada juego de Definición 4 y Definición 21.[7]

$$\begin{aligned}
\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}(\mathcal{A}) &\equiv V_{sub}(k, M^*, \tau^*) \wedge (\nexists((M', \tau') \in Q)\,(M^* \subseteq M' \wedge \tau^* = \tau')) \\
\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}(\mathcal{B}_m) &\equiv \exists m \in M^*, t \in tags^* : V_m(k_m, m, t) \wedge (m, t) \notin Q_m \\
\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}(\mathcal{B}_t) &\equiv V_t(k_t, tags^*, \tau_t^*) \wedge \tau^* \notin Q_t
\end{aligned}$$

Nótese también que, según Definición 22,

$$V_{sub}((k_m, k_t), M, (tags, \tau)) \equiv (\forall m \in M \exists t \in tags : V_m(k_m, m, t)) \wedge V_t(k_t, tags, \tau)$$

Luego, mostramos que si un adversario logra ganar $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$, entonces o gana $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}$ o $\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}$. Comenzamos con la condición de victoria de $\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}$ y dividimos la fórmula en dos mundos: uno donde τ^* fue consultado al oráculo, y otro donde no lo fue.

$$\begin{aligned}
\tau^* \text{ consultado} &\equiv (V_{sub}(k, M^*, \tau^*) \wedge (\forall (M', \tau') \in Q : M^* \nsubseteq M' \vee \tau^* \neq \tau') \wedge \tau^* \in Q_t) \\
&\iff (V_{sub}(k, M^*, \tau^*) \wedge (\forall (M', \tau') \in Q : M^* \nsubseteq M')) && \text{(Lemma 7)} \\
&\implies (V_{sub}(k, M^*, \tau^*) \wedge (\exists m \in M^*, t \in tags^* : (m, t) \notin Q_m)) && \text{(Lemma 8)} \\
&\implies (\forall m \in M^*, \exists t \in tags^* : V_m(k_m, m, t)) \wedge (\exists m \in M^*, t \in tags^* : (m, t) \notin Q_m) && \text{(def.)} \\
&\implies ((\exists m \in M^*, t \in tags^* : V_m(k_m, m, t) \wedge (m, t) \notin Q_m)) \\
&\iff \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}(\mathcal{B}_m) && \text{(def.)}
\end{aligned}$$

[7] Abusamos de la notación y decimos que una función de verificación evalúa a verdadero si y solo si devuelve 1.

$$\begin{aligned}
\tau^* \text{ no consultado} &\equiv (V_{sub}(k, M^*, \tau^*) \land (\forall (M', \tau') \in Q : M^* \not\subseteq M' \lor \tau^* \neq \tau') \land \tau^* \notin Q_t) \\
&\iff (V_{sub}(k, M^*, \tau^*) \land \tau^* \notin Q_t) && \text{(Lemma 7)} \\
&\implies (V_t(k_t, tags^*, \tau_t^*) \land \tau^* \notin Q_t) && \text{(def.)} \\
&\iff \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}(\mathcal{B}_t) && \text{(def.)}
\end{aligned}$$

Ahora, aplicamos los resultados anteriores para terminar el lema

$$\begin{aligned}
\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}}(\mathcal{A}) &\equiv V_{sub}(k, M^*, \tau^*) \land (\nexists (M', \tau') \in Q : M^* \subseteq M' \land \tau^* = \tau') \\
&\iff V_{sub}(k, M^*, \tau^*) \land (\forall (M', \tau') \in Q : M^* \not\subseteq M' \lor \tau^* \neq \tau') \\
&\qquad \lor (\tau^* \in Q_t \land \tau^* \notin Q_t) \\
&\iff \tau^* \text{ consultado} \lor \tau^* \text{ no consultado} \\
&\implies \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m}(\mathcal{B}_m) \lor \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t}(\mathcal{B}_t)
\end{aligned}$$

Y así $(\mathsf{Game}^{\text{S-SUF-CMA}}_{\mathcal{M}_{sub}} \Rightarrow 1) \implies (\mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_m} \Rightarrow 1 \lor \mathsf{Game}^{\text{SUF-CMA}}_{\mathcal{M}_t} \Rightarrow 1)$. □

D Ejemplo de TypeModel


```
"CalendarEvent": {
    "encrypted": true,
    "values": {
        "_id": {
            "final": true,
            "type": "CustomId",
            "cardinality": "One",
            "encrypted": false
        },
        "_ownerEncSessionKey": {
            "final": true,
            "type": "Bytes",
            "cardinality": "ZeroOrOne",
            "encrypted": false
        },
        "_permissions": {
            "final": true,
            "type": "GeneratedId",
            "cardinality": "One",
            "encrypted": false
        },
        "description": {
            "final": false,
            "type": "String",
            "cardinality": "One",
            "encrypted": true
        },
        "startTime": {
            "final": false,
            "type": "Date",
            "cardinality": "One",
            "encrypted": true
        },
        "endTime": {
            "final": false,
            "type": "Date",
            "cardinality": "One",
            "encrypted": true
        },
        "location": {
            "final": false,
            "type": "String",
            "cardinality": "One",
            "encrypted": true
        },
        "summary": {
            "final": false,
            "type": "String",
            "cardinality": "One",
```

```
            "encrypted": true
        }
    },
    "associations": {
        "attendees": {
            "final": false,
            "type": "AGGREGATION",
            "cardinality": "Any",
            "refType": "CalendarEventAttendee",
            "dependency": null
        },
        "organizer": {
            "final": false,
            "type": "AGGREGATION",
            "cardinality": "ZeroOrOne",
            "refType": "EncryptedMailAddress",
            "dependency": null
        },
        "repeatRule": {
            "final": false,
            "type": "AGGREGATION",
            "cardinality": "ZeroOrOne",
            "refType": "CalendarRepeatRule",
            "dependency": null
        }
    },
    "app": "tutanota",
    "version": "83"
}
```

Listing 4: The CalendarEvent type model with its `values` and `associations`. Some fields are omitted for brevity. `final` fields are marked as immutable, and the `encrypted` flag signifies whether the field in question is encrypted by this instance's *session key*

E Código del Ataque de Integridad

```
 diff --git a/src/common/api/worker/rest/EntityRestClient.ts
   ↪ b/src/common/api/worker/rest/EntityRestClient.ts
index 8ca4a4595..7d18b444e 100644
--- a/src/common/api/worker/rest/EntityRestClient.ts
+++ b/src/common/api/worker/rest/EntityRestClient.ts
@@ -1,655 +1,666 @@
 async _handleLoadMultipleResult<T extends SomeEntity>(
     typeRef: TypeRef<T>,
     loadedEntities: Array<any>,
     ownerEncSessionKeyProvider?: OwnerEncSessionKeyProvider,
 ): Promise<Array<T>> {
     const model = await resolveTypeReference(typeRef)

     // PushIdentifier was changed in the system model v43 to encrypt the
      ↪ name.
     // We check here to check the type only once per array and not for each
      ↪ element.
     if (isSameTypeRef(typeRef, PushIdentifierTypeRef)) {
         await promiseMap(loadedEntities, (instance) =>
          ↪ this._crypto.applyMigrations(typeRef, instance), {
             concurrency: 5,
         })
     }

+    if (isSameTypeRef(typeRef, CalendarEventTypeRef)) {
+        for (const entity of loadedEntities) {
+            const event = entity as CalendarEvent
+            const tmp = event.startTime
+            event.startTime = event.endTime
+            event.endTime = tmp
+        }
+    }
+
     return promiseMap(
         loadedEntities,
         (instance) => {
             return this._decryptMapAndMigrate(instance, model,
              ↪ ownerEncSessionKeyProvider)
         },
         { concurrency: 5 },
     )
 }
```

Listing 5: Diff output showing how a malicious server could undermine ciphertext integrity in Tuta's encryption scheme. This sample manipulates objects before decryption, thus this attack can be executed server-side.